Import Essential Modules

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import missingno as msno
from sklearn import preprocessing
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.layers import Dense
from sklearn import metrics
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report,confusion_matrix
from sklearn.model_selection import RandomizedSearchCV, GridSearchCV
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import roc_curve
from sklearn import tree
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.naive_bayes import CategoricalNB
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import SGDClassifier
from sklearn.ensemble import AdaBoostClassifier
import xgboost as xgb
from sklearn.decomposition import PCA
from mlxtend.plotting import plot_decision_regions
from sklearn.tree import export_graphviz
from six import StringIO
from IPython.display import Image
import graphviz
%matplotlib inline
```

```
from google.colab import drive
drive.mount("/content/drive")
```

```
Mounted at /content/drive
```

About the Dataset

```
df = pd.read_csv('/content/drive/MyDrive/data/dataset_sdn.csv')
df.head(10)
```

	dt	switch	src	dst	pktcount	bytecount	dur	dur_nsec	tot_dur	flows	...	pktrate	Pairflow	Protocol	port_nc
0	11425	1	10.0.0.1	10.0.0.8	45304	48294064	100	716000000	1.010000e+11	3	...	451	0	UDP	3
1	11605	1	10.0.0.1	10.0.0.8	126395	134737070	280	734000000	2.810000e+11	2	...	451	0	UDP	4
2	11425	1	10.0.0.2	10.0.0.8	90333	96294978	200	744000000	2.010000e+11	3	...	451	0	UDP	1
3	11425	1	10.0.0.2	10.0.0.8	90333	96294978	200	744000000	2.010000e+11	3	...	451	0	UDP	2
4	11425	1	10.0.0.2	10.0.0.8	90333	96294978	200	744000000	2.010000e+11	3	...	451	0	UDP	3
5	11425	1	10.0.0.2	10.0.0.8	90333	96294978	200	744000000	2.010000e+11	3	...	451	0	UDP	1
6	11425	1	10.0.0.1	10.0.0.8	45304	48294064	100	716000000	1.010000e+11	3	...	451	0	UDP	4
7	11425	1	10.0.0.1	10.0.0.8	45304	48294064	100	716000000	1.010000e+11	3	...	451	0	UDP	1
8	11425	1	10.0.0.1	10.0.0.8	45304	48294064	100	716000000	1.010000e+11	3	...	451	0	UDP	2
9	11425	1	10.0.0.2	10.0.0.8	90333	96294978	200	744000000	2.010000e+11	3	...	451	0	UDP	4

10 rows × 23 columns

Data Preprocessing

Dataset Dimensions

```
print("This Dataset has {} rows and {} columns".format(df.shape[0], df.shape[1]))
```

```
This Dataset has 104345 rows and 23 columns
```

Concise summary of dataset

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 104345 entries, 0 to 104344
Data columns (total 23 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   dt           104345 non-null  int64
 1   switch       104345 non-null  int64
 2   src          104345 non-null  object
 3   dst          104345 non-null  object
 4   pktcount     104345 non-null  int64
 5   bytecount    104345 non-null  int64
 6   dur          104345 non-null  int64
 7   dur_nsec     104345 non-null  int64
 8   tot_dur      104345 non-null  float64
 9   flows        104345 non-null  int64
 10  packetins    104345 non-null  int64
 11  pktperflow   104345 non-null  int64
 12  byteperflow  104345 non-null  int64
 13  pktrate      104345 non-null  int64
 14  Pairflow     104345 non-null  int64
 15  Protocol     104345 non-null  object
 16  port_no      104345 non-null  int64
 17  tx_bytes     104345 non-null  int64
 18  rx_bytes     104345 non-null  int64
 19  tx_kbps      104345 non-null  int64
 20  rx_kbps      103839 non-null  float64
 21  tot_kbps     103839 non-null  float64
 22  label        104345 non-null  int64
dtypes: float64(3), int64(17), object(3)
memory usage: 18.3+ MB
```

Descriptive statistics of dataset

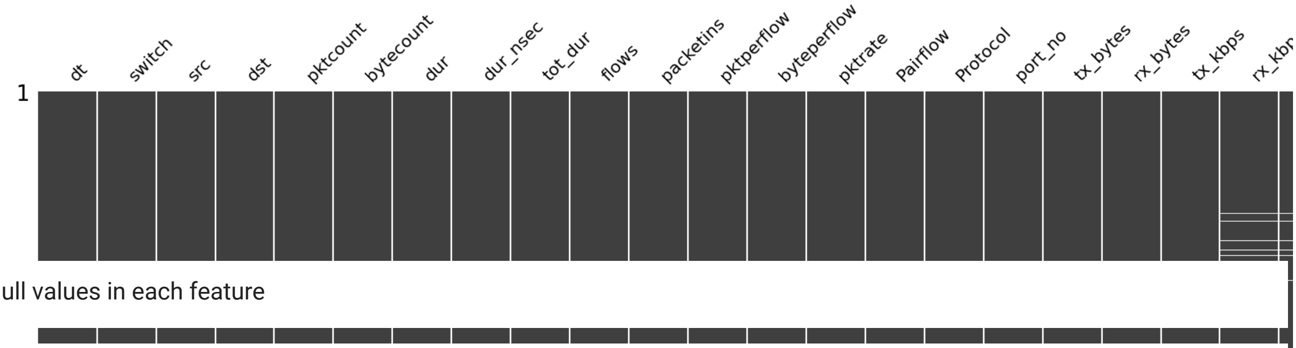
```
df.describe()
```

	dt	switch	pktcount	bytecount	dur	dur_nsec	tot_dur	flows	packetir
count	104345.000000	104345.000000	104345.000000	1.043450e+05	104345.000000	1.043450e+05	1.043450e+05	104345.000000	104345.00000
mean	17927.514169	4.214260	52860.954746	3.818660e+07	321.497398	4.613880e+08	3.218865e+11	5.654234	5200.3834€
std	11977.642655	1.956327	52023.241460	4.877748e+07	283.518232	2.770019e+08	2.834029e+11	2.950036	5257.0014
min	2488.000000	1.000000	0.000000	0.000000e+00	0.000000	0.000000e+00	0.000000e+00	2.000000	4.00000
25%	7098.000000	3.000000	808.000000	7.957600e+04	127.000000	2.340000e+08	1.270000e+11	3.000000	1943.00000
50%	11905.000000	4.000000	42828.000000	6.471930e+06	251.000000	4.180000e+08	2.520000e+11	5.000000	3024.00000
75%	29952.000000	5.000000	94796.000000	7.620354e+07	412.000000	7.030000e+08	4.130000e+11	7.000000	7462.00000
max	42935.000000	10.000000	260006.000000	1.471280e+08	1881.000000	9.990000e+08	1.880000e+12	17.000000	25224.00000

heatmap of missing values

```
msno.matrix(df)
```

```
<Axes: >
```



Count of null values in each feature

```
df.isnull().sum()
```

```
dt              0
switch          0
src             0
dst             0
pktcount        0
bytecount       0
dur             0
dur_nsec        0
tot_dur         0
flows           0
packetins       0
pktperflow      0
byteperflow     0
pktrate         0
Pairflow        0
Protocol        0
port_no         0
tx_bytes        0
rx_bytes        0
tx_kbps         0
rx_kbps       506
tot_kbps      506
label           0
dtype: int64
```

```
(df.isnull().sum()/df.isnull().count())*100
```

```
dt             0.00000
switch         0.00000
src            0.00000
dst            0.00000
pktcount       0.00000
bytecount      0.00000
dur            0.00000
dur_nsec       0.00000
tot_dur        0.00000
flows          0.00000
packetins      0.00000
pktperflow     0.00000
byteperflow    0.00000
pktrate        0.00000
Pairflow       0.00000
Protocol       0.00000
port_no        0.00000
tx_bytes       0.00000
rx_bytes       0.00000
tx_kbps        0.00000
rx_kbps        0.48493
tot_kbps       0.48493
label          0.00000
dtype: float64
```

Drop rows with null values

```
df.dropna(inplace=True)
```

Info after handling Null Values

```
print(df.isnull().sum())
print("This Dataframe has {} rows and {} columns after removing null values".format(df.shape[0], df.shape[1]))
```

```
dt            0
switch        0
src           0
```

```
dst            0
pktcount       0
bytecount      0
dur            0
dur_nsec       0
tot_dur        0
flows          0
packetins      0
pktperflow     0
byteperflow    0
pktrate        0
Pairflow       0
Protocol       0
port_no        0
tx_bytes       0
rx_bytes       0
tx_kbps        0
rx_kbps        0
tot_kbps       0
label          0
dtype: int64
This Dataframe has 103839 rows and 23 columns after removing null values
```

Distribution of Target Class

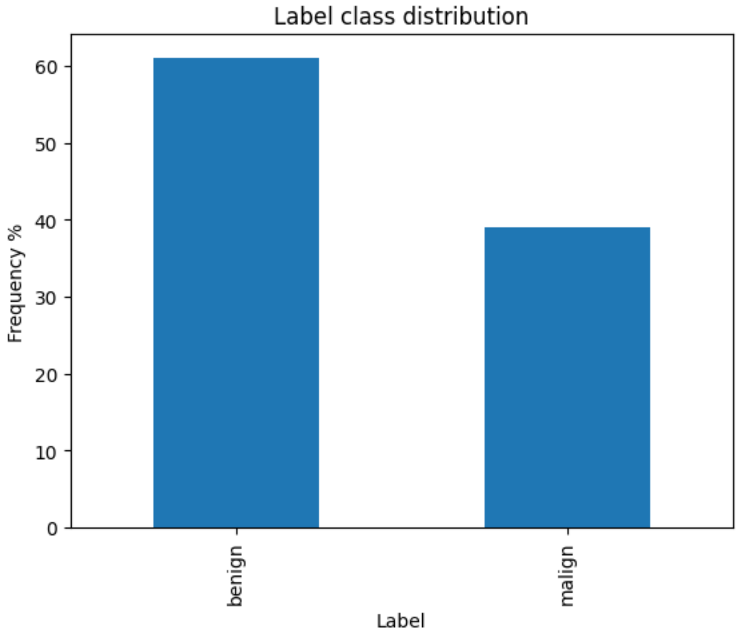
```
malign = df[df['label'] == 1]
benign = df[df['label'] == 0]

print('Number of DDOS attacks that has occured :',round((len(malign)/df.shape[0])*100,2),'%')
print('Number of DDOS attacks that has not occured :',round((len(benign)/df.shape[0])*100,2),'%')
```

```
Number of DDOS attacks that has occured : 39.01 %
Number of DDOS attacks that has not occured : 60.99 %
```

Barplot of Target Class

```
# Let's plot the Label class against the Frequency
labels = ['benign','malign']
classes = pd.value_counts(df['label'], sort = True) / df['label'].count() *100
classes.plot(kind = 'bar')
plt.title("Label class distribution")
plt.xticks(range(2), labels)
plt.xlabel("Label")
plt.ylabel("Frequency %")
```

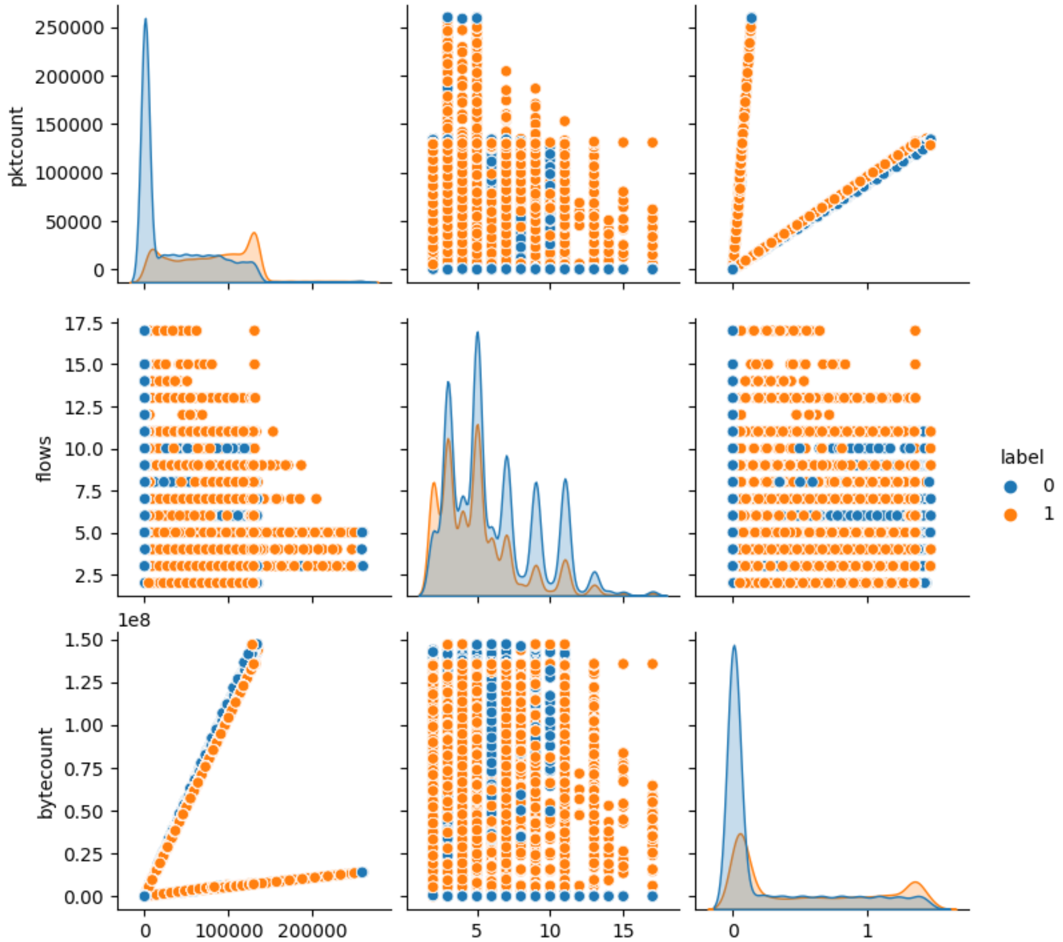
```
Text(0, 0.5, 'Frequency %')
```



Pairplot of select features

```
sns.pairplot(df,hue="label",vars=['pktcount','flows','bytecount'])
```

```
<seaborn.axisgrid.PairGrid at 0x7d8aa937d2d0>
```



Columns in the dataset

```
df.columns
```

```
Index(['dt', 'switch', 'src', 'dst', 'pktcount', 'bytecount', 'dur',
       'dur_nsec', 'tot_dur', 'flows', 'packetins', 'pktperflow',
       'byteperflow', 'pktrate', 'Pairflow', 'Protocol', 'port_no', 'tx_bytes',
       'rx_bytes', 'tx_kbps', 'rx_kbps', 'tot_kbps', 'label'],
      dtype='object')
```

Unique values in each column

```
print(df.apply(lambda col: col.unique()))
```

```
dt             [11425, 11605, 11455, 11515, 9906, 11335, 1157...
switch                                [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
src            [10.0.0.1, 10.0.0.2, 10.0.0.4, 10.0.0.10, 10.0...
dst            [10.0.0.8, 10.0.0.7, 10.0.0.3, 10.0.0.5, 10.0....
pktcount       [45304, 126395, 90333, 103866, 85676, 32914, 4...
bytecount      [48294064, 134737070, 96294978, 110721156, 913...
dur            [100, 280, 200, 230, 190, 73, 10, 250, 80, 260...
dur_nsec       [716000000, 734000000, 744000000, 747000000, 7...
tot_dur        [101000000000.0, 281000000000.0, 201000000000....
flows          [3, 2, 4, 5, 6, 7, 8, 11, 9, 10, 13, 15, 17, 1...
packetins      [1943, 1931, 1790, 1306, 1910, 2242, 2175, 110...
pktperflow     [13535, 13531, 13534, 13533, 13306, 13385, 0, ...
byteperflow    [14428310, 14424046, 14427244, 14426178, 14184...
pktrate        [451, 443, 446, 0, 288, 450, 448, 449, 455, 14...
Pairflow                                                   [0, 1]
Protocol                                         [UDP, TCP, ICMP]
port_no                                           [3, 4, 1, 2, 5]
tx_bytes       [143928631, 3842, 3795, 3688, 3413, 3665, 3775...
rx_bytes       [3917, 3520, 1242, 1492, 3665, 1402, 3413, 429...
tx_kbps        [0, 16578, 19164, 12831, 7676, 10271, 2587, 16...
rx_kbps        [0.0, 6307.0, 3838.0, 6400.0, 7676.0, 10271.0,...
tot_kbps       [0.0, 16578.0, 19164.0, 6307.0, 3838.0, 6400.0...
label                                                      [0, 1]
dtype: object
```

Numerical Features

```
numerical_features = [feature for feature in df.columns if df[feature].dtypes != 'O']
print("The number of numerical features is",len(numerical_features),"and they are : \n",numerical_features)
```

```
The number of numerical features is 20 and they are :
 ['dt', 'switch', 'pktcount', 'bytecount', 'dur', 'dur_nsec', 'tot_dur', 'flows', 'packetins', 'pktperflow', 'byteperflow', 'pktrat
```

Categorical Features

```
categorical_features = [feature for feature in df.columns if df[feature].dtypes == 'O']
print("The number of categorical features is",len(categorical_features),"and they are : \n",categorical_features)
```

```
The number of categorical features is 3 and they are :
 ['src', 'dst', 'Protocol']
```

Number of Unique values in the numerical features

```
# number of unique values in each numerical variable
df[numerical_features].nunique(axis=0)
```

```
dt              858
switch           10
pktcount       9044
bytecount      9270
dur             840
dur_nsec       1000
tot_dur        4183
flows            15
packetins       168
pktperflow     2092
byteperflow    2793
pktrate         446
Pairflow          2
port_no           5
tx_bytes      12257
rx_bytes      11623
tx_kbps        1800
rx_kbps        1730
tot_kbps       2259
label             2
dtype: int64
```

Discrete numerical features

```
#discrete numerical features
discrete_feature = [feature for feature in numerical_features if df[feature].nunique()<=15 and feature != 'label']
print("The number of discrete features is",len(discrete_feature),"and they are : \n",discrete_feature)
```

```
The number of discrete features is 4 and they are :
 ['switch', 'flows', 'Pairflow', 'port_no']
```

```
df[discrete_feature].head(10)
```

	switch	flows	Pairflow	port_no
0	1	3	0	3
1	1	2	0	4
2	1	3	0	1
3	1	3	0	2
4	1	3	0	3
5	1	3	0	1
6	1	3	0	4
7	1	3	0	1
8	1	3	0	2
9	1	3	0	4

Continuous features

```
continuous_feature=[feature for feature in numerical_features if feature not in discrete_feature + ['label']]
print("The number of continuous_feature features is",len(continuous_feature),"and they are : \n",continuous_feature)
```

```
The number of continuous_feature features is 15 and they are :
 ['dt', 'pktcount', 'bytecount', 'dur', 'dur_nsec', 'tot_dur', 'packetins', 'pktperflow', 'byteperflow', 'pktrate', 'tx_bytes', 'rx_
```

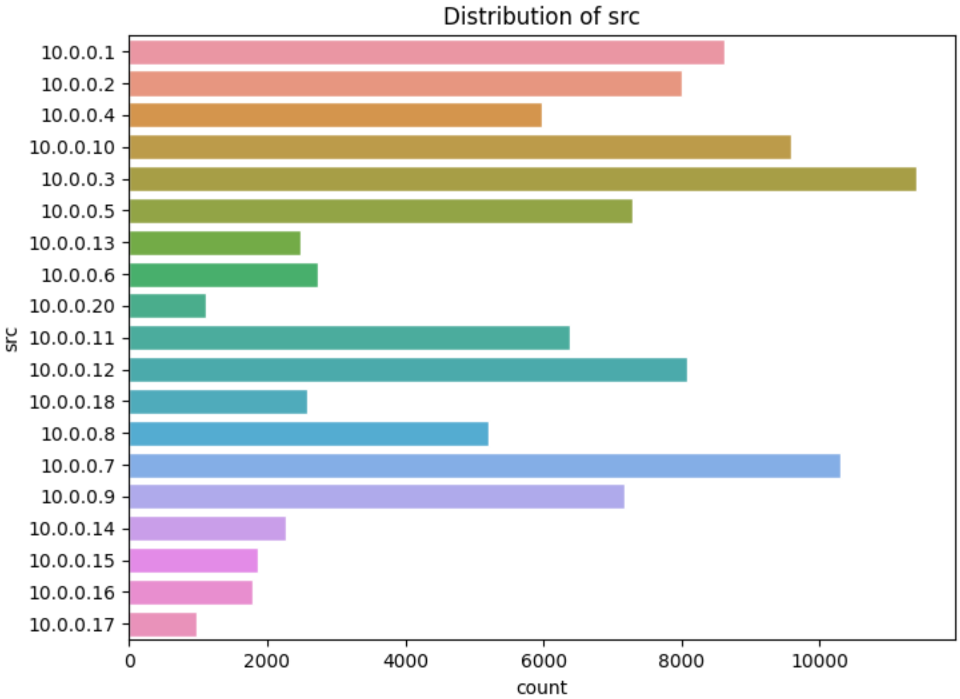
Exploratory Data Analysis

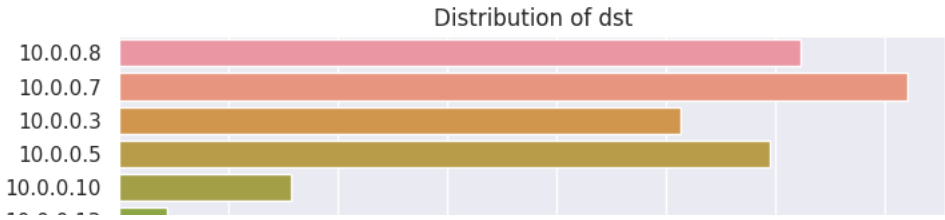
Plotting function definition

```
def countplot_distribution(col):
    sns.set_theme(style="darkgrid")
    sns.countplot(y=col, data=df).set(title = 'Distribution of ' + col)

def histplot_distribution(col):
    sns.set_theme(style="darkgrid")
    sns.histplot(data=df,x=col, kde=True,color="red").set(title = 'Distribution of ' + col)
```

Visualize the distribution of Categorical features

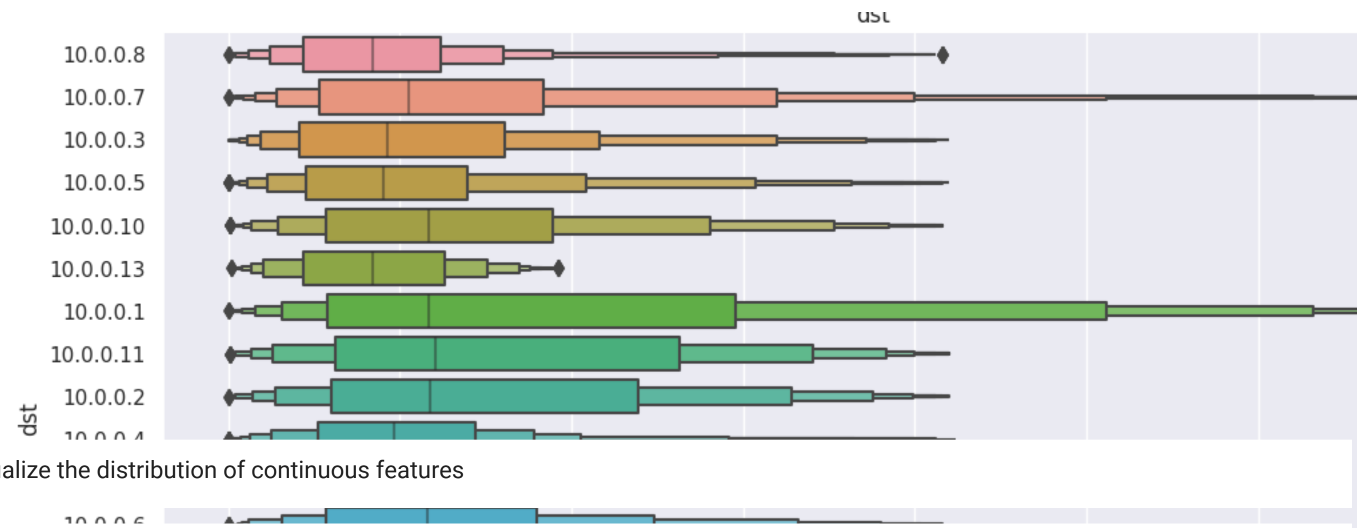
```
## Lets analyse the categorical values by creating histograms to understand the distribution
f = plt.figure(figsize=(8,20))
for i in range(len(categorical_features)):
    f.add_subplot(len(categorical_features), 1, i+1)
    countplot_distribution(categorical_features[i])
plt.show()
```





Visualize the quartiles of categorical features wrt total duration


```
for i in range(len(categorical_features)):
    g = sns.catplot(data=df,x="tot_dur",y=categorical_features[i],kind="boxen").set(title = categorical_features[i])
    g.fig.set_figheight(7)
    g.fig.set_figwidth(15)
```



Visualize the distribution of continuous features

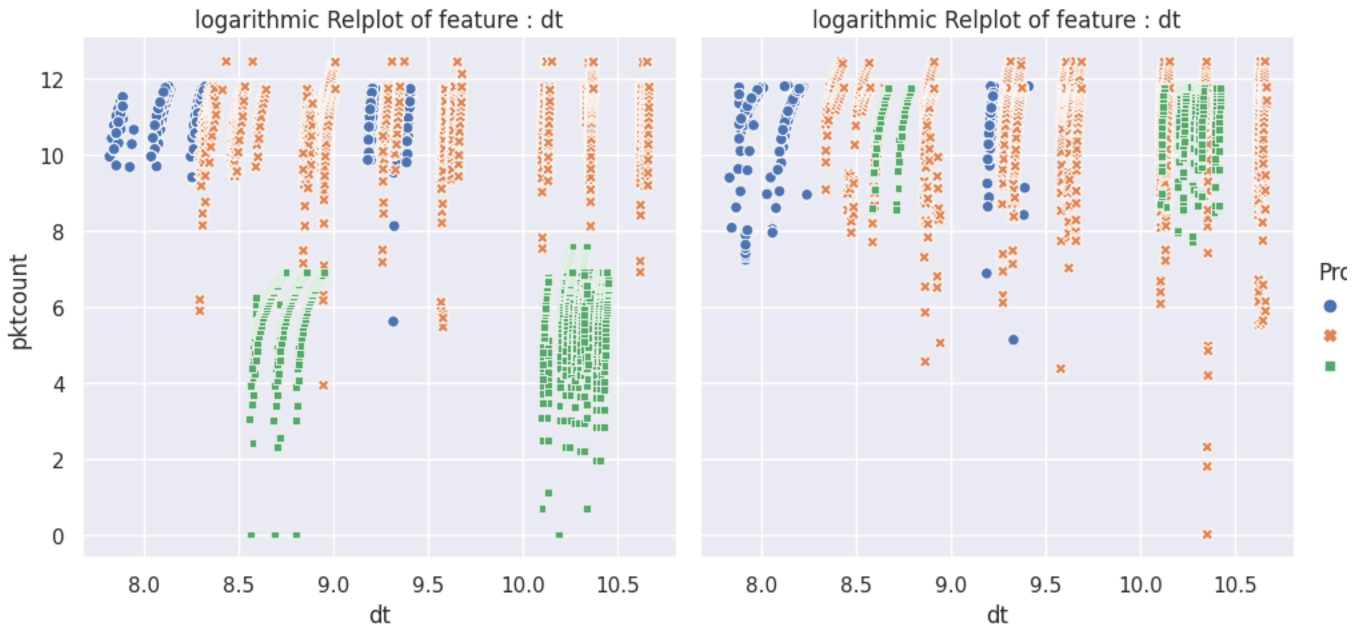
```
## Lets analyse the continuous values by creating histograms to understand the distribution
f = plt.figure(figsize=(20,90))
for i in range(len(continuous_feature)):
    f.add_subplot(len(continuous_feature), 2, i+1)
    histplot_distribution(continuous_feature[i])
plt.show()
```


Distribution of dt
dt
Count
Distribution of pktcount
pktcount
Distribution of bytecount
bytecount
1e8
Distribution of dur
dur
Distribution of dur_nsec
dur_nsec
1e9
Distribution of tot_dur
tot_dur
Distribution of packetins
packetins
Distribution of pktperflow
pktperflow
Distribution of byteperflow
Distribution of pktrate

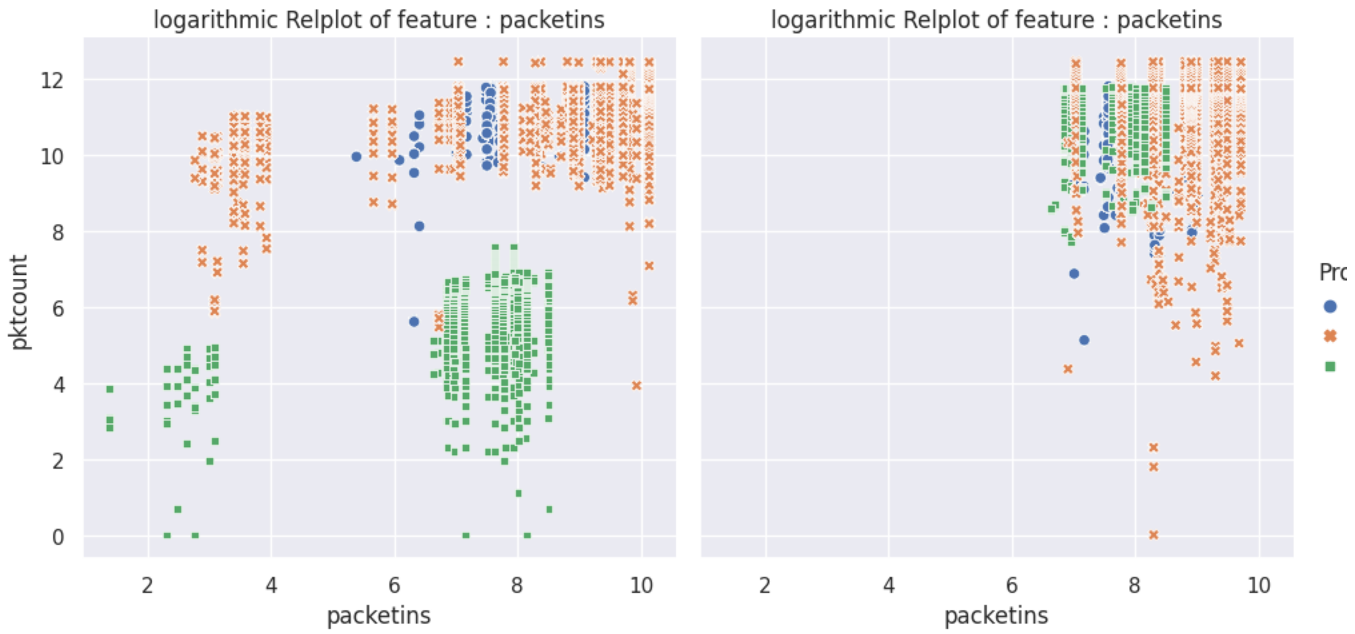
Visualize the distribution of continuous features wrt packet count, protocol and type of attack

```
## Relplot of log(variable)
import warnings
warnings.filterwarnings("ignore")
for feature in continuous_feature:
    data=df.copy()
    if 0 in data[feature].unique():
        pass
    else:
        data[feature]=np.log(data[feature])
        data['pktcount']=np.log(data['pktcount'])
        plt.figure(figsize=(20,20))
        sns.relplot(data=data, x=data[feature],y=data['pktcount'],hue="Protocol",style="Protocol",
                    col="label",kind="scatter").set(title="logarithmic Relplot of feature : " + feature)
```

<Figure size 2000x2000 with 0 Axes>



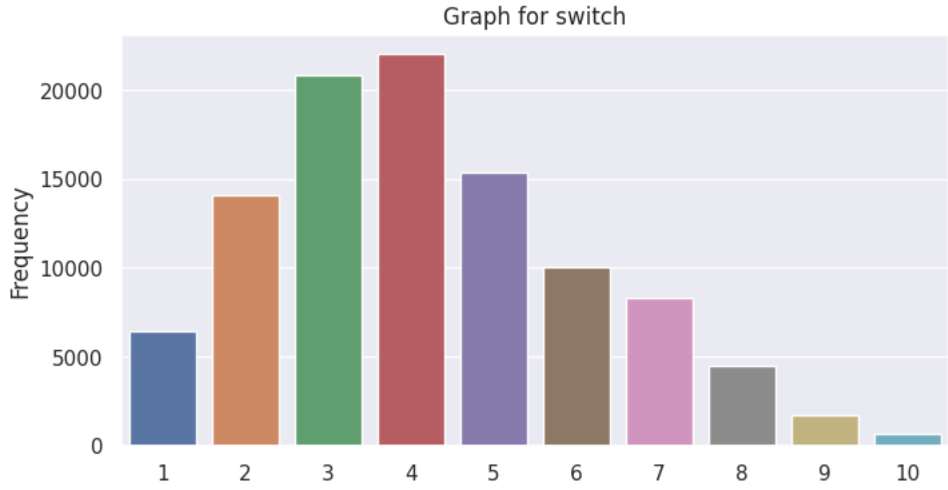
<Figure size 2000x2000 with 0 Axes>

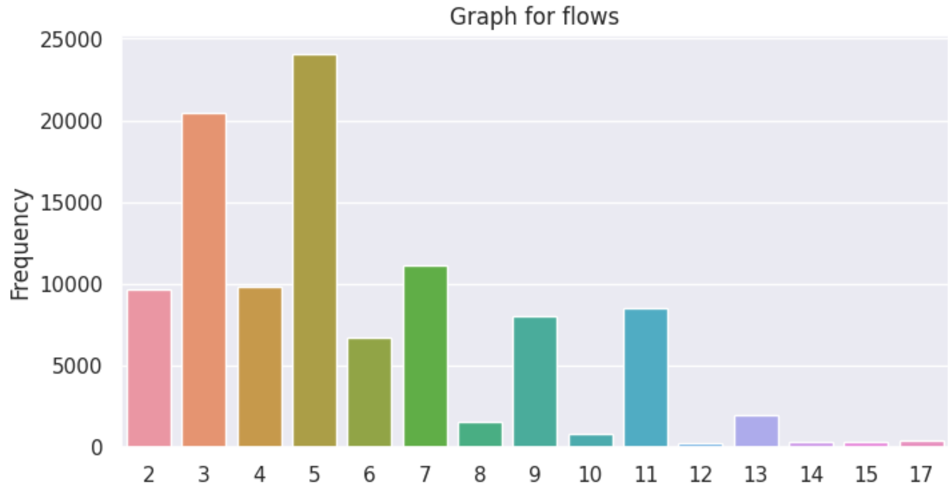


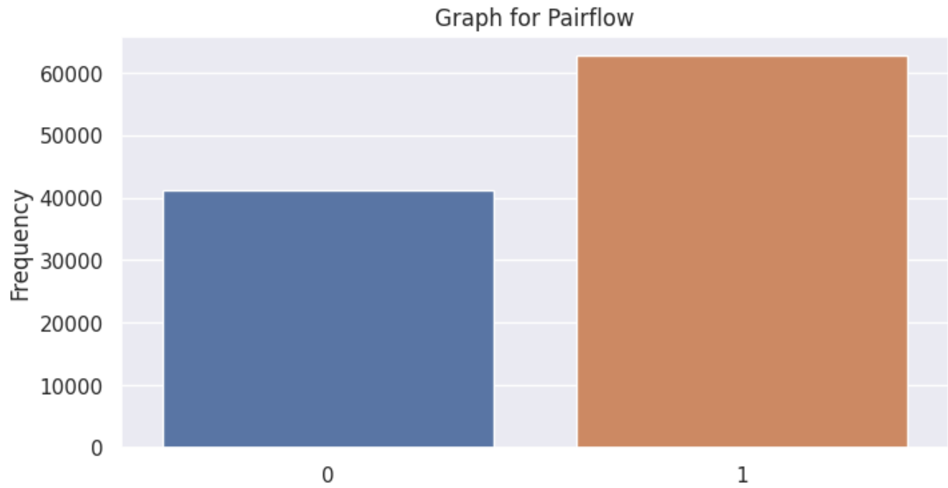
<Figure size 2000x2000 with 0 Axes>

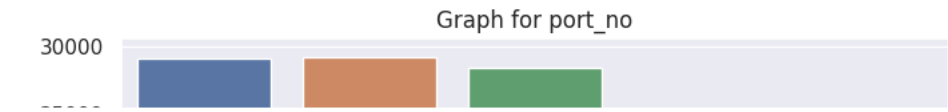
Visualize the distribution of numerical discrete features

```
for feature in discrete_feature:
    plt.figure(figsize=(8,4))
    cat_num = df[feature].value_counts()
    sns.barplot(x=cat_num.index, y = cat_num).set(title = "Graph for "+feature, ylabel="Frequency")
    plt.show()
```







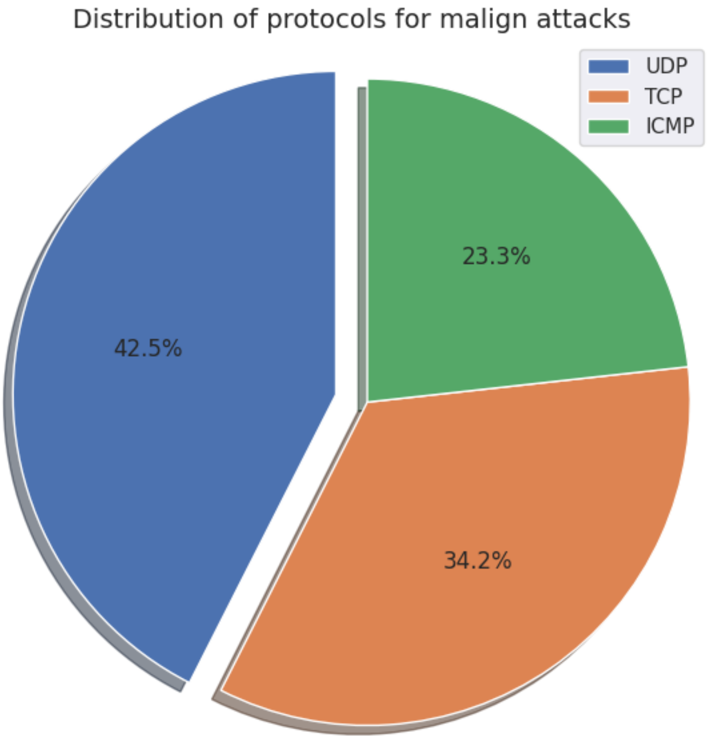



```
def get_percentage_malign_protocols():
    arr = [x for x, y in zip(df['Protocol'], df['label']) if y == 1]
    perc_arr = []
    for i in ['UDP','TCP','ICMP']:
        perc_arr.append(arr.count(i)/len(arr) *100)
    return perc_arr
```

Distribution of protocols for malign attacks

```
fig1, ax1 = plt.subplots(figsize=[7,7])
ax1.pie(get_percentage_malign_protocols(), explode=(0.1, 0, 0), autopct='%1.1f%%',
        shadow=True, startangle=90)
ax1.axis('equal')
ax1.legend(['UDP', 'TCP', 'ICMP'],loc="best")
```

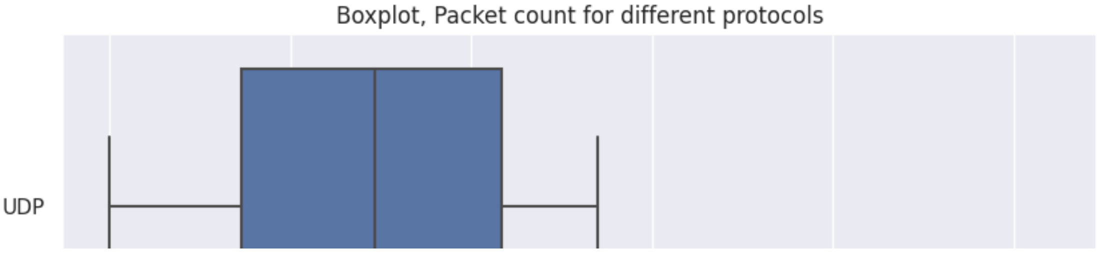
```
plt.title('Distribution of protocols for malign attacks',fontsize = 14)
plt.show()
```



Checking for outliers in Packet count feature

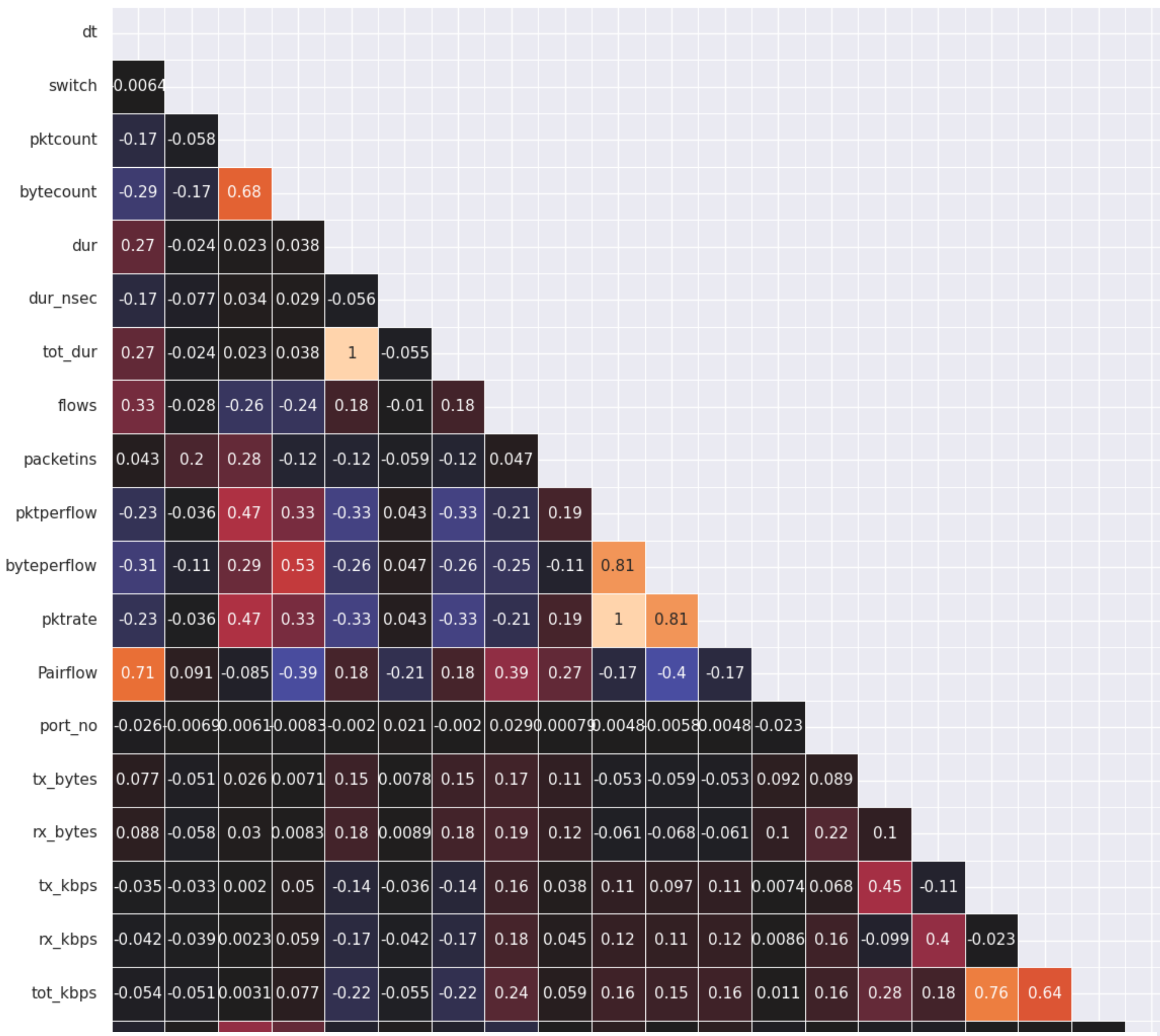
```
fig, ax = plt.subplots(figsize=[10, 10])
sns.boxplot(
    data=df,
    x='pktcount',
    y='Protocol'
)
ax.set_title('Boxplot, Packet count for different protocols')
```

```
Text(0.5, 1.0, 'Boxplot, Packet count for different protocols')
```



Heat map of correlation of features

```
correlation_matrix = df.corr()
fig = plt.figure(figsize=(17,17))
mask = np.zeros_like(correlation_matrix, dtype=np.bool)
mask[np.triu_indices_from(mask)]= True
sns.set_theme(style="darkgrid")
ax = sns.heatmap(correlation_matrix,square = True,annot=True,center=0,vmin=-1,linewidths = .5,annot_kws = {"size": 11},mask = mask)
ax.set_xticklabels(ax.get_xticklabels(),rotation=45, horizontalalignment='right');
plt.show()
```




```
print("Features which need to be encoded are : \n" ,categorical_features)
```

```
Features which need to be encoded are :
 ['src', 'dst', 'Protocol']
```

Encoding categorical features

```
df = pd.get_dummies(df, columns=categorical_features,drop_first=True)
print("This Dataframe has {} rows and {} columns after encoding".format(df.shape[0], df.shape[1]))
```

```
This Dataframe has 103839 rows and 57 columns after encoding
```

```
#dataframe after encoding
df.head(10)
```

	dt	switch	pktcount	bytecount	dur	dur_nsec	tot_dur	flows	packetins	pktperflow	...	dst_10.0.0.2	dst_10.0.0.3	ds
0	11425	1	45304	48294064	100	716000000	1.010000e+11	3	1943	13535	...	0	0	
1	11605	1	126395	134737070	280	734000000	2.810000e+11	2	1943	13531	...	0	0	
2	11425	1	90333	96294978	200	744000000	2.010000e+11	3	1943	13534	...	0	0	
3	11425	1	90333	96294978	200	744000000	2.010000e+11	3	1943	13534	...	0	0	
4	11425	1	90333	96294978	200	744000000	2.010000e+11	3	1943	13534	...	0	0	
5	11425	1	90333	96294978	200	744000000	2.010000e+11	3	1943	13534	...	0	0	
6	11425	1	45304	48294064	100	716000000	1.010000e+11	3	1943	13535	...	0	0	
7	11425	1	45304	48294064	100	716000000	1.010000e+11	3	1943	13535	...	0	0	
8	11425	1	45304	48294064	100	716000000	1.010000e+11	3	1943	13535	...	0	0	
9	11425	1	90333	96294978	200	744000000	2.010000e+11	3	1943	13534	...	0	0	

10 rows × 57 columns

```
df.dtypes
```

```
dt                int64
switch            int64
pktcount          int64
bytecount         int64
dur               int64
dur_nsec          int64
tot_dur         float64
flows             int64
packetins         int64
pktperflow        int64
byteperflow       int64
pktrate           int64
Pairflow          int64
port_no           int64
tx_bytes          int64
rx_bytes          int64
tx_kbps           int64
rx_kbps         float64
tot_kbps        float64
label             int64
src_10.0.0.10     uint8
src_10.0.0.11     uint8
src_10.0.0.12     uint8
src_10.0.0.13     uint8
src_10.0.0.14     uint8
src_10.0.0.15     uint8
src_10.0.0.16     uint8
src_10.0.0.17     uint8
src_10.0.0.18     uint8
src_10.0.0.2      uint8
src_10.0.0.20     uint8
src_10.0.0.3      uint8
src_10.0.0.4      uint8
src_10.0.0.5      uint8
src_10.0.0.6      uint8
src_10.0.0.7      uint8
src_10.0.0.8      uint8
src_10.0.0.9      uint8
dst_10.0.0.10     uint8
dst_10.0.0.11     uint8
dst_10.0.0.12     uint8
dst_10.0.0.13     uint8
dst_10.0.0.14     uint8
dst_10.0.0.15     uint8
dst_10.0.0.16     uint8
dst_10.0.0.17     uint8
dst_10.0.0.18     uint8
dst_10.0.0.2      uint8
dst_10.0.0.3      uint8
```

```
dst_10.0.0.4      uint8
dst_10.0.0.5      uint8
dst_10.0.0.6      uint8
dst_10.0.0.7      uint8
dst_10.0.0.8      uint8
dst_10.0.0.9      uint8
Protocol_TCP      uint8
Protocol_UDP      uint8
dtype: object
```

Split into Independent and dependent variables

```
#separating input and output attributes
x = df.drop(['label'], axis=1)
y = df['label']
```

Normalizing features

```
ms = MinMaxScaler()
x = ms.fit_transform(x)
```

Train-Test-Split [75-25]

```
X_train, X_test, y_train, y_test = train_test_split(x, y,test_size=0.3)
print(X_train.shape, X_test.shape)
```

```
(72687, 56) (31152, 56)
```

Deep Neural Network-LSTM

```
Classifier_accuracy = []
```

Defining the Deep Neural Network-long short term memory

```
# Define and compile model
model = keras.Sequential()
model.add(Dense(28 , input_shape=(56,) , activation="relu" , name="Hidden_Layer_1"))
model.add(Dense(10 , activation="relu" , name="Hidden_Layer_2"))
model.add(Dense(1 , activation="sigmoid" , name="Output_Layer"))
opt = keras.optimizers.Adam(learning_rate=0.01)
model.compile( optimizer=opt, loss="binary_crossentropy", metrics=['accuracy'])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 Hidden_Layer_1 (Dense)      (None, 28)                1596

 Hidden_Layer_2 (Dense)      (None, 10)                290

 Output_Layer (Dense)        (None, 1)                 11

=================================================================
Total params: 1897 (7.41 KB)
Trainable params: 1897 (7.41 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Model fitting

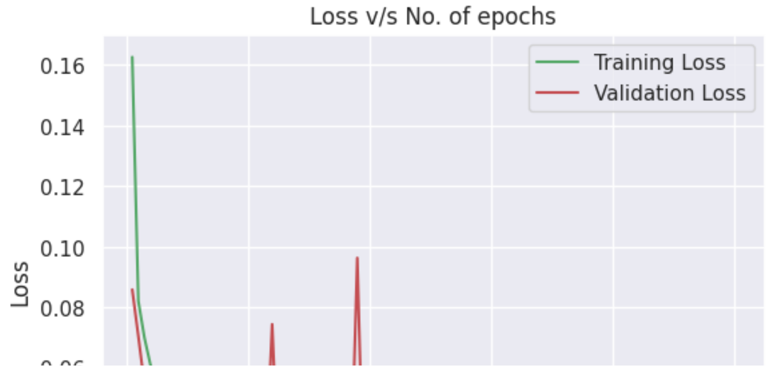
```
# fit model
history_org = model.fit(
    X_train,
    y_train,
    batch_size=32,
    epochs=100, verbose=2,
    callbacks=None,
    validation_data=(X_test,y_test),
    shuffle=True,
    class_weight=None,
```

```
    sample_weight=None,
    initial_epoch=0)
```

```
Epoch 72/100
2272/2272 - 6s - loss: 0.0214 - accuracy: 0.9909 - val_loss: 0.0216 - val_accuracy: 0.9908 - 6s/epoch - 3ms/step
Epoch 73/100
2272/2272 - 5s - loss: 0.0195 - accuracy: 0.9910 - val_loss: 0.0291 - val_accuracy: 0.9888 - 5s/epoch - 2ms/step
Epoch 74/100
2272/2272 - 4s - loss: 0.0214 - accuracy: 0.9906 - val_loss: 0.0190 - val_accuracy: 0.9915 - 4s/epoch - 2ms/step
Epoch 75/100
2272/2272 - 6s - loss: 0.0208 - accuracy: 0.9909 - val_loss: 0.0196 - val_accuracy: 0.9916 - 6s/epoch - 3ms/step
Epoch 76/100
2272/2272 - 5s - loss: 0.0204 - accuracy: 0.9908 - val_loss: 0.0201 - val_accuracy: 0.9911 - 5s/epoch - 2ms/step
Epoch 77/100
2272/2272 - 4s - loss: 0.0211 - accuracy: 0.9907 - val_loss: 0.0183 - val_accuracy: 0.9925 - 4s/epoch - 2ms/step
Epoch 78/100
2272/2272 - 7s - loss: 0.0200 - accuracy: 0.9910 - val_loss: 0.0196 - val_accuracy: 0.9922 - 7s/epoch - 3ms/step
Epoch 79/100
2272/2272 - 4s - loss: 0.0196 - accuracy: 0.9913 - val_loss: 0.0183 - val_accuracy: 0.9917 - 4s/epoch - 2ms/step
Epoch 80/100
2272/2272 - 4s - loss: 0.0218 - accuracy: 0.9906 - val_loss: 0.0165 - val_accuracy: 0.9923 - 4s/epoch - 2ms/step
Epoch 81/100
2272/2272 - 6s - loss: 0.0205 - accuracy: 0.9911 - val_loss: 0.0231 - val_accuracy: 0.9899 - 6s/epoch - 3ms/step
Epoch 82/100
2272/2272 - 4s - loss: 0.0197 - accuracy: 0.9910 - val_loss: 0.0167 - val_accuracy: 0.9928 - 4s/epoch - 2ms/step
Epoch 83/100
2272/2272 - 5s - loss: 0.0212 - accuracy: 0.9910 - val_loss: 0.0174 - val_accuracy: 0.9925 - 5s/epoch - 2ms/step
Epoch 84/100
2272/2272 - 6s - loss: 0.0194 - accuracy: 0.9910 - val_loss: 0.0208 - val_accuracy: 0.9907 - 6s/epoch - 3ms/step
Epoch 85/100
2272/2272 - 5s - loss: 0.0202 - accuracy: 0.9911 - val_loss: 0.0167 - val_accuracy: 0.9920 - 5s/epoch - 2ms/step
Epoch 86/100
2272/2272 - 6s - loss: 0.0196 - accuracy: 0.9912 - val_loss: 0.0218 - val_accuracy: 0.9912 - 6s/epoch - 3ms/step
Epoch 87/100
2272/2272 - 5s - loss: 0.0191 - accuracy: 0.9916 - val_loss: 0.0419 - val_accuracy: 0.9897 - 5s/epoch - 2ms/step
Epoch 88/100
2272/2272 - 4s - loss: 0.0204 - accuracy: 0.9912 - val_loss: 0.0218 - val_accuracy: 0.9906 - 4s/epoch - 2ms/step
Epoch 89/100
2272/2272 - 6s - loss: 0.0192 - accuracy: 0.9913 - val_loss: 0.0195 - val_accuracy: 0.9917 - 6s/epoch - 3ms/step
Epoch 90/100
2272/2272 - 4s - loss: 0.0211 - accuracy: 0.9909 - val_loss: 0.0204 - val_accuracy: 0.9913 - 4s/epoch - 2ms/step
Epoch 91/100
2272/2272 - 5s - loss: 0.0185 - accuracy: 0.9916 - val_loss: 0.0272 - val_accuracy: 0.9889 - 5s/epoch - 2ms/step
Epoch 92/100
2272/2272 - 6s - loss: 0.0185 - accuracy: 0.9916 - val_loss: 0.0193 - val_accuracy: 0.9916 - 6s/epoch - 3ms/step
Epoch 93/100
2272/2272 - 4s - loss: 0.0213 - accuracy: 0.9908 - val_loss: 0.0171 - val_accuracy: 0.9923 - 4s/epoch - 2ms/step
Epoch 94/100
2272/2272 - 5s - loss: 0.0173 - accuracy: 0.9920 - val_loss: 0.0231 - val_accuracy: 0.9903 - 5s/epoch - 2ms/step
Epoch 95/100
2272/2272 - 6s - loss: 0.0198 - accuracy: 0.9914 - val_loss: 0.0148 - val_accuracy: 0.9930 - 6s/epoch - 3ms/step
Epoch 96/100
2272/2272 - 5s - loss: 0.0193 - accuracy: 0.9917 - val_loss: 0.0282 - val_accuracy: 0.9897 - 5s/epoch - 2ms/step
Epoch 97/100
2272/2272 - 5s - loss: 0.0192 - accuracy: 0.9912 - val_loss: 0.0165 - val_accuracy: 0.9927 - 5s/epoch - 2ms/step
Epoch 98/100
2272/2272 - 6s - loss: 0.0176 - accuracy: 0.9919 - val_loss: 0.0247 - val_accuracy: 0.9896 - 6s/epoch - 3ms/step
Epoch 99/100
2272/2272 - 4s - loss: 0.0195 - accuracy: 0.9914 - val_loss: 0.0162 - val_accuracy: 0.9926 - 4s/epoch - 2ms/step
Epoch 100/100
2272/2272 - 6s - loss: 0.0184 - accuracy: 0.9916 - val_loss: 0.0316 - val_accuracy: 0.9905 - 6s/epoch - 3ms/step
```

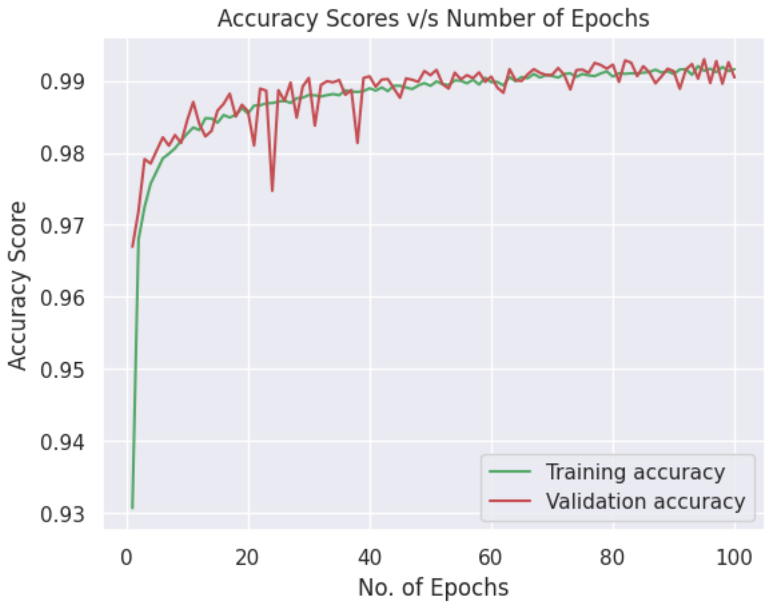
Plotting Loss v/s Epochs

```
loss = history_org.history['loss']
val_loss = history_org.history['val_loss']
epochs = range(1, len(loss) + 1)
plt.plot(epochs, loss, 'g', label = 'Training Loss')
plt.plot(epochs, val_loss, 'r', label = 'Validation Loss')
plt.title('Loss v/s No. of epochs')
plt.xlabel('Number of Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



Plotting Accuracy v/s Epochs

```
loss = history_org.history['accuracy']
val_loss = history_org.history['val_accuracy']
plt.plot(epochs, loss, 'g', label = 'Training accuracy')
plt.plot(epochs, val_loss, 'r', label = 'Validation accuracy')
plt.title('Accuracy Scores v/s Number of Epochs')
plt.xlabel('No. of Epochs')
plt.ylabel('Accuracy Score')
plt.legend()
plt.show()
```



Model Evaluation

```
loss, accuracy = model.evaluate(X_test, y_test)
print('Accuracy of Deep neural Network : %.2f' % (accuracy*100))
Classifier_accuracy.append(accuracy*100)
```

```
974/974 [==============================] - 1s 1ms/step - loss: 0.0316 - accuracy: 0.9905
Accuracy of Deep neural Network : 99.05
```

K-Nearest Neighbor Classifier

```
knn_clf = KNeighborsClassifier()
knn_clf.fit(X_train, y_train)
y_pred = knn_clf.predict(X_test)
accuracy = metrics.accuracy_score(y_test, y_pred)
Classifier_accuracy.append(accuracy*100)
print("Accuracy of KNN Classifier : %.2f" % (accuracy*100))
```

```
Accuracy of KNN Classifier : 96.62
```

SVM Classifier

```
svc_clf = SVC()
svc_clf.fit(X_train,y_train)
y_pred = svc_clf.predict(X_test)
accuracy = metrics.accuracy_score(y_test, y_pred)
Classifier_accuracy.append(accuracy*100)
print("Accuracy of SVM Classifier : %.2f" % (accuracy*100) )
```

```
Accuracy of SVM Classifier : 97.45
```

Decision Tree Classifier

```
dt_clf = DecisionTreeClassifier(max_depth=5)
dt_clf.fit(X_train,y_train)
y_pred = dt_clf.predict(X_test)
accuracy = metrics.accuracy_score(y_test, y_pred)
Classifier_accuracy.append(accuracy*100)
print("Accuracy of Decision Tree Classifier : %.2f" % (accuracy*100) )
```

```
Accuracy of Decision Tree Classifier : 96.50
```

Naive Bayes Classifier

```
nb_clf = CategoricalNB()
nb_clf.fit(X_train,y_train)
y_pred = nb_clf.predict(X_test)
accuracy = metrics.accuracy_score(y_test, y_pred)
Classifier_accuracy.append(accuracy*100)
print("Accuracy of Naive Bayes Classifier : %.2f" % (accuracy*100) )
```

```
Accuracy of Naive Bayes Classifier : 71.65
```

Quadratic Discriminant Analysis Classifier

```
qda_clf=QuadraticDiscriminantAnalysis()
qda_clf.fit(X_train,y_train)
y_pred=qda_clf.predict(X_test)
accuracy = metrics.accuracy_score(y_test, y_pred)
Classifier_accuracy.append(accuracy*100)
print("Accuracy of QDA Classifier : %.2f" % (accuracy*100))
```

```
Accuracy of QDA Classifier : 50.58
```

Stochastic Gradient Classifier

```
sgd_clf=SGDClassifier(loss="hinge", penalty="l2")
sgd_clf.fit(X_train,y_train)
y_pred=sgd_clf.predict(X_test)
accuracy = metrics.accuracy_score(y_test, y_pred)
Classifier_accuracy.append(accuracy*100)
print("Accuracy of SGD Classifier : %.2f" % (accuracy*100))
```

```
Accuracy of SGD Classifier : 84.26
```

Logistic Regression

```
lr_clf = LogisticRegression()
lr_clf.fit(X_train,y_train)
y_pred=lr_clf.predict(X_test)
accuracy = metrics.accuracy_score(y_test, y_pred)
Classifier_accuracy.append(accuracy*100)
print("Accuracy of Logistic Regression Classifier : %.2f" % (accuracy*100))
```

```
Accuracy of Logistic Regression Classifier : 83.93
```

XGBoost Classifier

```
xgb_clf=xgb.XGBClassifier(eval_metric = 'error',objective='binary:logistic',max_depth=2, learning_rate=0.1)
xgb_clf.fit(X_train,y_train)
y_pred=xgb_clf.predict(X_test)
accuracy = metrics.accuracy_score(y_test, y_pred)
```

```
Classifier_accuracy.append(accuracy*100)
print("Accuracy of XGBoost Classifier : %.2f" % (accuracy*100))
```

```
Accuracy of XGBoost Classifier : 98.29
```

Comparitive analysis of models

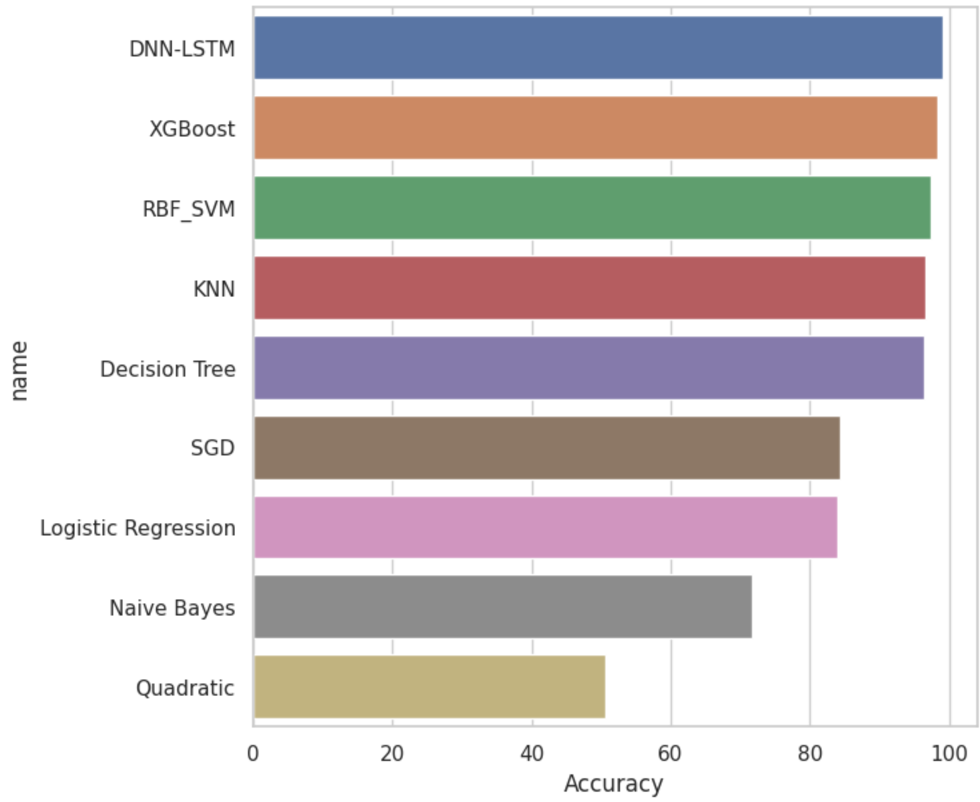
```
Classifier_names = ["DNN-LSTM", "KNN", "RBF_SVM", "Decision Tree","Naive Bayes","Quadratic","SGD","Logistic Regression","XGBoost"]
```

```
df_clf = pd.DataFrame()
df_clf['name'] = Classifier_names
df_clf['Accuracy'] = Classifier_accuracy
df_clf = df_clf.sort_values(by=['Accuracy'], ascending=False)
df_clf.head(10)
```

	name	Accuracy
0	DNN-LSTM	99.046612
8	XGBoost	98.289034
2	RBF_SVM	97.447997
1	KNN	96.616590
3	Decision Tree	96.497817
6	SGD	84.261043
7	Logistic Regression	83.933616
4	Naive Bayes	71.645480
5	Quadratic	50.584232

Visualize accuracies of the models

```
sns.set(style="whitegrid",rc={'figure.figsize':(7,7)})
ax = sns.barplot(y="name", x="Accuracy", data=df_clf)
```




```
print(f"The best baseline Classifier is {df_clf.name[0]} with an accuracy of {df_clf.Accuracy[0]}.")
```

```
The best baseline Classifier is DNN-LSTM with an accuracy of 99.04661178588867.
```

Hyperparameter tuning

```
def model_builder(hp):
    model = keras.Sequential()

    model.add(Dense(28 , input_shape=(56,) , activation="relu" , name="Hidden_Layer_1"))
    model.add(Dense(10 , activation="relu" , name="Hidden_Layer_2"))
    model.add(Dense(1 , activation="sigmoid" , name="Output_Layer"))
    opt = keras.optimizers.Adam(learning_rate=0.01)

    model.compile(optimizer=keras.optimizers.Adam(hp.Choice('learning_rate',[1e-2, 1e-3, 1e-4])), loss='binary_crossentropy', metrics=['a

    return history, model.layers, model
```

```
pip install keras-tuner --upgrade
```

```
Collecting keras-tuner
  Downloading keras_tuner-1.4.4-py3-none-any.whl (127 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 128.0/128.0 kB 2.2 MB/s eta 0:00:00
Collecting keras-core (from keras-tuner)
  Downloading keras_core-0.1.7-py3-none-any.whl (950 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 950.8/950.8 kB 18.6 MB/s eta 0:00:00
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from keras-tuner) (23.1)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from keras-tuner) (2.31.0)
Collecting kt-legacy (from keras-tuner)
  Downloading kt_legacy-1.0.5-py3-none-any.whl (9.6 kB)
Requirement already satisfied: absl-py in /usr/local/lib/python3.10/dist-packages (from keras-core->keras-tuner) (1.4.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from keras-core->keras-tuner) (1.23.5)
Requirement already satisfied: rich in /usr/local/lib/python3.10/dist-packages (from keras-core->keras-tuner) (13.5.3)
Collecting namex (from keras-core->keras-tuner)
  Downloading namex-0.0.7-py3-none-any.whl (5.8 kB)
Requirement already satisfied: h5py in /usr/local/lib/python3.10/dist-packages (from keras-core->keras-tuner) (3.9.0)
Requirement already satisfied: dm-tree in /usr/local/lib/python3.10/dist-packages (from keras-core->keras-tuner) (0.1.8)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->keras-tuner) (3.
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->keras-tuner) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->keras-tuner) (2.0.5)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->keras-tuner) (2023.7.2
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.10/dist-packages (from rich->keras-core->keras-tuner
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.10/dist-packages (from rich->keras-core->keras-tun
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.10/dist-packages (from markdown-it-py>=2.2.0->rich->keras-core-
Installing collected packages: namex, kt-legacy, keras-core, keras-tuner
Successfully installed keras-core-0.1.7 keras-tuner-1.4.4 kt-legacy-1.0.5 namex-0.0.7
```

```
classes = model.predict(X_test)
print(classes)
```

```
974/974 [==============================] - 1s 1ms/step
[[3.3104930e-06]
 [1.0682698e-22]
 [0.0000000e+00]
 ...
 [1.0376321e-08]
 [0.0000000e+00]
 [3.3709734e-24]]
```

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

```
y_pred[:20]
```

```
[0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0]
```

```
y_test[:20]
```

```
19725     0
61382     0
103595    0
60224     0
85426     0
57143     0
73674     0
56837     0
96473     1
81426     0
91794     0
7238      0
67253     0
83397     0
```