```
''' Computer Networks Project

    Group Members: Oindri Kar
                   Niki Esmaeili

    Google collab link of the code : https://colab.research.google.com/drive/1LIZz
    '''
```

```
' Computer Networks Project \n\n    Group Members: Oindri Kar \n
Niki Esmaeili\n\n    Google collab link of the code : https://colab.research.g
oogle.com/drive/1LIZzDBD-dz4Dn8_zg1jRQEFoCKRJOVxq?usp=sharing\n    '
```

```
# importing libraries
import time
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, OneHotEncoder
from sklearn.metrics import confusion_matrix, classification_report
import sklearn
from sklearn import metrics
import tensorflow as tf
import os

from sklearn.cluster import KMeans
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from matplotlib import pyplot as plt
%matplotlib inline
```

```
# This package facilitates access to Google Drive through Python.
!pip install -U -q PyDrive
```

```
# Authentication Process to access the drive
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials
```

```
WARNING:root:pydrive is deprecated and no longer maintained. We recommend that
```

```
# Authenticate and create the PyDrive client.
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)
```

The dataset was taken from Kaggel (https://www.kaggle.com/datasets/crawford/computer-network-traffic/data) and save on google drive

```
link = 'https://drive.google.com/file/d/1TG-cTzRJYxszH0lPtT_XqiMrroPjnR0n/view?usp

import pandas as pd

# to get the id part of the file
id = link.split("/")[-2]

downloaded = drive.CreateFile({'id':id})
downloaded.GetContentFile('network_data.csv')

df = pd.read_csv('network_data.csv')
df.head()
```

	Flow.ID	Source.IP	Source.Port	Destination.IP	Destination.Port	Prot
0	172.19.1.46-10.200.7.7-52422-3128-6	172.19.1.46	52422	10.200.7.7	3128	
1	172.19.1.46-10.200.7.7-52422-3128-6	10.200.7.7	3128	172.19.1.46	52422	
2	10.200.7.217-50.31.185.39-38848-80-6	50.31.185.39	80	10.200.7.217	38848	
3	10.200.7.217-50.31.185.39-38848-80-6	50.31.185.39	80	10.200.7.217	38848	
4	192.168.72.43-10.200.7.7-55961-3128-6	192.168.72.43	55961	10.200.7.7	3128	

5 rows × 87 columns

```
# Checking types of values
print(df.dtypes)
```

```
Flow.ID               object
Source.IP             object
Source.Port            int64
Destination.IP        object
Destination.Port       int64
                      ...
Idle.Max             float64
Idle.Min             float64
Label                 object
L7Protocol             int64
ProtocolName          object
Length: 87, dtype: object
```

```
# Checking if any value in the dataframe is null
df.isnull().values.any()
```

```
False
```

```
# Checking columns that have only one unique value
df.columns[df.nunique() <= 1]
```

```
Index(['Bwd.PSH.Flags', 'Fwd.URG.Flags', 'Bwd.URG.Flags', 'CWE.Flag.Count',
       'Fwd.Avg.Bytes.Bulk', 'Fwd.Avg.Packets.Bulk', 'Fwd.Avg.Bulk.Rate',
       'Bwd.Avg.Bytes.Bulk', 'Bwd.Avg.Packets.Bulk', 'Bwd.Avg.Bulk.Rate',
       'Label'],
      dtype='object')
```

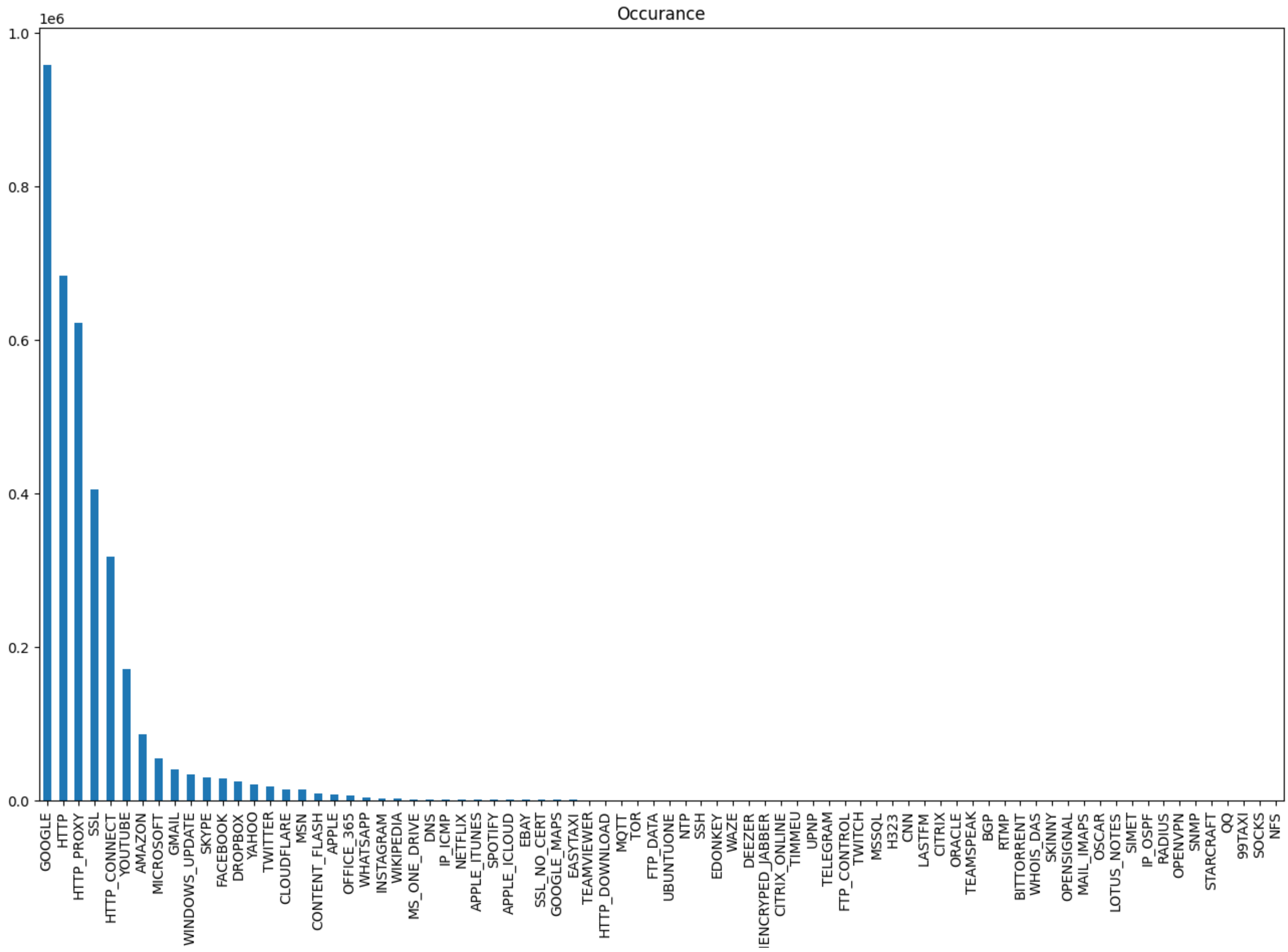
```
# Checking occurance of each application
df['ProtocolName'].value_counts()
```

```
GOOGLE          959110
HTTP            683734
HTTP_PROXY      623210
SSL             404883
HTTP_CONNECT    317526
                 ...
STARCRAFT            3
QQ                   2
99TAXI               1
SOCKS                1
NFS                  1
Name: ProtocolName, Length: 78, dtype: int64
```

```
# Features that will be removed from dataset because they have low occurances of r
feats_toDelete = df['ProtocolName'].value_counts()[-25:].index
feats_toDelete
```

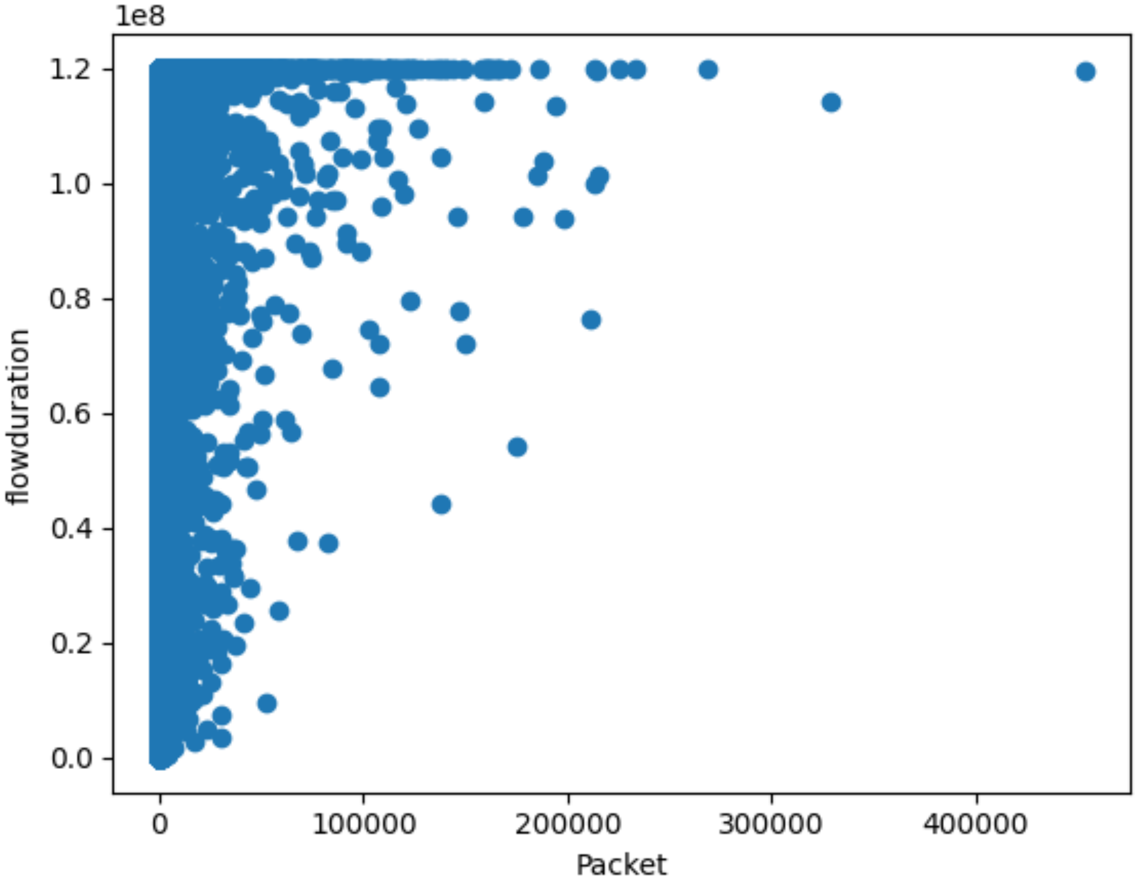
```
Index(['H323', 'CNN', 'LASTFM', 'CITRIX', 'ORACLE', 'TEAMSPEAK', 'BGP',
'RTMP',
       'BITTORRENT', 'WHOIS_DAS', 'SKINNY', 'OPENSIGNAL', 'MAIL_IMAPS',
       'OSCAR', 'LOTUS_NOTES', 'SIMET', 'IP_OSPF', 'RADIUS', 'OPENVPN',
'SNMP',
       'STARCRAFT', 'QQ', '99TAXI', 'SOCKS', 'NFS'],
      dtype='object')
```

```
# Plot the number of records for individual applications
target_count = df['ProtocolName'].value_counts()
plt.figure(figsize=(16,10))
target_count.plot(kind='bar', title='Occurance');
```



```
# Scatter plot of packets and their flow duration
plt.scatter(df['Total.Fwd.Packets'],df['Flow.Duration'])
plt.xlabel('Packet')
plt.ylabel('flowduration')
```

```
Text(0, 0.5, 'flowduration')
```




```
# clustering on subset of columns
km = KMeans(n_clusters=2)
y_predicted = km.fit_predict(df[['Total.Fwd.Packets','Flow.Duration','Total.Backwa
y_predicted
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Future
  warnings.warn(
array([0, 0, 0, ..., 0, 0, 0], dtype=int32)
```

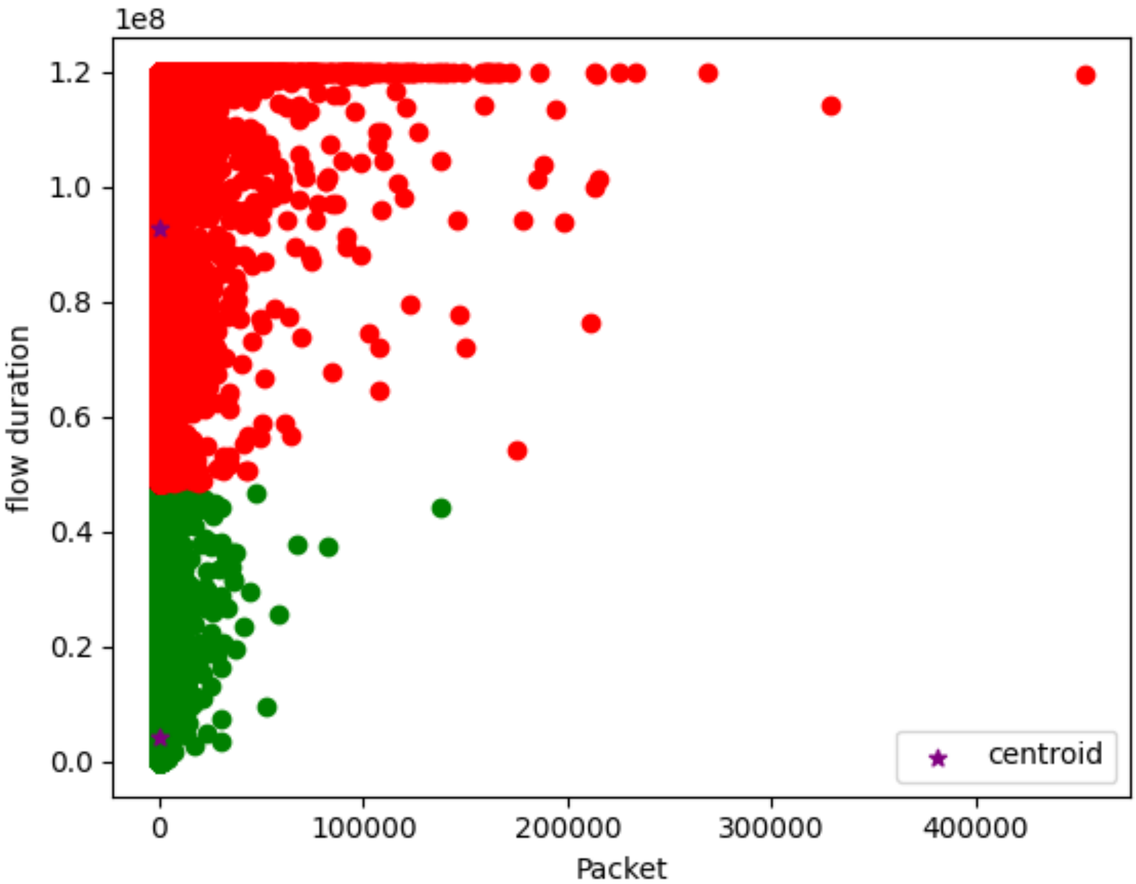
```
df['Label']=y_predicted
```

```
ct=0
for i in df['Label']:
    if i==0:
        ct=ct+1
ct
```

```
2717822
```

```
# Scatter plot after clustering from KMeans
df1 = df[df['Label']==0]
df2 = df[df['Label']==1]

plt.scatter(df1['Total.Fwd.Packets'],df1['Flow.Duration'],color='green')
plt.scatter(df2['Total.Fwd.Packets'],df2['Flow.Duration'],color='red')

plt.scatter(km.cluster_centers_[:,0],km.cluster_centers_[:,1],color='purpl
plt.xlabel('Packet')
plt.ylabel('flow duration')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7bce026dd690>
```



```
# dataframe representation
df
```

	Flow.ID	Source.IP	Source.Port	Destination.IP	Destination.Port
0	172.19.1.46-10.200.7.7-52422-3128-6	172.19.1.46	52422	10.200.7.7	3128
1	172.19.1.46-10.200.7.7-52422-3128-6	10.200.7.7	3128	172.19.1.46	52422
2	10.200.7.217-50.31.185.39-38848-80-6	50.31.185.39	80	10.200.7.217	38848
3	10.200.7.217-50.31.185.39-38848-80-6	50.31.185.39	80	10.200.7.217	38848
4	192.168.72.43-10.200.7.7-55961-3128-6	192.168.72.43	55961	10.200.7.7	3128
...	...	...	...	...	...
3577291	10.200.7.199-98.138.79.73-42135-443-6	98.138.79.73	443	10.200.7.199	42135
3577292	10.200.7.217-98.138.79.73-51546-443-6	98.138.79.73	443	10.200.7.217	51546
3577293	10.200.7.218-98.138.79.73-44366-443-6	98.138.79.73	443	10.200.7.218	44366
3577294	10.200.7.195-98.138.79.73-52341-443-6	98.138.79.73	443	10.200.7.195	52341
3577295	10.200.7.196-98.138.79.73-34188-443-6	98.138.79.73	443	10.200.7.196	34188

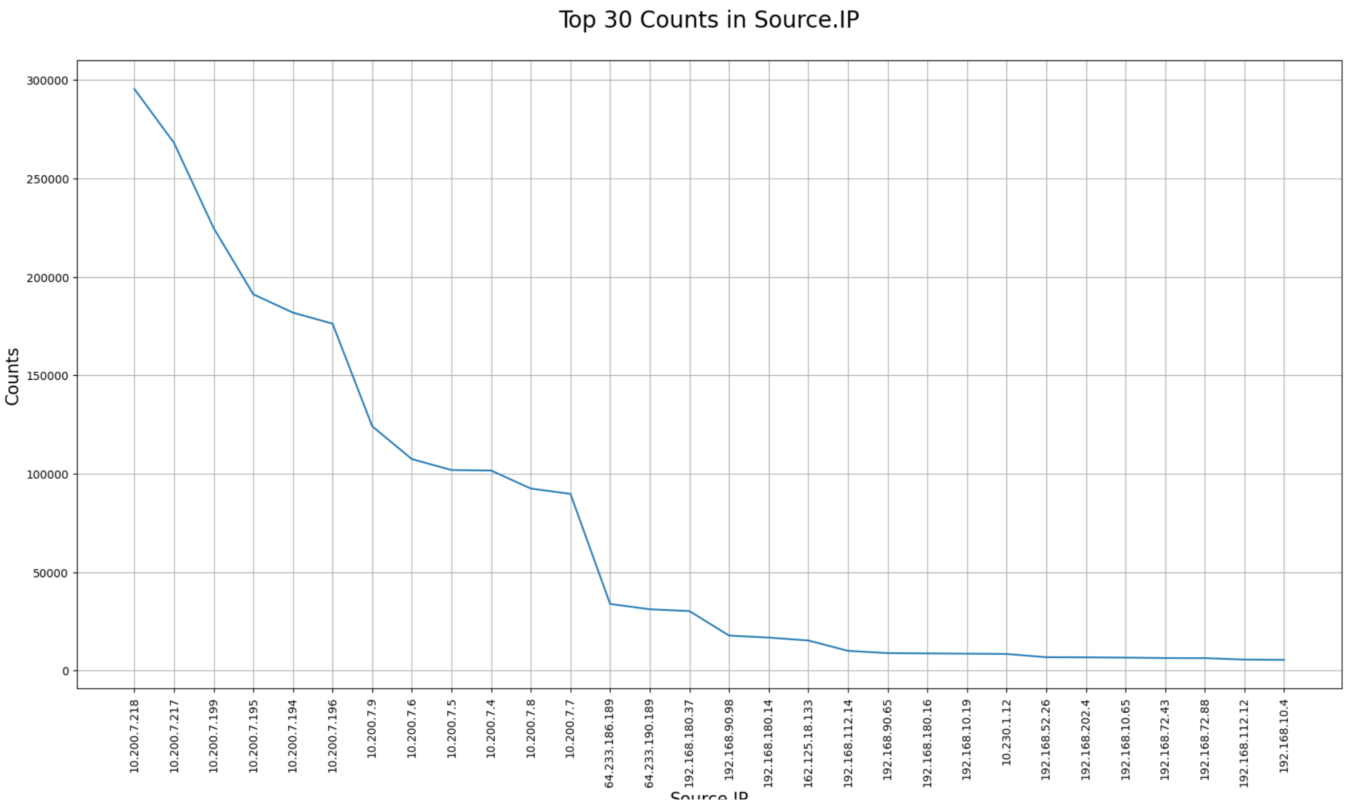
3577296 rows × 87 columns

```
# Coordinated of the centroids for each cluster
km.cluster_centers_
```

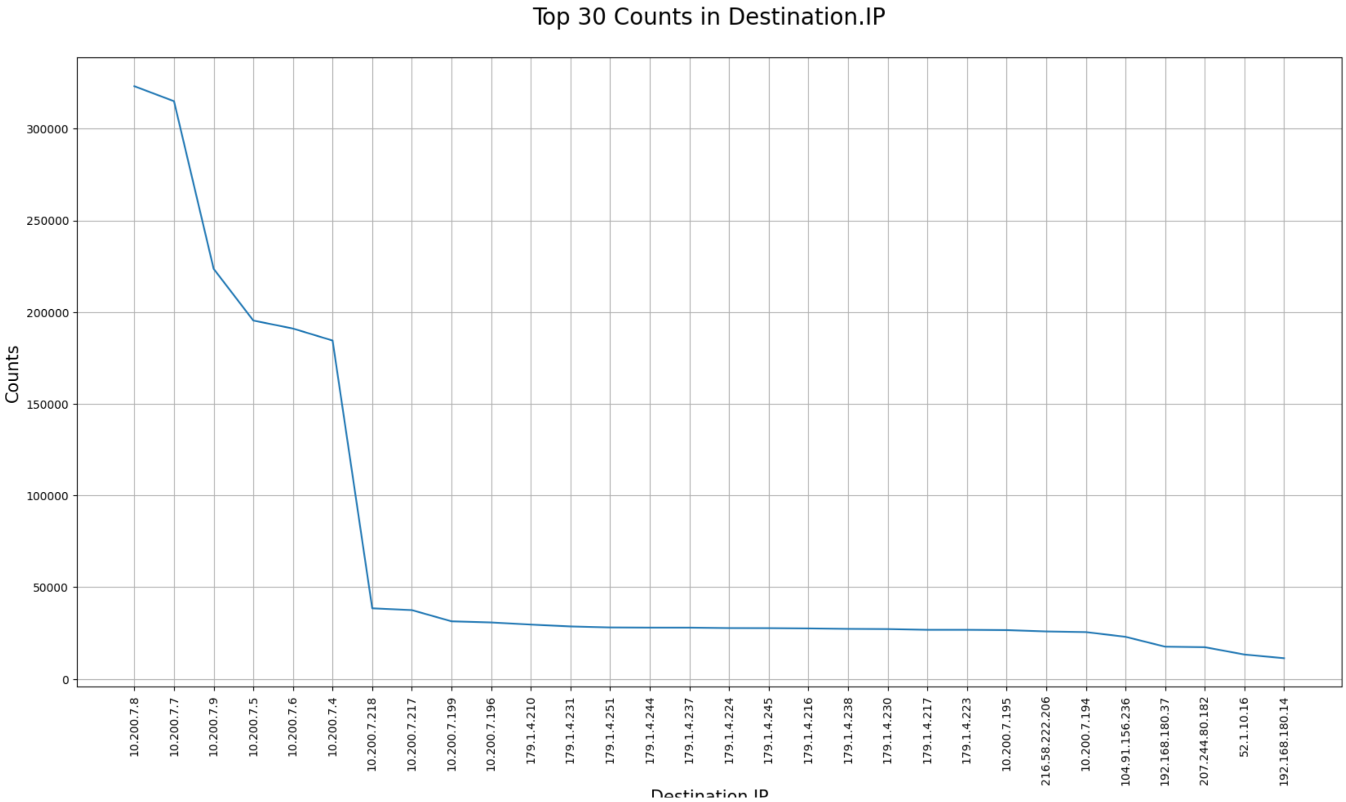
```
array([[1.95309541e+01, 4.08978582e+06, 2.36739997e+01, 3.41130526e+04,
        7.63873198e+03],
       [1.97880805e+02, 9.29698458e+07, 1.97111265e+02, 2.43670391e+05,
        1.70784950e+05]])
```

```
dataset=df
```

```
# Histogram on Source.IP
Sour_feat = pd.DataFrame(dataset['Source.IP'].value_counts()[:30])
plt.figure(figsize=(20,10))
plt.plot(Sour_feat)
plt.xticks(rotation=90)
plt.xlabel('Source.IP', {'fontsize':15})
plt.ylabel('Counts', {'fontsize':15})
plt.title('Top 30 Counts in Source.IP\n', {'fontsize':20})
plt.grid()
plt.savefig('hist Source.IP.png')
Sour_feat = Sour_feat.reset_index()['index'].values
```



```
# Histogram on Destination.IP
Dest_feat = pd.DataFrame(dataset['Destination.IP'].value_counts()[:30])
plt.figure(figsize=(20,10))
plt.plot(Dest_feat)
plt.xticks(rotation=90)
plt.xlabel('Destination.IP', {'fontsize':15})
plt.ylabel('Counts', {'fontsize':15})
plt.title('Top 30 Counts in Destination.IP\n', {'fontsize':20})
plt.grid()
plt.savefig('hist Destination.IP.png')
Dest_feat = Dest_feat.reset_index()['index'].values
```




```
# Filtering the dataset to contain only 30 frequently reported IP address in Sourc
f_dataset = dataset[dataset['Destination.IP'].isin(Dest_feat) & dataset['Source.IP
f_dataset = f_dataset.drop('index', axis=1)
```

```
# removing columns
f_dataset = f_dataset.drop(f_dataset.select_dtypes(include = ['object']).c
f_dataset = f_dataset.drop(['Source.Port','Destination.Port','L7Protocol',
f_dataset.columns
```

```
Index(['Flow.Duration', 'Total.Fwd.Packets', 'Total.Backward.Packets',
       'Total.Length.of.Fwd.Packets', 'Total.Length.of.Bwd.Packets',
       'Fwd.Packet.Length.Max', 'Fwd.Packet.Length.Min',
       'Fwd.Packet.Length.Mean', 'Fwd.Packet.Length.Std',
       'Bwd.Packet.Length.Max', 'Bwd.Packet.Length.Min',
       'Bwd.Packet.Length.Mean', 'Bwd.Packet.Length.Std', 'Flow.Bytes.s',
       'Flow.Packets.s', 'Flow.IAT.Mean', 'Flow.IAT.Std', 'Flow.IAT.Max',
       'Flow.IAT.Min', 'Fwd.IAT.Total', 'Fwd.IAT.Mean', 'Fwd.IAT.Std',
       'Fwd.IAT.Max', 'Fwd.IAT.Min', 'Bwd.IAT.Total', 'Bwd.IAT.Mean',
       'Bwd.IAT.Std', 'Bwd.IAT.Max', 'Bwd.IAT.Min', 'Fwd.PSH.Flags',
       'Bwd.PSH.Flags', 'Fwd.URG.Flags', 'Bwd.URG.Flags', 'Fwd.Header.Length',
       'Bwd.Header.Length', 'Fwd.Packets.s', 'Bwd.Packets.s',
       'Min.Packet.Length', 'Max.Packet.Length', 'Packet.Length.Mean',
       'Packet.Length.Std', 'Packet.Length.Variance', 'FIN.Flag.Count',
       'SYN.Flag.Count', 'RST.Flag.Count', 'PSH.Flag.Count', 'ACK.Flag.Count',
       'URG.Flag.Count', 'CWE.Flag.Count', 'ECE.Flag.Count', 'Down.Up.Ratio',
       'Average.Packet.Size', 'Avg.Fwd.Segment.Size', 'Avg.Bwd.Segment.Size',
       'Fwd.Header.Length.1', 'Fwd.Avg.Bytes.Bulk', 'Fwd.Avg.Packets.Bulk',
       'Fwd.Avg.Bulk.Rate', 'Bwd.Avg.Bytes.Bulk', 'Bwd.Avg.Packets.Bulk',
       'Bwd.Avg.Bulk.Rate', 'Subflow.Fwd.Packets', 'Subflow.Fwd.Bytes',
       'Subflow.Bwd.Packets', 'Subflow.Bwd.Bytes', 'Init_Win_bytes_forward',
       'Init_Win_bytes_backward', 'act_data_pkt_fwd', 'min_seg_size_forward',
       'Active.Mean', 'Active.Std', 'Active.Max', 'Active.Min', 'Idle.Mean',
       'Idle.Std', 'Idle.Max', 'Idle.Min', 'Label'],
      dtype='object')
```

```
label = pd.get_dummies(f_dataset['Label'])
```

```
f_dataset=f_dataset.drop(["Label"],axis=1)
```

Epoch

```
nr=25
```

Using RNN Method

```
# installing TensorFlow to set up neural network
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, LSTM

from tensorflow.keras.layers import Embedding
```

```
X=f_dataset
X.shape
```

```
(674133, 77)
```

```
y=label
y.shape
```

```
(674133, 2)
```

```
y=pd.DataFrame([x for x in np.where(y ==1, y.columns,'').flatten().tolist() if len
```

```
y=y.to_numpy()
```

```
# preprocessing steps before feeding data into KNN
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X)
X = scaler.transform(X)
from tensorflow.keras.utils import to_categorical
y = to_categorical(y)
```

```
print(X.shape)
print(y.shape)
```

```
(674133, 77)
(674133, 2)
```

```
# splitting the dataset into training and testing
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_
```

```
print(X_test.shape)
print(y_test.shape)
print(X_train.shape)
print(y_train.shape)
```

```
(134827, 77)
(134827, 2)
(539306, 77)
(539306, 2)
```

```
X_train = np.reshape(X_train, (X_train.shape[0],1,X.shape[1]))
X_test = np.reshape(X_test, (X_test.shape[0],1,X.shape[1]))
```

```
# clearing last session
tf.keras.backend.clear_session()

model = Sequential()

# adding LSTM layers
model.add(LSTM(128, input_shape=(1,77),activation="relu",return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(128,activation="relu"))
model.add(Dropout(0.2))

# adding output dense layer
model.add(Dense(y.shape[1], activation='sigmoid'))

# compiling the model
from tensorflow.keras.optimizers import SGD
model.compile(loss = 'binary_crossentropy', optimizer = "adam", metrics = ['accura
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 1, 128)            105472

 dropout (Dropout)           (None, 1, 128)            0

 lstm_1 (LSTM)               (None, 128)               131584

 dropout_1 (Dropout)         (None, 128)               0

 dense (Dense)               (None, 2)                 258

=================================================================
Total params: 237314 (927.01 KB)
Trainable params: 237314 (927.01 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
# Implementation of early stopping
from keras import callbacks
earlystopping = callbacks.EarlyStopping(monitor ="val_loss",
                                        mode ="min", patience = 2,
                                        restore_best_weights = True)
```

```
# Train neural network using training data
history = model.fit(X_train, y_train, epochs = nr, validation_data= (X_tes
history
```

```
Epoch 1/25
16854/16854 [==============================] - 199s 12ms/step - loss: 0.0109 -
Epoch 2/25
16854/16854 [==============================] - 189s 11ms/step - loss: 0.0059 -
Epoch 3/25
16854/16854 [==============================] - 191s 11ms/step - loss: 0.0058 -
<keras.src.callbacks.History at 0x7bcd28316ef0>
```

```
acc2 = model.evaluate(X_test, y_test)
```

```
4214/4214 [==============================] - 16s 4ms/step - loss: 0.0044 - acc
```

```
ans=model.predict(X_test)
```

```
4214/4214 [==============================] - 16s 4ms/step
```

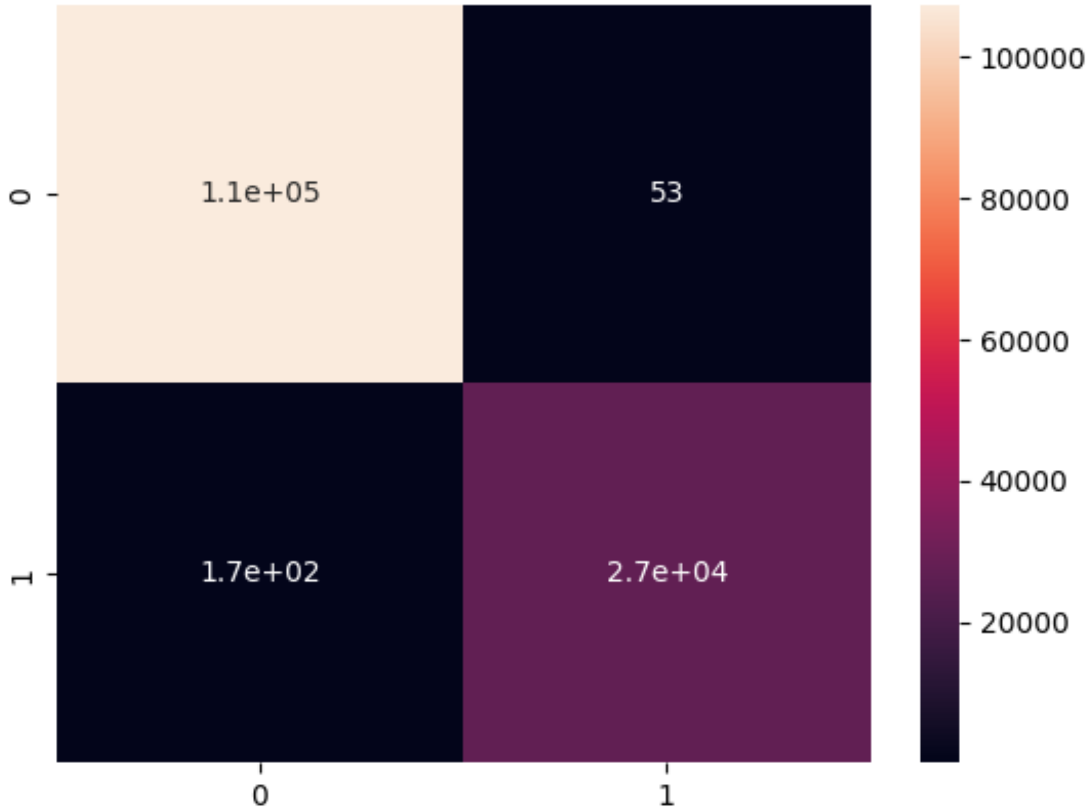
```
y_pred=np.argmax(ans, axis=1)
y_test=np.argmax(y_test, axis=1)
cm = confusion_matrix(y_test, y_pred)
```

```
cm
```

```
array([[107335,     53],
       [   167,  27272]])
```

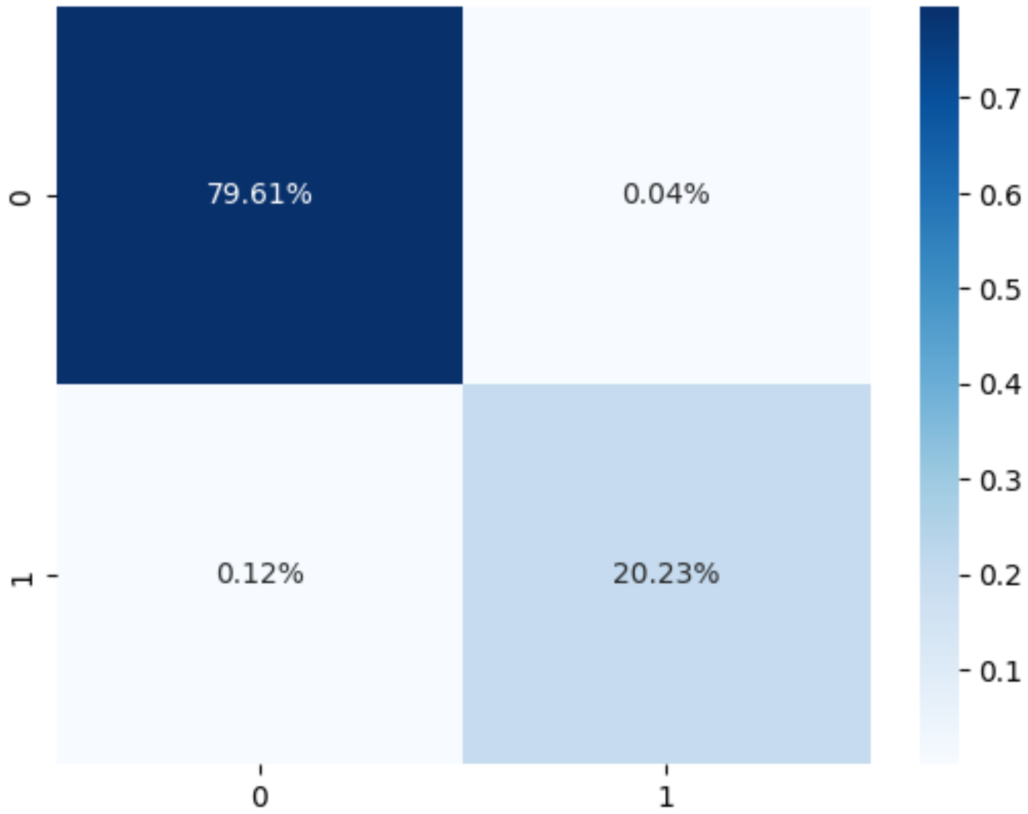
```
# heatmap visualization of confusion matrix
import matplotlib.pyplot as plt
import seaborn as sns
sns.heatmap(cm, annot=True)
```

<Axes: >



```
# heatmal visualization of normalized confusion matrix
sns.heatmap(cm/np.sum(cm), annot=True,
            fmt='.2%', cmap='Blues')
```

<Axes: >




```
# represents the components of confusion matrix
TP=cm[0][0]    # True Positive
FN=cm[0][1]    # False Negative
FP=cm[1][0]    # False Positive
TN=cm[1][1]    # True Negative
```

```
from sklearn.metrics import precision_recall_fscore_support
```

```
# Accuracy Calculation
acc_best=(TP+TN)/(TP+FN+FP+TN)
print("acc_best",acc_best)
sum=0
for i in range(1,nr+1,1):
    sum=sum+pow(acc_best,i)
print("sum",sum)

# Average Accuracy Calculation
av_acc=sum/nr
print("Average Recall av_acc",av_acc)
```

```
acc_best 0.9983682793505751
sum 24.476548834600766
Average Recall av_acc 0.9790619533840307
```

```
# Sensitivity/Recall Calculation
sens_best=TP/(TP+FN)
sum=0
print("sens_best",sens_best)
for i in range(1,nr+1,1):
    sum=sum+pow(sens_best,i)
print("sum",sum)

# Average Sensitivity Calculation
av_sens=sum/nr
print("Average Recall av_sens",av_sens)
```

```
sens_best 0.9995064625470257
sum 24.840231840423645
Average Recall av_sens 0.9936092736169457
```

```
# Precision Calculation
prec_best=TP/(FP+TP)
sum=0
print("prec_best",prec_best)
for i in range(1,nr+1,1):
    sum=sum+pow(prec_best,i)
print("sum",sum)

# Average Precision Calculation
av_prec=sum/nr
print("Average Precision av_prec",av_prec)
```

```
prec_best 0.9984465405294786
sum 24.50134442201038
Average Precision av_prec 0.9800537768804152
```

GRU (Gated Recurrent Unit)

```
# The GRU architecture
regressorGRU = Sequential()
# First GRU layer with Dropout regularisation
GRU=tf.keras.layers.GRU
regressorGRU.add(GRU(units=77, return_sequences=True, input_shape=(1,77), activati
regressorGRU.add(Dropout(0.2))
# Second GRU layer
regressorGRU.add(GRU(units=77, return_sequences=True, input_shape=(1,77), activati
regressorGRU.add(Dropout(0.2))
# Third GRU layer
regressorGRU.add(GRU(units=77, return_sequences=True, input_shape=(1,77), activati
regressorGRU.add(Dropout(0.2))
# Fourth GRU layer
regressorGRU.add(GRU(units=77, activation='sigmoid'))
regressorGRU.add(Dropout(0.2))
# The output layer
regressorGRU.add(Dense(units=y.shape[1]))
# Compiling the RNN
#regressorGRU.compile(optimizer=SGD(learning_rate=0.01, decay=1e-7, momentum=0.9,
regressorGRU.compile(loss = 'binary_crossentropy', optimizer = "adam", metrics = [
# Fitting to the training set
regressorGRU.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 gru (GRU)                   (None, 1, 77)             36036

 dropout_2 (Dropout)         (None, 1, 77)             0

 gru_1 (GRU)                 (None, 1, 77)             36036

 dropout_3 (Dropout)         (None, 1, 77)             0

 gru_2 (GRU)                 (None, 1, 77)             36036

 dropout_4 (Dropout)         (None, 1, 77)             0

 gru_3 (GRU)                 (None, 77)                36036

 dropout_5 (Dropout)         (None, 77)                0

 dense_1 (Dense)             (None, 2)                 156

=================================================================
Total params: 144300 (563.67 KB)
Trainable params: 144300 (563.67 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
# Train the regressorGRU model usinf fit model
historyGRU = regressorGRU.fit(X_train, y_train, epochs = nr, validation_data= (X_t
historyGRU
```

```
Epoch 1/25
16854/16854 [==============================] - 174s 10ms/step - loss: 0.0434 -
Epoch 2/25
16854/16854 [==============================] - 167s 10ms/step - loss: 0.0279 -
Epoch 3/25
16854/16854 [==============================] - 179s 11ms/step - loss: 0.0299 -
<keras.src.callbacks.History at 0x7bcd25b75e40>
```

```
# Model's performance metrics
acc_GRU =regressorGRU.evaluate(X_test, y_test)
```

```
4214/4214 [==============================] - 15s 3ms/step - loss: 7.5212 - acc
```

```
ans=regressorGRU.predict(X_test)
```

```
4214/4214 [==============================] - 14s 3ms/step
```

```
y_pred.shape
```

```
(134827,)
```

```
y_test
```

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

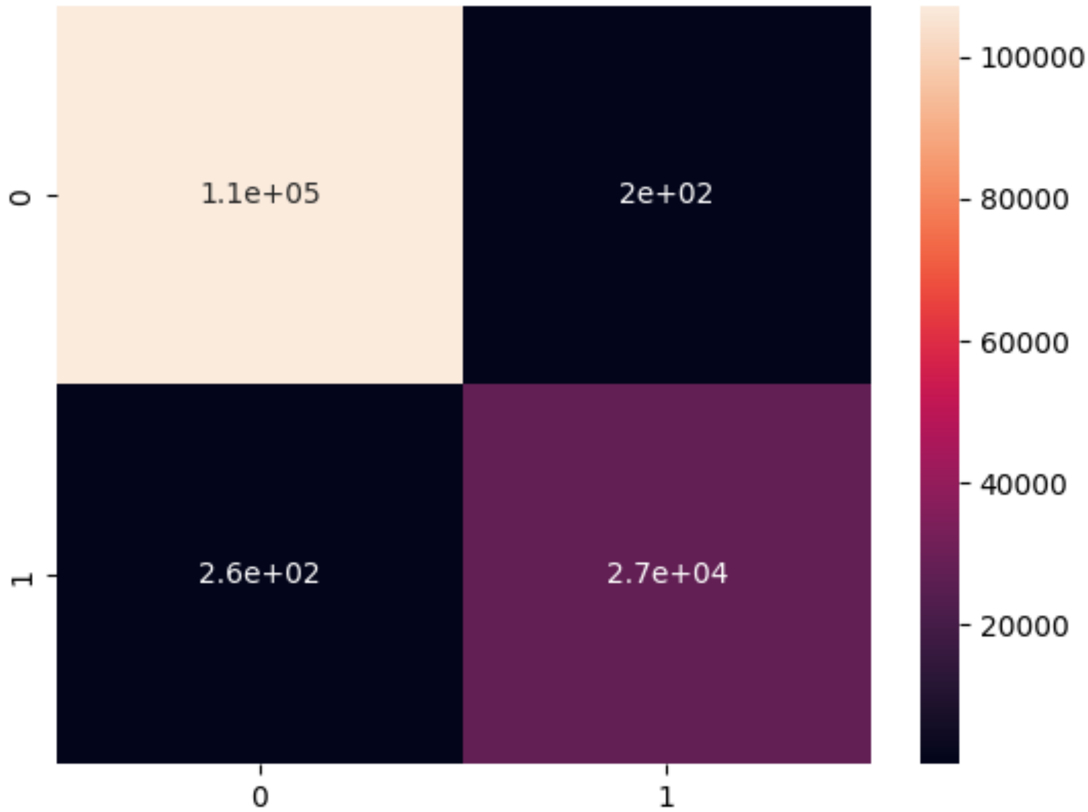
```
y_pred=np.argmax(ans, axis=1)

cm = confusion_matrix(y_test, y_pred)
```

```
cm
```

```
array([[107193,    195],
       [   265,  27174]])
```

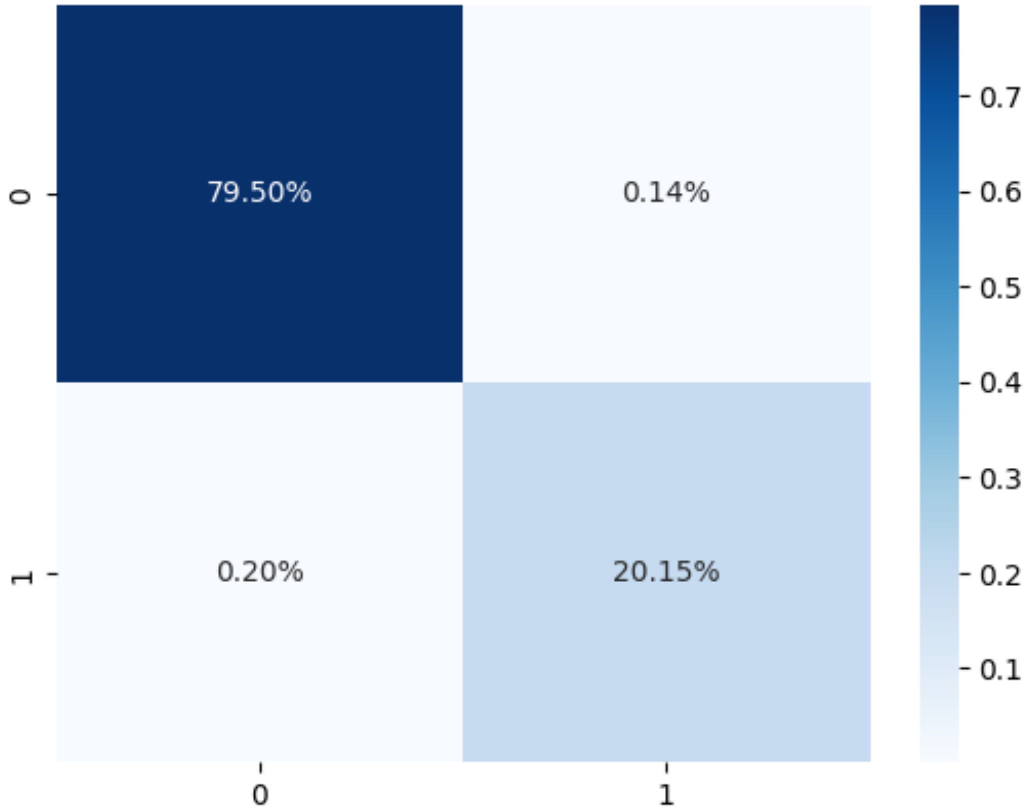
```
import matplotlib.pyplot as plt
import seaborn as sns
sns.heatmap(cm, annot=True)
```

```
<Axes: >
```



```
sns.heatmap(cm/np.sum(cm), annot=True,
            fmt='.2%', cmap='Blues')
```

```
<Axes: >
```




```
TP=cm[0][0]
FN=cm[0][1]
FP=cm[1][0]
TN=cm[1][1]
```

```
acc_best=(TP+TN)/(TP+FN+FP+TN)
print("acc_best",acc_best)
sum=0
for i in range(1,nr+1,1):
    sum=sum+pow(acc_best,i)
print("sum",sum)
av_acc=sum/nr
print("Average Recall av_acc",av_acc)
```

```
acc_best 0.9965882204602936
sum 23.92085135682399
Average Recall av_acc 0.9568340542729596
```

```
sens_best=TP/(TP+FN)
sum=0
print("sens_best",sens_best)
for i in range(1,nr+1,1):
    sum=sum+pow(sens_best,i)
print("sum",sum)
av_sens=sum/nr
print("Average Recall av_sens",av_sens)
```

```
sens_best 0.9981841546541513
sum 24.418334427200204
Average Recall av_sens 0.9767333770880081
```

```
prec_best=TP/(FP+TP)
sum=0
print("prec_best",prec_best)
for i in range(1,nr+1,1):
    sum=sum+pow(prec_best,i)
print("sum",sum)
av_prec=sum/nr
print("Average Precision av_prec",av_prec)
```

```
prec_best 0.9975339202292989
sum 24.214114301180537
Average Precision av_prec 0.9685645720472215
```