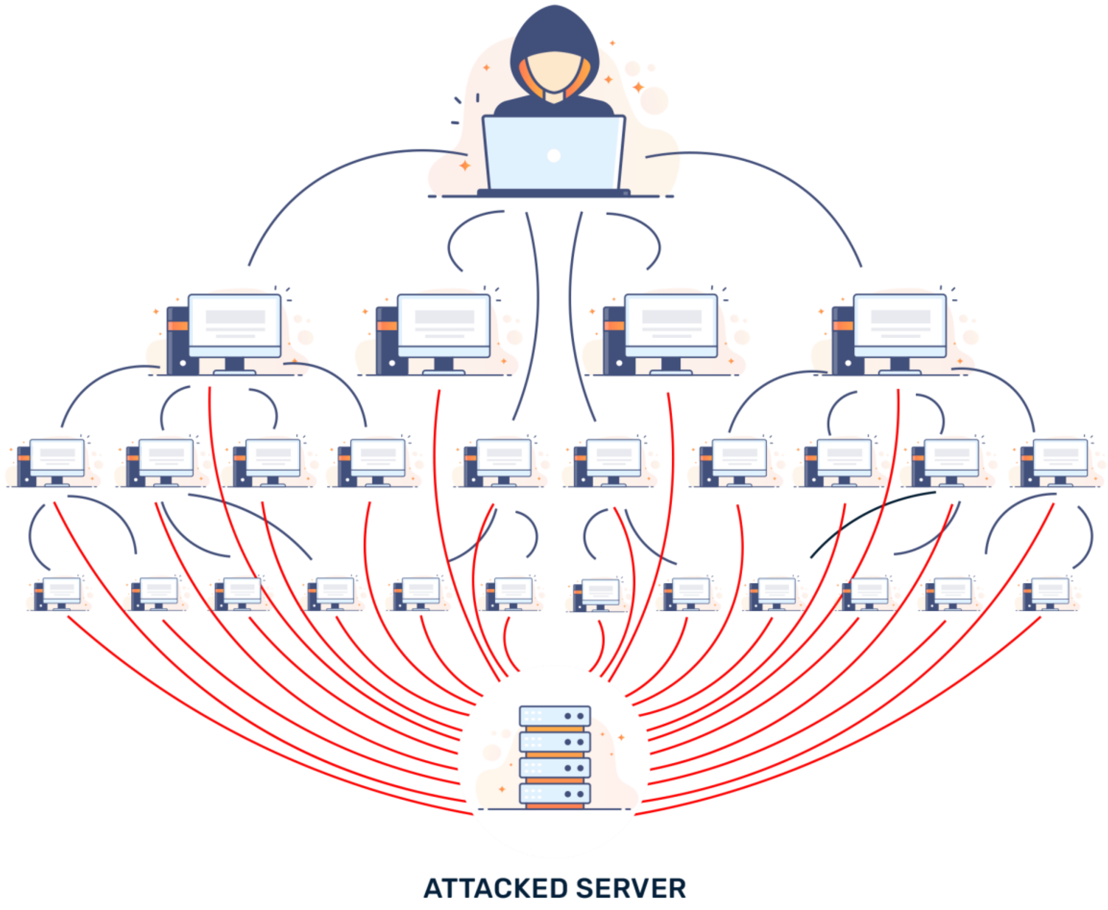
DDOS ATTACK DETECTION

G-42



B21087- Anubhav Singh
B21288- Deepika Tanania
B21318- Rishika
B21022- Shambhabi Dhar
B21026- Vallabhi Upadhyay

PROBLEM STATEMENT

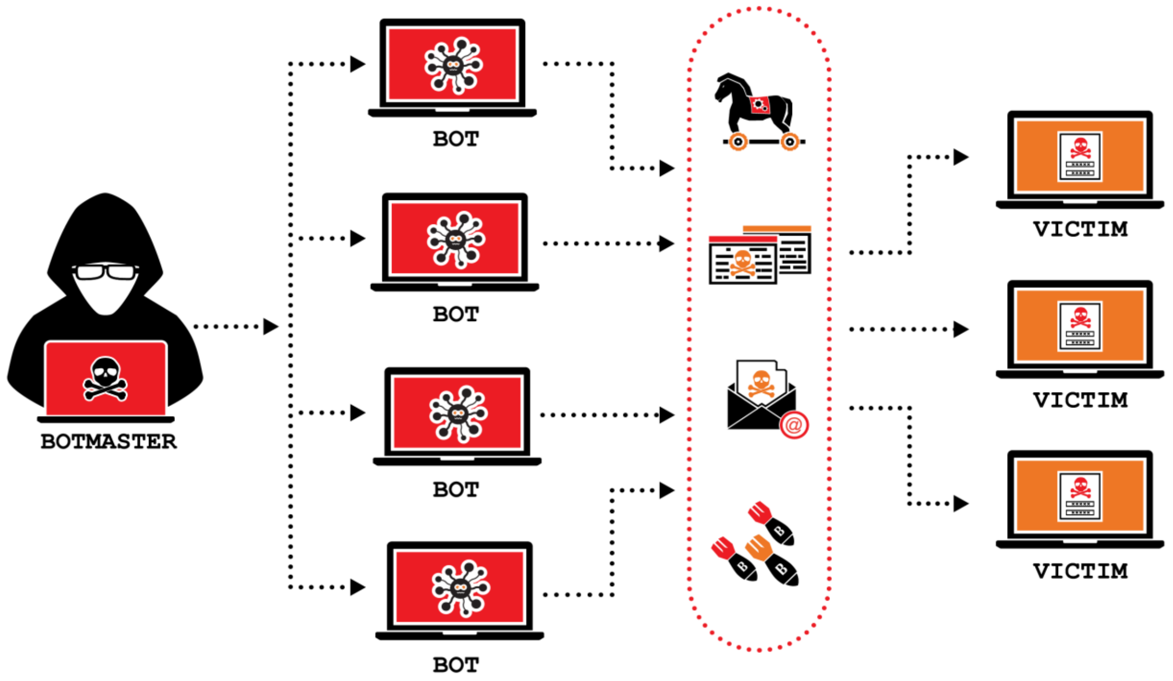
Create a platform that can detect DDoS attacks.

Description:

Distributed Denial of Service attack (DDoS) is the most dangerous attack in the field of network security. DDoS attack halts normal functionality of critical services of various online applications. Systems under DDoS attacks remain busy with false requests (Bots) rather than providing services to legitimate users. These attacks are increasing day by day and have become more and more sophisticated. So, it has become difficult to detect these attacks and secure online services from these attacks.

What is DDOS Attack?

A distributed denial-of-service (DDoS) attack is a malicious attempt to disrupt the normal traffic of a targeted server, service or network by overwhelming the target or its surrounding infrastructure with a flood of Internet traffic.



How does a DDoS attack work?

DDoS attacks are carried out with networks of Internet-connected machines. These networks consist of computers and other devices (such as IoT devices)which have been infected with malware, allowing them to be controlled remotely by an attacker. These individual devices are referred to as bots (or zombies), and a group of bots is called a botnet. Once a botnet has been established, the attacker is able to direct an attack by sending remote instructions to each bot. When a victim's server or network is targeted by the botnet, each bot sends requests to the target's IP Address, potentially causing the server or network to become overwhelmed, resulting in a denial-of-service to normal traffic. Because each bot is a legitimate Internet device, separating the attack traffic from normal traffic can be difficult.

How to identify a DDoS attack?

The most obvious symptom of a DDoS attack is a site or service suddenly becoming slow or unavailable. But since a number of causes — such a legitimate spike in traffic — can create similar performance issues, further investigation is usually required. Traffic analytics tools can help you spot some of these tell-tale signs of a DDoS attack: Suspicious amounts of traffic originating from a single IP address or IP range
A flood of traffic from users who share a single behavioural profile, such as device type, geolocation, or web browser version. An unexplained surge in requests to a single page or endpoint. Odd traffic patterns such as spikes at odd hours of the day or patterns that appear to be unnatural (e.g. a spike every 10 minutes). There are other, more specific signs of DDoS attack that can vary depending on the type of attack.

METHODOLOGY

Dataset Used: SDN Dataset (https://www.kaggle.com/code/aikenkazin/ddos-attack-detection-classification/data)

Algorithms used:

- Random Forest
- KNN
- Logistic Regression

Comparison of algorithms is done using:

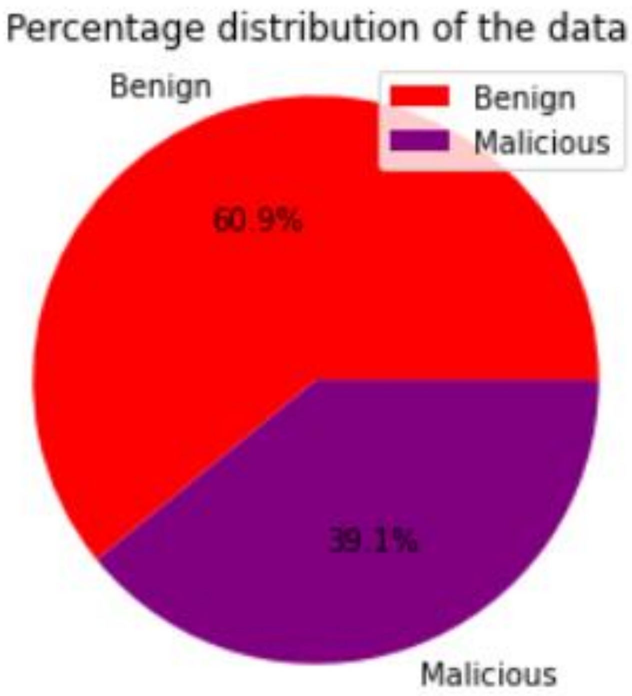
- Accuracy Score
- Confusion Matrix
- F1- Score

LIBRARIES USED

```
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
from matplotlib.pyplot import figure
import seaborn as sns

from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import f1_score as f1
from sklearn.neighbors import KNeighborsClassifier              KNN

from sklearn.linear_model import LogisticRegression             Logistic Regression
from sklearn.ensemble import RandomForestClassifier             Random Forest
from sklearn.decomposition import PCA
```

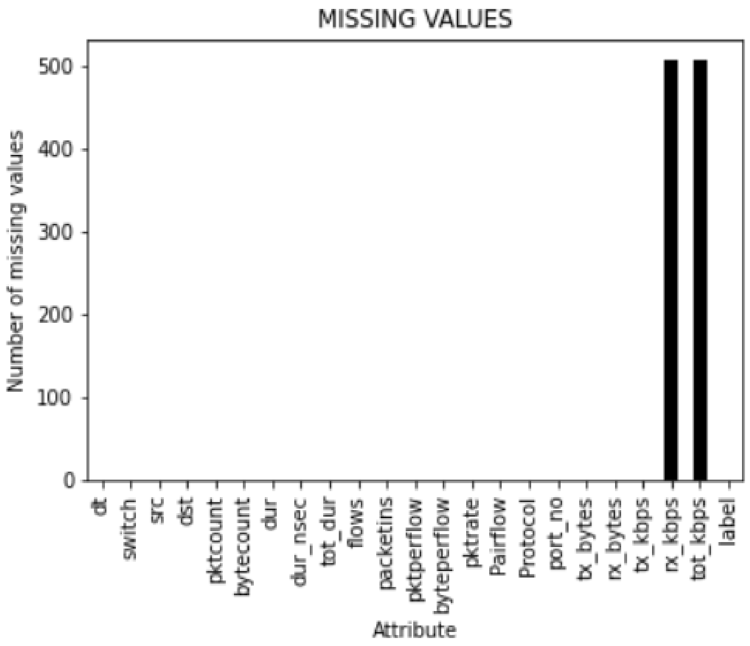
PIE CHART OF LABELS

```
labels = ["Benign", "Malicious"]
counts = [dt.label.value_counts()[0], dt.label.value_counts()[1]]
# plt.figure(figsize = (13,8))
plt.pie(counts, labels= labels, radius=1.1, colors=['red', 'purple'], labeldistance=1.1, autopct='%1.1f%%')
plt.legend()
plt.title("Percentage distribution of the data")
plt.show()
```



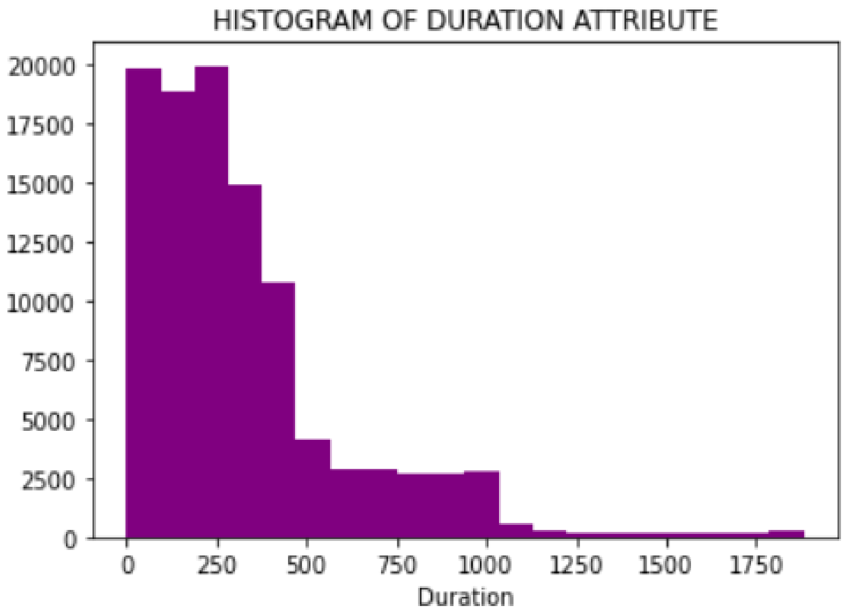
MISSING VALUES IN THE DATA

```
dt.isna().sum().plot.bar(color='black')
plt.title("MISSING VALUES")
plt.xlabel("Attribute")
plt.ylabel("Number of missing values")
plt.show()
```

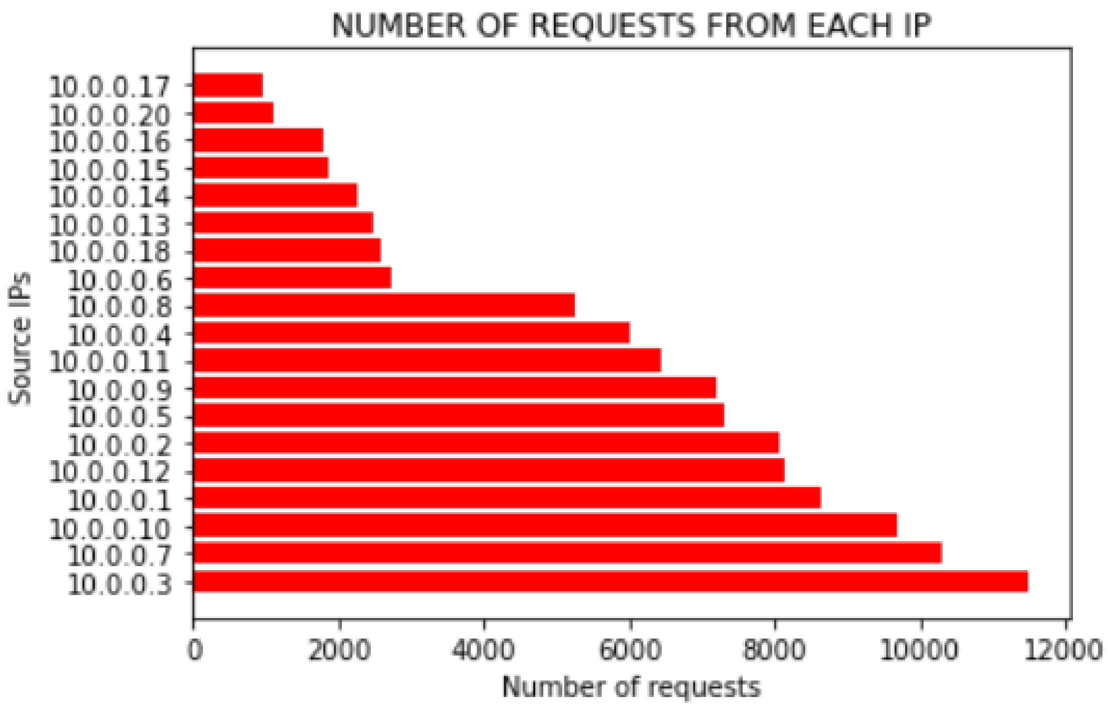


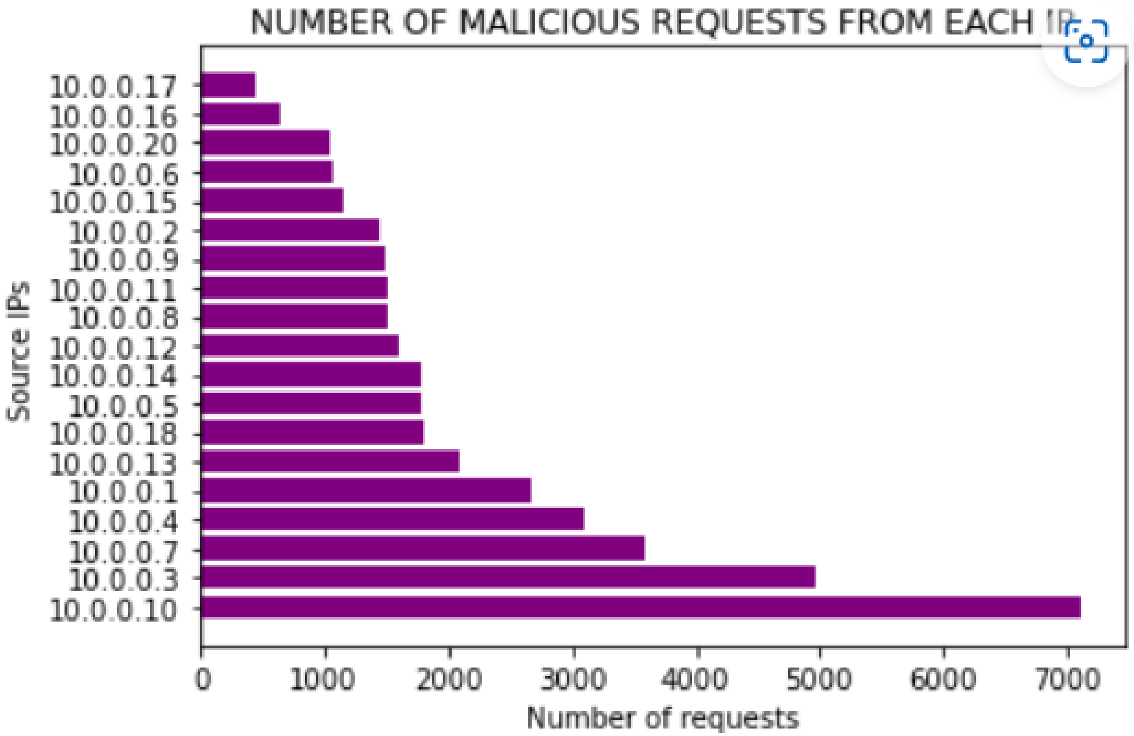
DURATIONS OF REQUESTS

```
plt.hist(dt.dur, bins=20, color='PURPLE')
plt.title('HISTOGRAM OF DURATION ATTRIBUTE')
plt.xlabel("Duration")
plt.show()
```

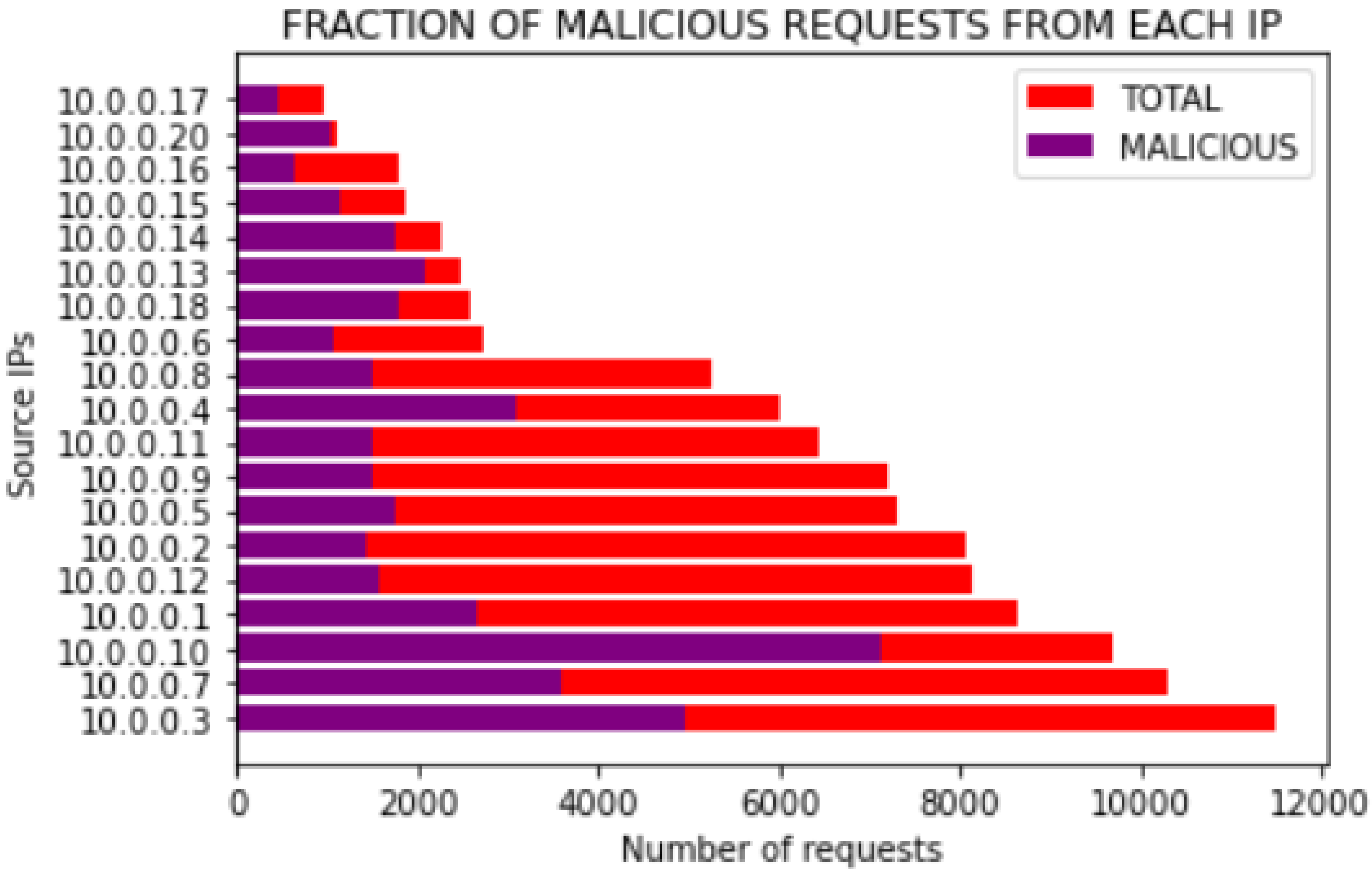


TOTAL AND MALICIOUS REQUESTS

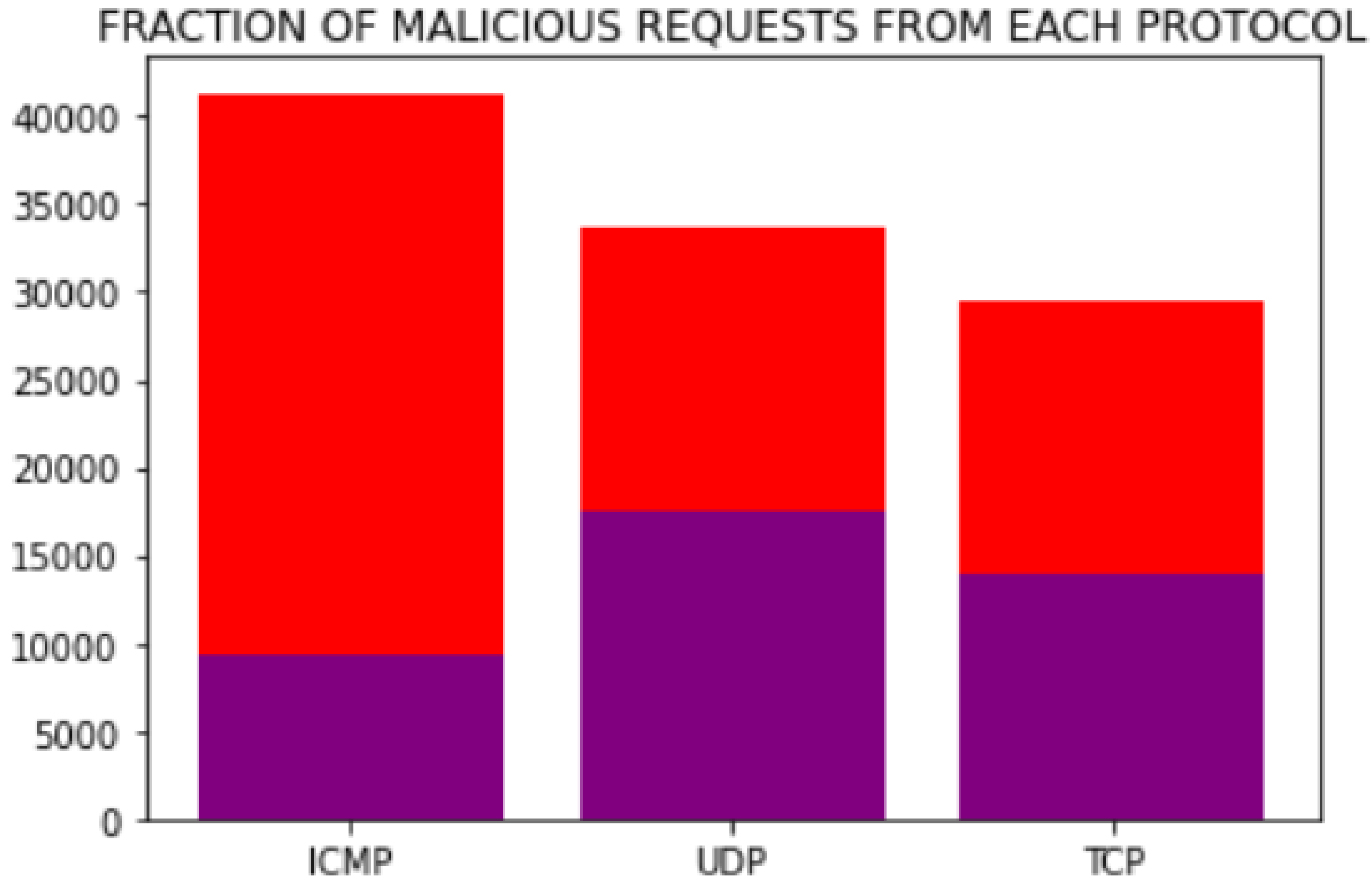




COMPARISON BETWEEN TOTAL REQUESTS AND MALICIOUS REQUESTS



COMPARISON BETWEEN TOTAL REQUESTS AND MALICIOUS REQUESTS FROM EACH PROTOCOL



DATA PREPROCESSING AND SPLITTING

```
dt_0 = dt.copy()
dt_0.dropna(inplace=True)
#dropping the NULL values
dumb_dt = pd.get_dummies(dt_0)
st = StandardScaler()
st.fit(dumb_dt)
dt_1 = st.transform(dumb_dt)
dt_1 = pd.DataFrame(dt_1)
dt_1.columns = dumb_dt.columns
dt_1.drop(['label'], axis=1, inplace=True)
X_train, X_test, Y_train, Y_test = train_test_split(dt_1, dt_0.label, random_state=42, test_size=0.3)
```

LOGISTIC REGRESSION

```
solvers = ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga']
acc_l = []
f1_l = []
for solver in solvers:
    lr = LogisticRegression(C=0.03, solver=solver).fit(X_train,Y_train)
    pred_lr = lr.predict(X_test)
    acc_lr = accuracy_score(Y_test, pred_lr)
    print(f"Accuracy score using {solver} solver: {acc_lr}\n")
    acc_l.append(acc_lr)
    f1_l.append(f1(Y_test, pred_lr))
best_solver = solvers[acc_l.index(max(acc_l))]
lr = LogisticRegression(C=0.03, solver=best_solver).fit(X_train, Y_train)
pred_lr = lr.predict(X_test)
acc_lr = accuracy_score(Y_test, pred_lr)
print(f"Accuracy score of Logistic Regression using the best solver '{best_solver}': {acc_lr}\n")
print(f"CLASSIFICATION REPORT:\n {classification_report(pred_lr, Y_test)}")
maxacc_lr = max(acc_l)
maxf1_lr  = max(f1_l)
```

RESULTS OF LOGISTIC REGRESSION

```
Accuracy score using newton-cg solver: 0.7504494093477144

Accuracy score using lbfgs solver: 0.7505136106831022

Accuracy score using liblinear solver: 0.7505457113507961

Accuracy score using sag solver: 0.7504494093477144

Accuracy score using saga solver: 0.7504494093477144

Accuracy score of Logistic Regression using the best solver 'liblinear': 0.7505457
113507961

CLASSIFICATION REPORT:
               precision    recall  f1-score   support

           0       0.85      0.77      0.81     20961
           1       0.60      0.72      0.65     10191

    accuracy                           0.75     31152
   macro avg       0.72      0.74      0.73     31152
weighted avg       0.77      0.75      0.76     31152
```

KNN CLASSIFIER

```
K=[5,7,13,19]
acc_l=[]
f1l=[]
for i in K:
    knn= KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train, Y_train)
    pred_knn=knn.predict(X_test)
    acc_l.append(accuracy_score(Y_test,pred_knn))
    f1l.append(f1(Y_test,pred_knn))
print(f"Maximum accurcay score for KNN, K= {K[acc_l.index(max(acc_l))]}: {max(acc_l)}\n")
print(f"CLASSIFICATION REPORT:\n{classification_report(pred_knn, Y_test)}")
maxacc_knn = max(acc_l)
maxf1_knn = max(f1l)
```

RESULTS OF KNN CLASSIFIER

```
Maximum accurcay score for KNN, K= 15: 0.9788135593220338

CLASSIFICATION REPORT:
              precision    recall  f1-score   support

           0       0.98      0.97      0.98     19195
           1       0.95      0.97      0.96     11957

    accuracy                           0.97     31152
   macro avg       0.97      0.97      0.97     31152
weighted avg       0.97      0.97      0.97     31152
```

RANDOM FOREST

```
RF = RandomForestClassifier(n_jobs=-1, n_estimators=500, min_samples_split=10, criterion='gini',max_features='auto',oob_score=Tru
RF.fit(X_train, Y_train)
pred_rf = RF.predict(X_test)
acc_rf = accuracy_score(Y_test, pred_rf)
print(f"Accuracy score for Random Forest: {acc_rf}\n")
print(f"CLASSIFICATION REPORT:\n{classification_report(pred_rf, Y_test)}")
maxacc_rf = acc_rf
maxf1_rf = f1(Y_test, pred_rf)
```

RESULTS OF RANDOM FOREST

Accuracy score for Random Forest: 1.0

CLASSIFICATION REPORT:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	18986
1	1.00	1.00	1.00	12166
accuracy			1.00	31152
macro avg	1.00	1.00	1.00	31152
weighted avg	1.00	1.00	1.00	31152

DIMENSION REDUCTION

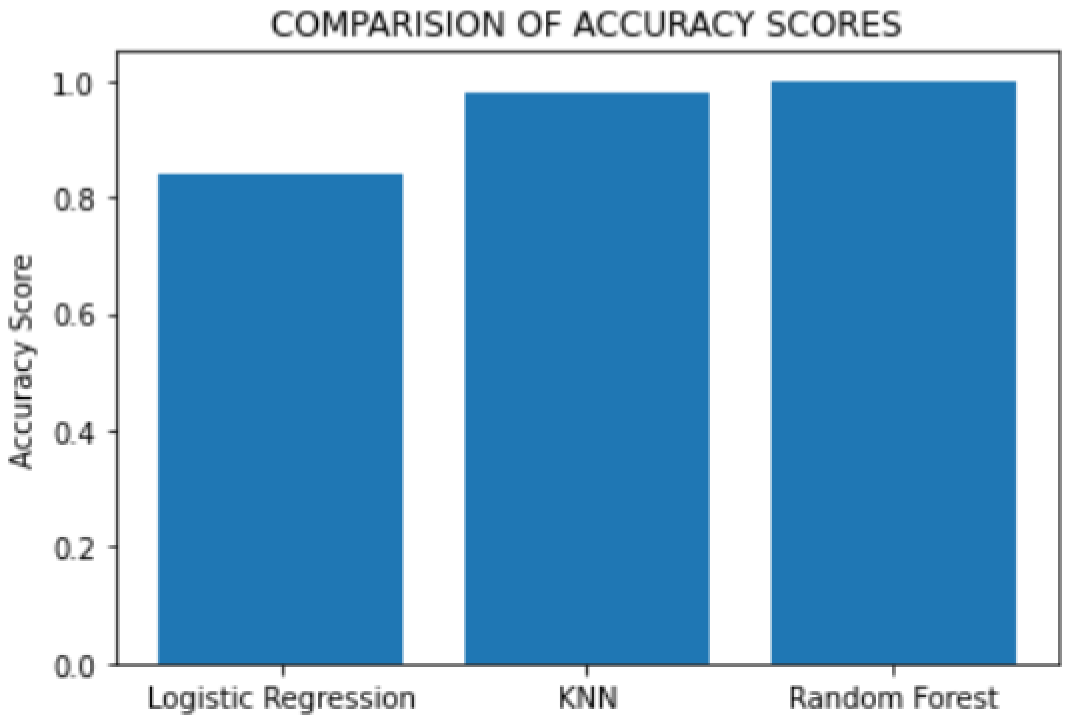
IS DIMENSION REDUCTION ADVANTAGEOUS?

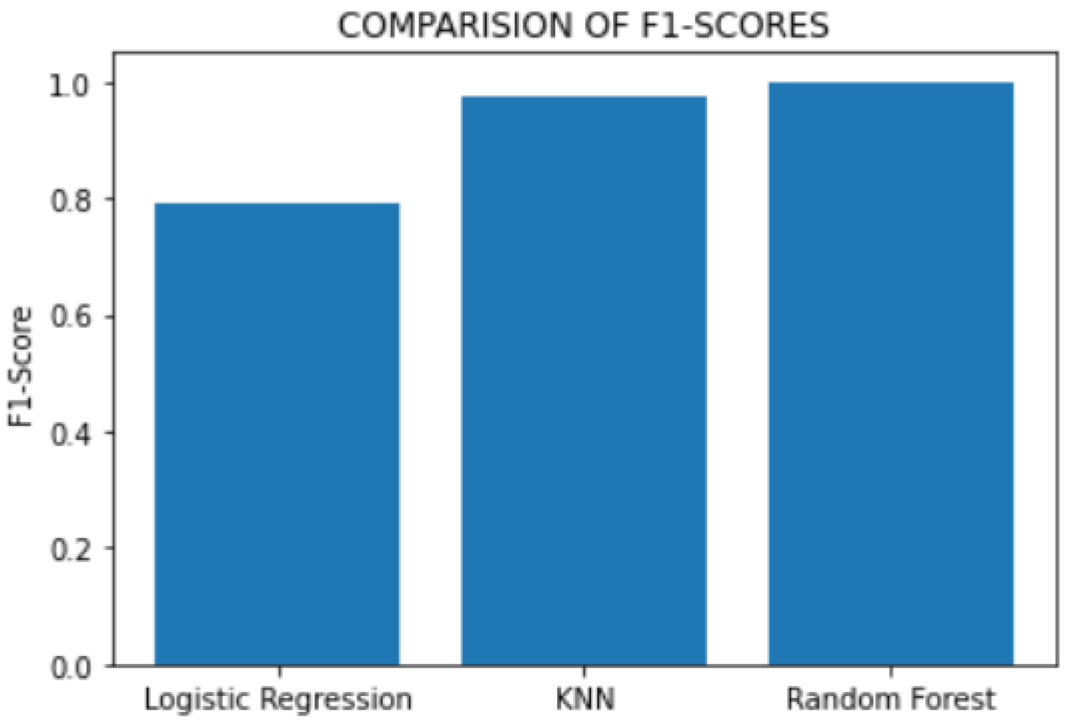
```
dt_2 = dt_1.copy()
comp = [2, 5, 10, 13, 15, 19, 21]
max_acc = []
for i in comp:
    pca = PCA(n_components=i)
    pca.fit(dt_2)
    pca_dt = pca.transform(dt_2)
    X_train, X_test, Y_train, Y_test = train_test_split(pca_dt, dumb_dt.label, random_state=42, test_size=0.3)
    solvers = ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga']
    results_lr = []
    acc_l = []
    for solver in solvers:
        lr = LogisticRegression(C=0.03, solver=solver).fit(X_train,Y_train)
        pred_lr = lr.predict(X_test)
        acc_lr = accuracy_score(Y_test, pred_lr)
        acc_l.append(acc_lr)
    max_acc.append(max(acc_l))
print(f"Maximum Accuracy score using PCA for 3
      components = {max(max_acc)}")
```

```
Maximum Accuracy score using PCA for 3 components = 0.7263418079096046
```

PCA REDUCED DATA IS GIVING VERY LOW ACCURACY. HENCE, CLASSIFICATION WITHOUT DATA REDUCTION IS BETTER

WHICH ALGORITHM IS THE BEST ?





RANDOM FOREST, AS IT HAS THE MAXIMUM ACCURACY SCORE

Thank You.