SOURCE CODE FOR DETECTION OF DOS, DDOS AND BOTNET ATTACKS :

#========================== IMPORT PACKAGES ============================

import pandas as pd

from sklearn import preprocessing

from sklearn.model\_selection import train\_test\_split

from sklearn import metrics

from sklearn import linear\_model

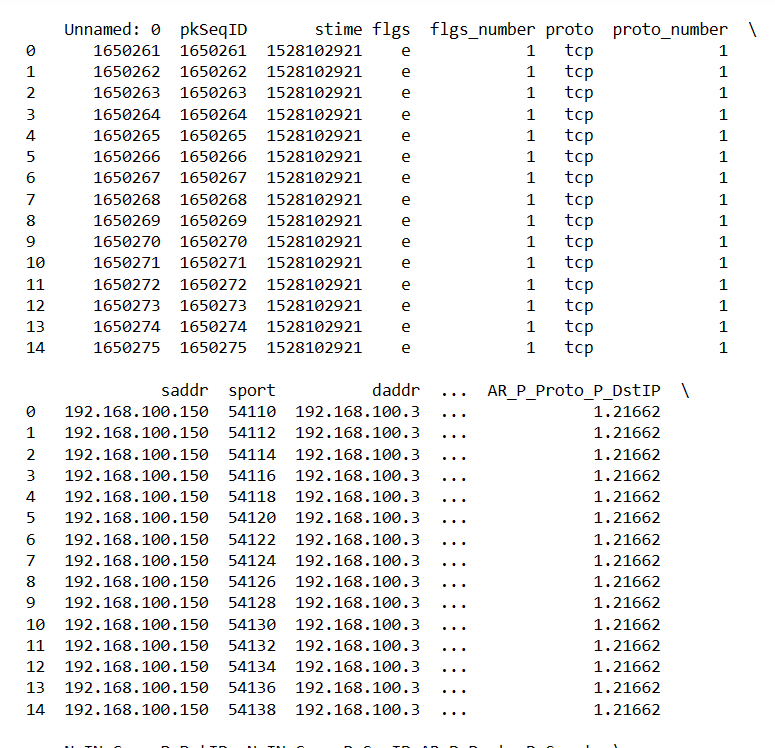
import warnings

warnings.filterwarnings('ignore')

* Now Extract the Network Packets From the Device Through Wireshark.
* Extract as Excel Sheet and Read that data.

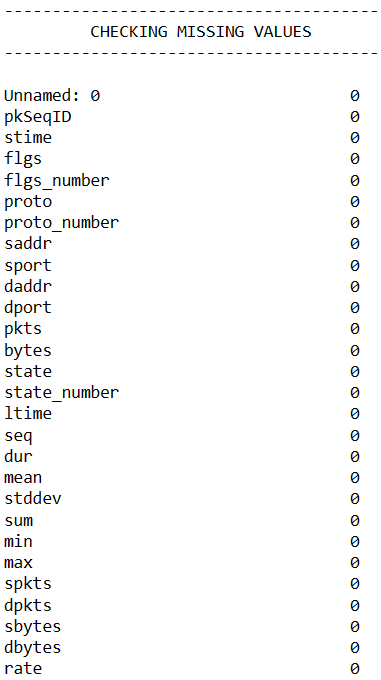
dataframe=pd.read\_csv("Bot-Iot.csv")

Print(dataframe)



**# Checking Missing Values:**

print(dataframe.isnull().sum())



Label Encoding :

print("------------------------------------------")

print("           BEFORE LABEL ENCODING          ")

print("------------------------------------------")

print()

print(dataframe['category'].head(20))

label\_encoder = preprocessing.LabelEncoder()

#dataframe['category'] = label\_encoder.fit\_transform(dataframe['category'])

#dataframe['proto'] = label\_encoder.fit\_transform(dataframe['proto'])

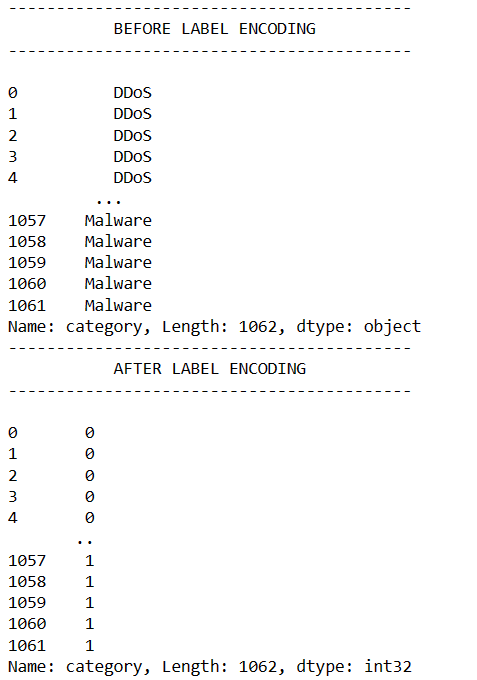
dataframe = dataframe.astype(str).apply(label\_encoder.fit\_transform)

print("------------------------------------------")

print("           AFTER LABEL ENCODING           ")

print("------------------------------------------")

print(dataframe['category'].head(20))



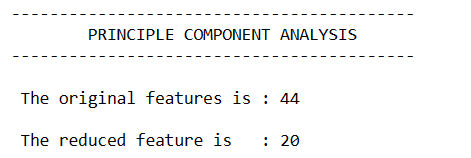
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=0)

**PCA :**

from sklearn.decomposition import PCA

pca = PCA(n\_components = 4)

X\_train1 = pca.fit\_transform(X)



**Code For Detection :**

import numpy as np

from sklearn import metrics

from tensorflow.keras.layers import Input, Dense, Dropout, LSTM

from tensorflow.keras.models import Sequential

# Assuming X\_train and y\_train are the training data

# Reshape the input data to match the model's expected input shape

x\_train = np.expand\_dims(X\_train, axis=-1) # Shape: (number\_of\_samples, sequence\_length, 1)

# Build the LSTM model

model = Sequential()

model.add(LSTM(units=50, input\_shape=(x\_train.shape[1], x\_train.shape[2]), return\_sequences=True, kernel\_initializer="uniform"))

model.add(Dropout(0.2))

model.add(LSTM(units=5, activation='relu', return\_sequences=False))

model.add(Dropout(0.2))

model.add(Dense(units=3, activation='relu'))

model.add(Dense(units=1, activation='sigmoid')) # Sigmoid for binary classification

model.compile(loss="binary\_crossentropy", optimizer='adam', metrics=['accuracy', 'mae'])

model.summary()

# Train the model

history = model.fit(x\_train, y\_train, epochs=5, batch\_size=2, verbose=2)

# Make predictions on the training data

y\_pred\_train = model.predict(x\_train)

# Convert predictions to binary classes (0 or 1)

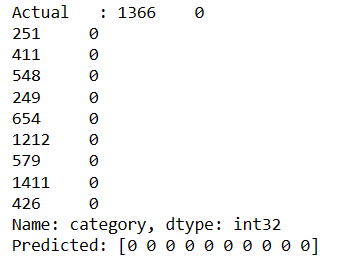
y\_pred\_train\_classes = (y\_pred\_train > 0.5).astype('int').reshape(-1)

# Print actual and predicted values for comparison

print()

print("Actual :", y\_train[:10]) # Print the first 10 actual values

print("Predicted:", y\_pred\_train\_classes[:10]) # Print the first 10 predicted values



**# FINAL PREDICTION**

