VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

***Intrusion Detection System***

using

***Machine Learning Algorithms***

A PROJECT DONE

for

Information Security Management – [F1 slot]

in

B. Tech – Information Technology and Engineering

By

18BIT0126- ROHITH REDDY (narurohith.reddy2018@vitstudent.ac.in)

18BIT0094- SAI SUCHETAN REDDY

SLOT- F1 (BOTH 18BIT0126,0094)

Under the Guidance of

SUMAIYA THASSEN I

isumaiyathassen@vit.ac.in

Associate Professor Grade 1, SITE

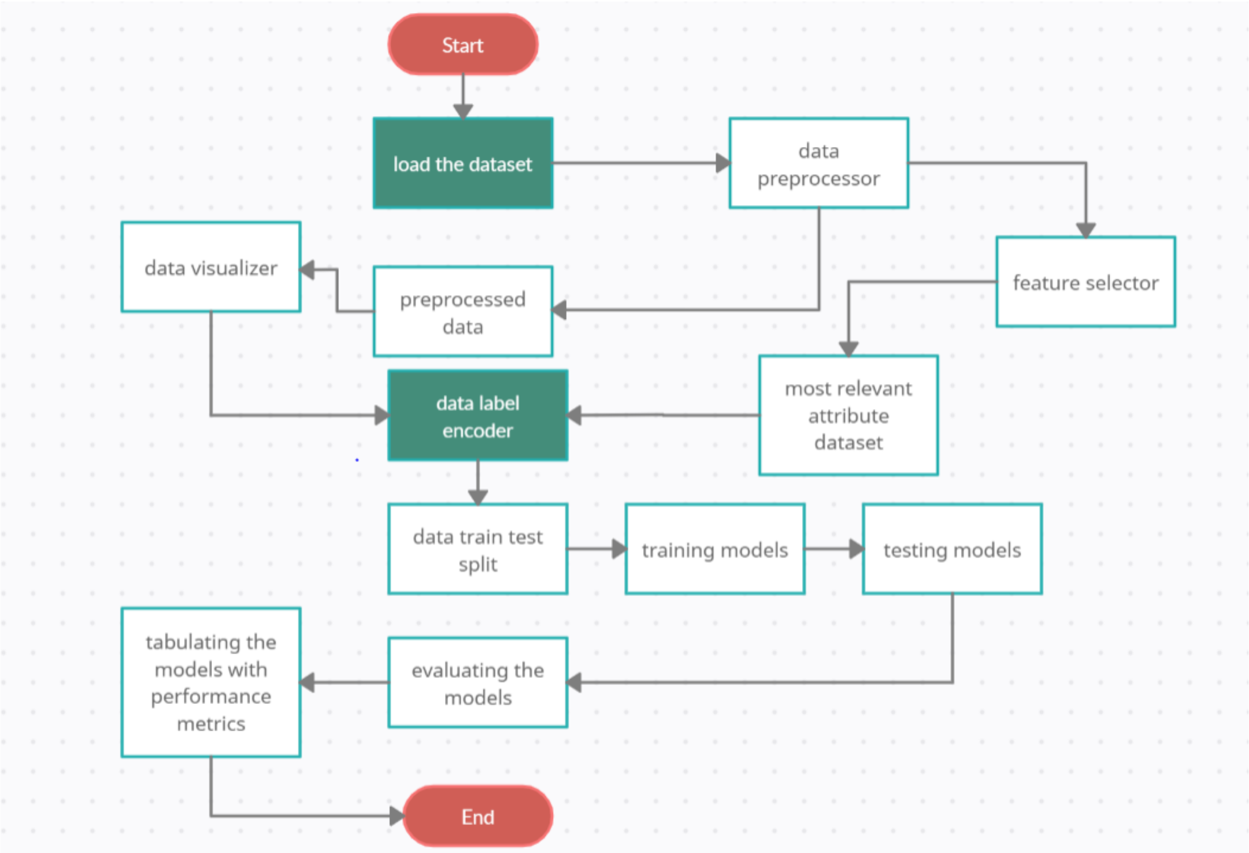
# ABSTRACT

Intrusion detection system (IDS) is one of the implemented solutions against harmful attacks. Furthermore, attackers always keep changing their tools and techniques. However, implementing an accepted IDS system is also a challenging task.

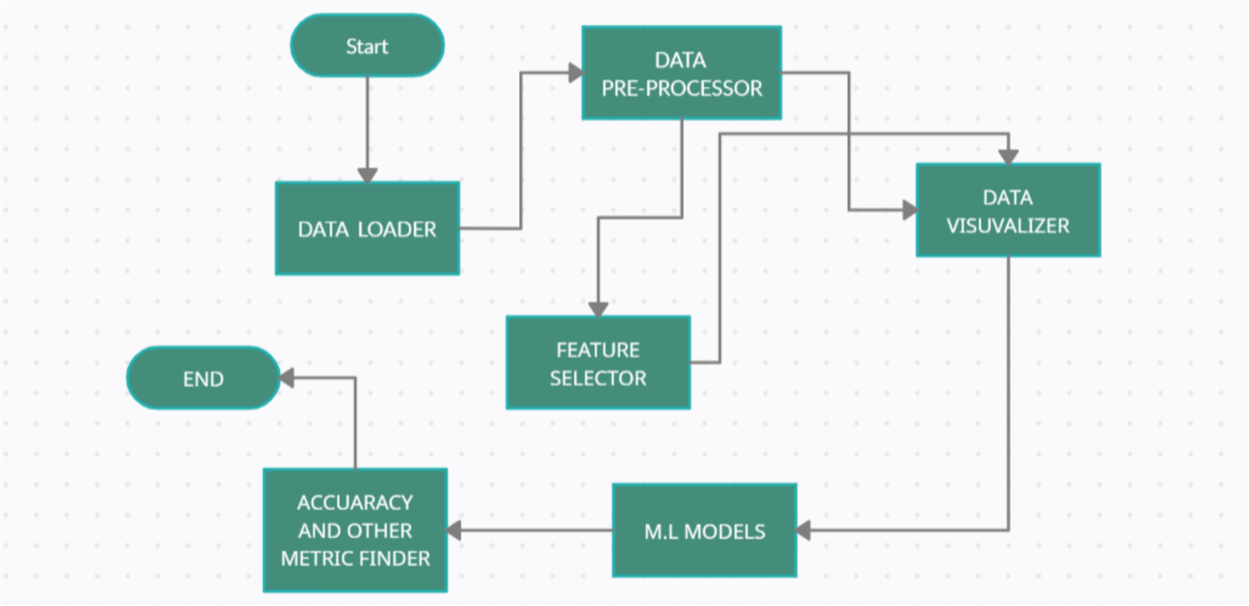
Intrusion Detect Systems (IDSs) are a range of cybersecurity-based technology initially developed to detect vulnerabilities and exploits against a target host. The sole use of the IDS is to identify malicious activities and policy violations. Existing intrusion detection systems rely heavily on human analysts to differentiate intrusive from non-intrusive network traffic. The large and growing amount of data confronts the analysts with an overwhelming task, making the automation of aspects of this task necessary.

In this project, we demonstrate an approach for network Intrusion Detection System (IDS) for cyber security using Machine Learning (ML) techniques using some Supervised Classification Algorithms. And we produce a detail report about the accuracy and the performance of each model we use in our project.

# WORKFLOW DIAGRAM



# ARCHITECTURE DIAGRAM



# intrusion Detection using Machine Learning techniques

# 18BIT0126

# 18BIT0094

In [1]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn.metrics import classification_report
from sklearn import metrics
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import  (precision_score, recall_score, f1_score, accurac
y_score, mean_squared_error, mean_absolute_error)
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
```

# loading the data from the user machine

In [2]:
```
data=pd.read_csv(r'C:\Users\asus\Downloads\NF-UQ-NIDS.csv')
data.describe()
```

Out[2]:
```
        L4_SRC_PORT    L4_DST_PORT    PROTOCOL       L7_PROTO       IN_BYTES       OUT_BYTES
count   1.199489e+07   1.199489e+07   1.199489e+07   1.199489e+07   1.199489e+07   1.199489e+07
mean    4.124519e+04   8.964758e+03   8.661742e+00   1.932050e+01   3.983727e+03   9.489449e+03
std     2.108654e+04   1.772082e+04   6.404263e+00   3.504857e+01   1.662256e+05   2.933544e+05
min     0.000000e+00   0.000000e+00   0.000000e+00   0.000000e+00   0.000000e+00   0.000000e+00
25%     3.299400e+04   5.300000e+01   6.000000e+00   0.000000e+00   6.800000e+01   0.000000e+00
50%     5.061000e+04   4.430000e+02   6.000000e+00   0.000000e+00   2.320000e+02   1.560000e+02
75%     5.581700e+04   3.389000e+03   6.000000e+00   7.000000e+00   1.440000e+03   1.873000e+03
max     6.553500e+04   6.553500e+04   2.550000e+02   2.510000e+02   2.282235e+08   2.432197e+08
```

```
In [3]: data
```

```
Out[3]:
           IPV4_SRC_ADDR  L4_SRC_PORT   IPV4_DST_ADDR  L4_DST_PORT  PROTOCOL  L7_PF

       0   149.171.126.0        62073      59.166.0.5        56082         6

       1   149.171.126.2        32284      59.166.0.5         1526         6

       2   149.171.126.0           21      59.166.0.1        21971         6

       3      59.166.0.1        23800   149.171.126.0        46893         6

       4      59.166.0.5        63062   149.171.126.2           21         6

     ...             ...          ...             ...          ...       ...

11994888  192.168.100.46           80   192.168.100.5           80         6

11994889   192.168.100.5            0   192.168.100.3            0         6

11994890   192.168.100.7          365   192.168.100.3          565        17

11994891   192.168.100.3        50850    13.54.166.67         8883         6     22

11994892   192.168.100.6        49160 192.168.100.149         4444         6

11994893 rows × 15 columns
```

# data information about datatypes

In [4]:
```
import warnings
warnings.filterwarnings('ignore')
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11994893 entries, 0 to 11994892
Data columns (total 15 columns):
IPV4_SRC_ADDR                 object
L4_SRC_PORT                   int64
IPV4_DST_ADDR                 object
L4_DST_PORT                   int64
PROTOCOL                      int64
L7_PROTO                      float64
IN_BYTES                      int64
OUT_BYTES                     int64
IN_PKTS                       int64
OUT_PKTS                      int64
TCP_FLAGS                     int64
FLOW_DURATION_MILLISECONDS    int64
Label                         int64
Attack                        object
Dataset                       object
dtypes: float64(1), int64(10), object(4)
memory usage: 1.3+ GB
```

# dropping unwanted columns

In [5]:
```
data=data.drop(['Dataset'],axis=1)
data
```

Out[5]:

| | IPV4_SRC_ADDR | L4_SRC_PORT | IPV4_DST_ADDR | L4_DST_PORT | PROTOCOL | L7_PF |
|---|---|---|---|---|---|---|
| **0** | 149.171.126.0 | 62073 | 59.166.0.5 | 56082 | 6 | |
| **1** | 149.171.126.2 | 32284 | 59.166.0.5 | 1526 | 6 | |
| **2** | 149.171.126.0 | 21 | 59.166.0.1 | 21971 | 6 | |
| **3** | 59.166.0.1 | 23800 | 149.171.126.0 | 46893 | 6 | |
| **4** | 59.166.0.5 | 63062 | 149.171.126.2 | 21 | 6 | |
| **...** | ... | ... | ... | ... | ... | |
| **11994888** | 192.168.100.46 | 80 | 192.168.100.5 | 80 | 6 | |
| **11994889** | 192.168.100.5 | 0 | 192.168.100.3 | 0 | 6 | |
| **11994890** | 192.168.100.7 | 365 | 192.168.100.3 | 565 | 17 | |
| **11994891** | 192.168.100.3 | 50850 | 13.54.166.67 | 8883 | 6 | 22 |
| **11994892** | 192.168.100.6 | 49160 | 192.168.100.149 | 4444 | 6 | |

11994893 rows × 14 columns

In [6]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11994893 entries, 0 to 11994892
Data columns (total 14 columns):
IPV4_SRC_ADDR                  object
L4_SRC_PORT                    int64
IPV4_DST_ADDR                  object
L4_DST_PORT                    int64
PROTOCOL                       int64
L7_PROTO                       float64
IN_BYTES                       int64
OUT_BYTES                      int64
IN_PKTS                        int64
OUT_PKTS                       int64
TCP_FLAGS                      int64
FLOW_DURATION_MILLISECONDS     int64
Label                          int64
Attack                         object
dtypes: float64(1), int64(10), object(3)
memory usage: 1.3+ GB
```

# checking for any null values in the dataset

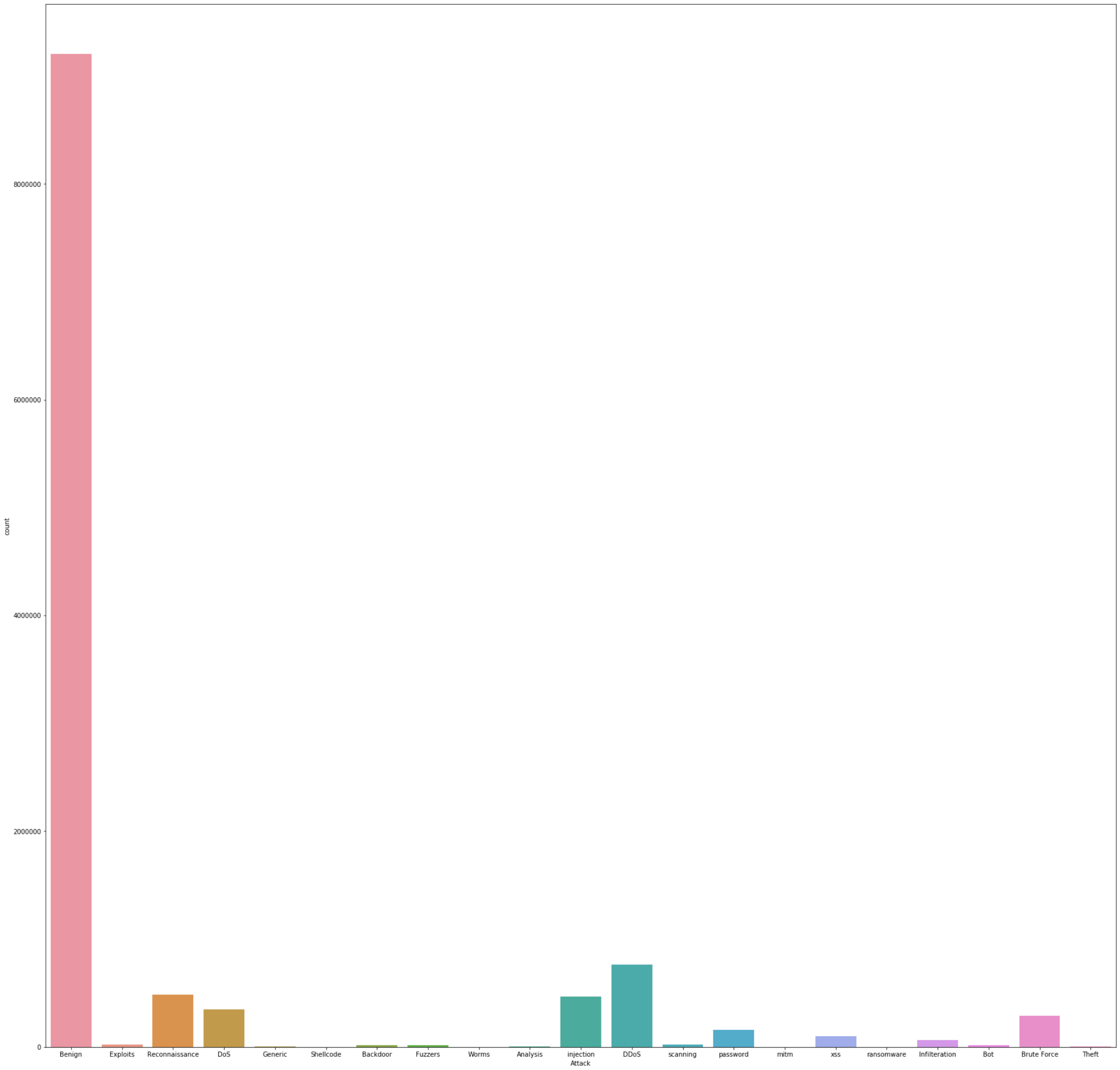
In [7]:
```
data.isnull().sum()
```

Out[7]:
```
IPV4_SRC_ADDR                  0
L4_SRC_PORT                    0
IPV4_DST_ADDR                  0
L4_DST_PORT                    0
PROTOCOL                       0
L7_PROTO                       0
IN_BYTES                       0
OUT_BYTES                      0
IN_PKTS                        0
OUT_PKTS                       0
TCP_FLAGS                      0
FLOW_DURATION_MILLISECONDS     0
Label                          0
Attack                         0
dtype: int64
```

# types of attacks that are in the dataset

In [8]:
```
import seaborn as sns
from matplotlib import pyplot as plt
fig, ax=plt.subplots(figsize=(30,30))
sns.countplot(ax=ax,x="Attack", data=data)
```

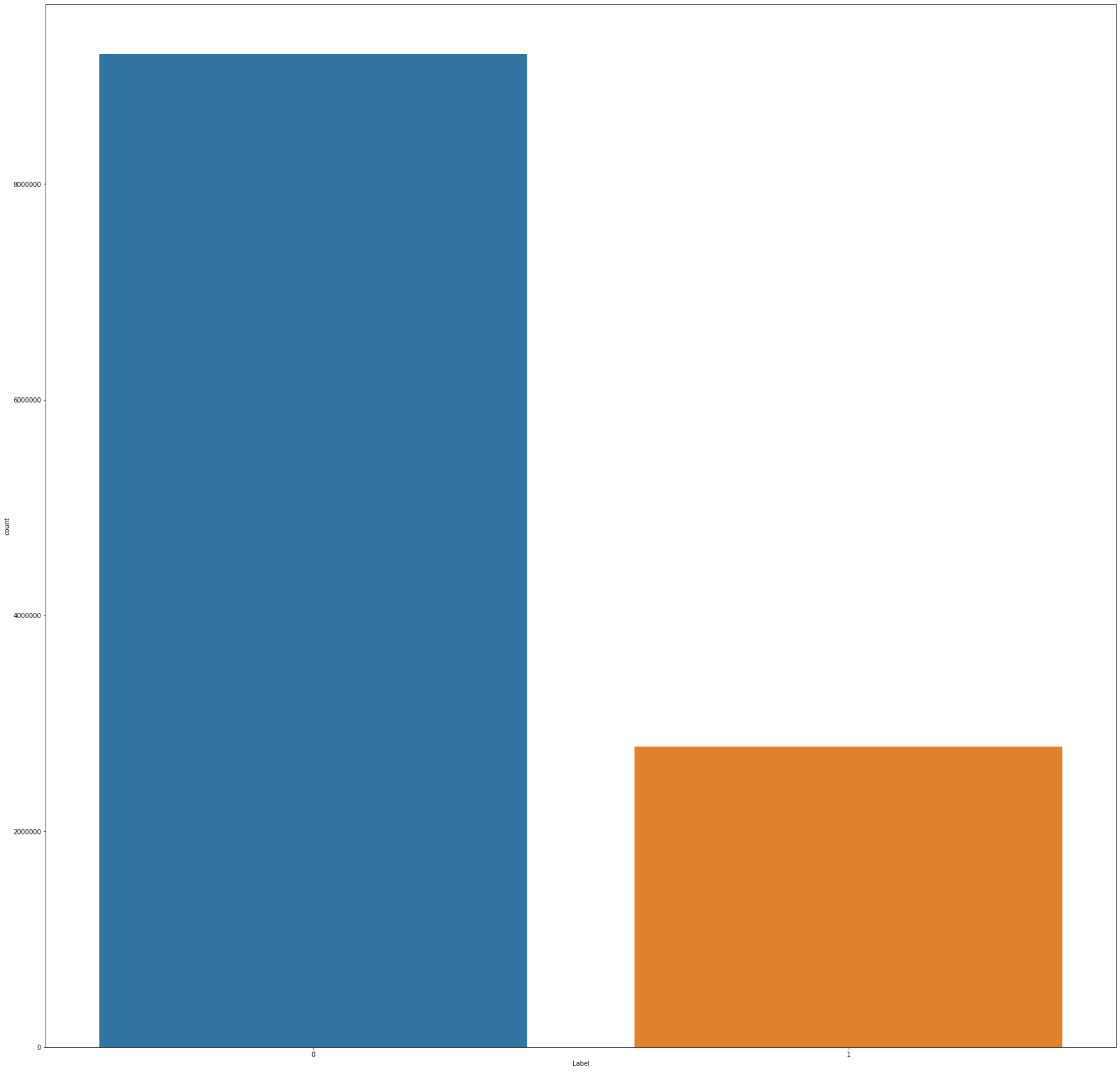
Out[8]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x21a4ca91bc8>
```



# plotting the labels 0-normal 1-intrusion

In [9]:
```
import seaborn as sns
from matplotlib import pyplot as plt
fig, ax=plt.subplots(figsize=(30,30))
sns.countplot(ax=ax,x="Label", data=data)
```

Out[9]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x21a4d25e388>
```



# categorical data into a seperate dataframe

In [10]:
```
cdata=data[['IPV4_SRC_ADDR','IPV4_DST_ADDR','Attack']]
cdata
```

Out[10]:

| | IPV4_SRC_ADDR | IPV4_DST_ADDR | Attack |
|---|---|---|---|
| **0** | 149.171.126.0 | 59.166.0.5 | Benign |
| **1** | 149.171.126.2 | 59.166.0.5 | Benign |
| **2** | 149.171.126.0 | 59.166.0.1 | Benign |
| **3** | 59.166.0.1 | 149.171.126.0 | Benign |
| **4** | 59.166.0.5 | 149.171.126.2 | Benign |
| **...** | ... | ... | ... |
| **11994888** | 192.168.100.46 | 192.168.100.5 | Benign |
| **11994889** | 192.168.100.5 | 192.168.100.3 | Benign |
| **11994890** | 192.168.100.7 | 192.168.100.3 | Benign |
| **11994891** | 192.168.100.3 | 13.54.166.67 | Benign |
| **11994892** | 192.168.100.6 | 192.168.100.149 | Theft |

11994893 rows × 3 columns

# label encoding the categorical data

In [11]:
```
from sklearn.preprocessing import LabelEncoder
cdata.columns
for label in cdata.columns:
    cdata[label]=LabelEncoder().fit(cdata[label]).transform(cdata[label])
```

In [12]:
```
cdata
```

Out[12]:

| | IPV4_SRC_ADDR | IPV4_DST_ADDR | Attack |
|---|---|---|---|
| **0** | 18035 | 25333 | 2 |
| **1** | 18047 | 25333 | 2 |
| **2** | 18035 | 25329 | 2 |
| **3** | 64615 | 5246 | 2 |
| **4** | 64619 | 5258 | 2 |
| **...** | ... | ... | ... |
| **11994888** | 42254 | 9060 | 2 |
| **11994889** | 42255 | 9057 | 2 |
| **11994890** | 42258 | 9057 | 2 |
| **11994891** | 42252 | 4508 | 2 |
| **11994892** | 42257 | 9054 | 13 |

11994893 rows × 3 columns

In [13]:
```
cdata.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11994893 entries, 0 to 11994892
Data columns (total 3 columns):
IPV4_SRC_ADDR    int32
IPV4_DST_ADDR    int32
Attack           int32
dtypes: int32(3)
memory usage: 137.3 MB
```

In [14]:

```
data=data.drop(['IPV4_SRC_ADDR','IPV4_DST_ADDR','Attack'],axis=1)
data
```

Out[14]:

| | L4_SRC_PORT | L4_DST_PORT | PROTOCOL | L7_PROTO | IN_BYTES | OUT_BYTES | IN_P |
|---|---|---|---|---|---|---|---|
| **0** | 62073 | 56082 | 6 | 0.000 | 9672 | 416 | |
| **1** | 32284 | 1526 | 6 | 0.000 | 1776 | 104 | |
| **2** | 21 | 21971 | 6 | 1.000 | 1842 | 1236 | |
| **3** | 23800 | 46893 | 6 | 0.000 | 528 | 8824 | |
| **4** | 63062 | 21 | 6 | 1.000 | 1786 | 2340 | |
| **...** | ... | ... | ... | ... | ... | ... | |
| **11994888** | 80 | 80 | 6 | 7.000 | 2330065 | 0 | |
| **11994889** | 0 | 0 | 6 | 0.000 | 1054423 | 0 | |
| **11994890** | 365 | 565 | 17 | 0.000 | 62422 | 0 | |
| **11994891** | 50850 | 8883 | 6 | 222.178 | 11300 | 1664 | |
| **11994892** | 49160 | 4444 | 6 | 0.000 | 40102320 | 37280 | |

11994893 rows × 11 columns

In [15]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11994893 entries, 0 to 11994892
Data columns (total 11 columns):
L4_SRC_PORT                   int64
L4_DST_PORT                   int64
PROTOCOL                      int64
L7_PROTO                      float64
IN_BYTES                      int64
OUT_BYTES                     int64
IN_PKTS                       int64
OUT_PKTS                      int64
TCP_FLAGS                     int64
FLOW_DURATION_MILLISECONDS    int64
Label                         int64
dtypes: float64(1), int64(10)
memory usage: 1006.7 MB
```

In [16]:
```
fdata = pd.concat([data, cdata], axis=1)
fdata.head()
```

Out[16]:

| | L4_SRC_PORT | L4_DST_PORT | PROTOCOL | L7_PROTO | IN_BYTES | OUT_BYTES | IN_PKTS | O |
|---|---|---|---|---|---|---|---|---|
| **0** | 62073 | 56082 | 6 | 0.0 | 9672 | 416 | 11 | |
| **1** | 32284 | 1526 | 6 | 0.0 | 1776 | 104 | 6 | |
| **2** | 21 | 21971 | 6 | 1.0 | 1842 | 1236 | 26 | |
| **3** | 23800 | 46893 | 6 | 0.0 | 528 | 8824 | 10 | |
| **4** | 63062 | 21 | 6 | 1.0 | 1786 | 2340 | 32 | |

In [17]:
```
fdata.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11994893 entries, 0 to 11994892
Data columns (total 14 columns):
L4_SRC_PORT                  int64
L4_DST_PORT                  int64
PROTOCOL                     int64
L7_PROTO                     float64
IN_BYTES                     int64
OUT_BYTES                    int64
IN_PKTS                      int64
OUT_PKTS                     int64
TCP_FLAGS                    int64
FLOW_DURATION_MILLISECONDS   int64
Label                        int64
IPV4_SRC_ADDR                int32
IPV4_DST_ADDR                int32
Attack                       int32
dtypes: float64(1), int32(3), int64(10)
memory usage: 1.1 GB
```

In [18]:
```
x=fdata.drop(['Label'],axis=1)
y=fdata['Label']
```

# train test split

In [19]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.30,random_state
=0)
x_train.shape,y_train.shape,x_test.shape,y_test.shape
```

Out[19]:
```
((8396425, 13), (8396425,), (3598468, 13), (3598468,))
```

# multiple linear regression

In [20]:

```
#multiple linear regression
print("******************** multiple linear regression************************
*******")
from sklearn.linear_model import  LinearRegression
from sklearn.metrics import classification_report,confusion_matrix
from sklearn.metrics import accuracy_score, r2_score, mean_squared_error
print("***********making the model....***********")
regressor = LinearRegression()
print("***********training the model....***********")
regressor=regressor.fit(x_train, y_train)
print("***********making prediction please wait....***********")
y_pred = regressor.predict(x_test)
y_pred
print("***********printing confusion matrix and classification report....*****
******")
print(confusion_matrix(y_test, y_pred.round()))
print(classification_report(y_test, y_pred.round()))
```

```
******************** multiple linear regression******************************
*
***********making the model....***********
***********training the model....***********
***********making prediction please wait....***********
***********printing confusion matrix and classification report....***********
[[      0       0       0       0       0]
 [      5 2762471      14       4       2]
 [    357  178246  621893   35476       0]
 [      0       0       0       0       0]
 [      0       0       0       0       0]]
              precision    recall  f1-score   support

        -1.0       0.00      0.00      0.00         0
         0.0       0.94      1.00      0.97   2762496
         1.0       1.00      0.74      0.85    835972
         2.0       0.00      0.00      0.00         0
         3.0       0.00      0.00      0.00         0

    accuracy                           0.94   3598468
   macro avg       0.39      0.35      0.36   3598468
weighted avg       0.95      0.94      0.94   3598468
```

# naive bayes algorithm

In [21]:

```
from sklearn import metrics
print("**************the naive bayes algorithm used*****************")
gnb=GaussianNB()
print("***********training the model....***********")
gnb.fit(x_train,y_train)
print("***********making the predictions....***********")
y_pred=gnb.predict(x_test)
y_test.value_counts()
print("***********printing confusion matrix and classification report....*****
******")
cm=confusion_matrix(y_test,y_pred)

print(classification_report(y_test,y_pred))
print(confusion_matrix(y_test,y_pred))
```

```
**************the naive bayes algorithm used*****************
***********training the model....***********
***********making the predictions....***********
***********printing confusion matrix and classification report....***********
              precision    recall  f1-score   support

           0       0.94      0.96      0.95   2762496
           1       0.86      0.78      0.82    835972

    accuracy                           0.92   3598468
   macro avg       0.90      0.87      0.88   3598468
weighted avg       0.92      0.92      0.92   3598468

[[2658257  104239]
 [ 184495  651477]]
```

# decision tree algorithm

In [22]:

```
from sklearn import tree
import matplotlib.pyplot as plt
print("**************the decision tree was used*****************")
clf=DecisionTreeClassifier(criterion='gini',splitter='random',max_leaf_nodes=1
0,min_samples_leaf=5,max_depth=5,random_state=0)
print("***********training the model....***********")
clf.fit(x_train,y_train)
print("***********making the predictions....***********")
y_pred=clf.predict(x_test)
y_pred
print("***********printing the classification report please wait....**********
*")
print(classification_report(y_test,y_pred))
```

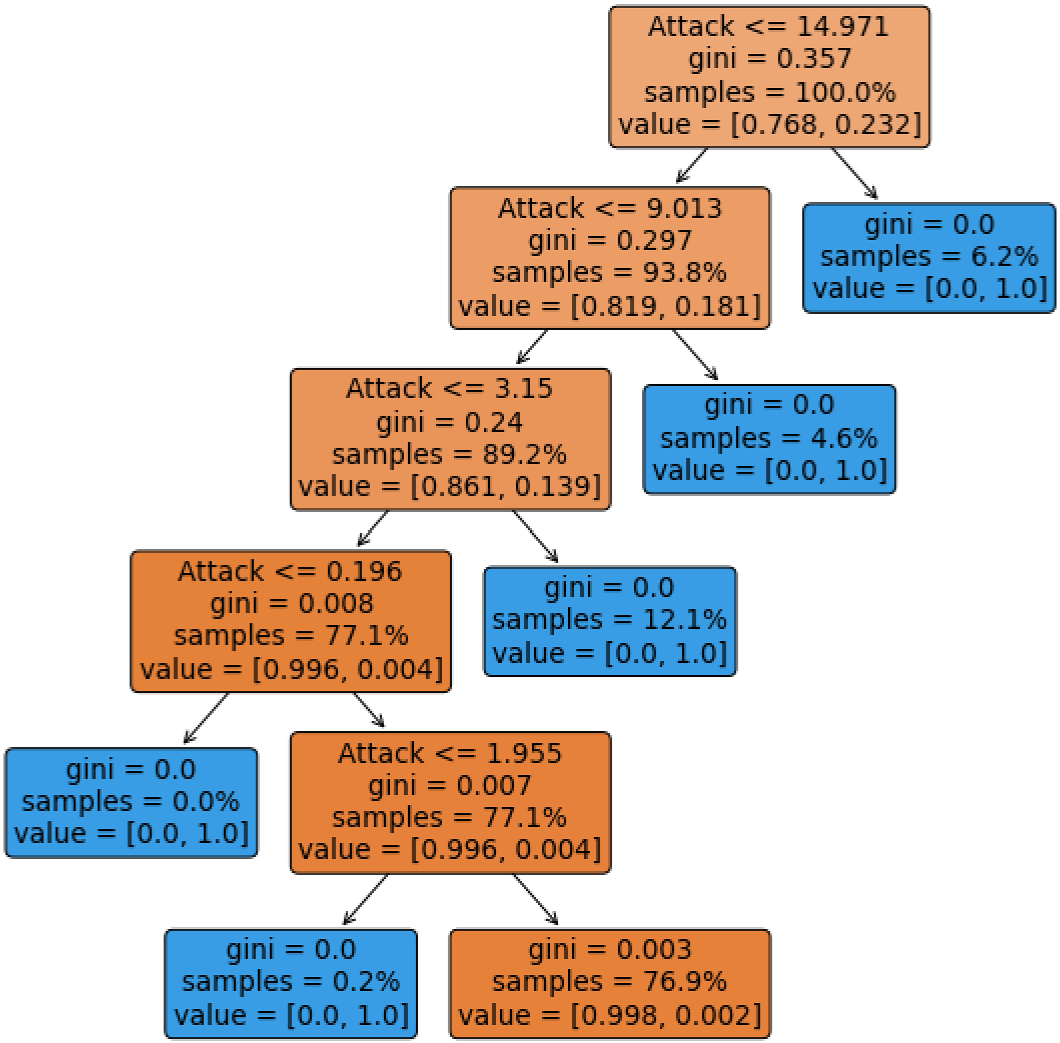
```
**************the decision tree was used*****************
***********training the model....***********
***********making the predictions....***********
***********printing the classification report please wait....***********
              precision    recall  f1-score   support

           0       1.00      1.00      1.00   2762496
           1       1.00      0.99      1.00    835972

    accuracy                           1.00   3598468
   macro avg       1.00      1.00      1.00   3598468
weighted avg       1.00      1.00      1.00   3598468
```

# decision tree visuvalization

In [23]:

```
print("***********making the decision tree....***********")
cols=list(x.columns.values)
plt.figure(figsize=(10,10))
tree.plot_tree(clf.fit(x,y),feature_names=cols,filled=True,precision=3,proportion=True,rounded=True)
plt.show()
```

```
***********making the decision tree....***********
```



# adaboost algorithm

In [24]:
```
print("************the ensemble ADABOOST was used*************")
from sklearn.ensemble import AdaBoostClassifier
model=DecisionTreeClassifier(criterion='entropy',max_depth=1,random_state=0)
print("***********making the model....***********")
Adaboost=AdaBoostClassifier(base_estimator=model,n_estimators=20,random_state=
0)
print("***********training the model....***********")
boostmodel=Adaboost.fit(x_train,y_train)
print("***********making the predictions please wait....***********")
y_pred=boostmodel.predict(x_test)
print("***********printing the accuracy of the model please wait....**********
*")
predictions=metrics.accuracy_score(y_test,y_pred)
print("accuracy: ",predictions)
```

```
************the ensemble ADABOOST was used*************
***********making the model....***********
***********training the model....***********
***********making the predictions please wait....***********
***********printing the accuracy of the model please wait....***********
accuracy:  1.0
```

# random forest algorithm

In [25]:
```
#random forest
print("************the random forest was used*************")
from sklearn.ensemble import RandomForestClassifier
print("***********making the model....***********")
classifier = RandomForestClassifier(n_estimators = 80, criterion = 'entropy',
random_state = 12)
print("***********training the model....***********")
classifier.fit(x_train, y_train)
# predict
print("***********making the predictions....***********")
y_pred = classifier.predict(x_test)
print("***********printing the accuracy....***********")
accuracy = accuracy_score(y_test, y_pred)
print("accuracy: ",accuracy)
```

```
************the random forest was used*************
***********making the model....***********
***********training the model....***********
***********making the predictions....***********
***********printing the accuracy....***********
accuracy:  1.0
```

# logistic regression algorithm

In [26]:

```
print("************the log regression was used*************")
from sklearn.linear_model import LogisticRegression
print("***********making the model....***********")
logmodel = LogisticRegression(max_iter=18000)
print("***********training the model....***********")
logmodel.fit(x_train,y_train)
print("***********making the predictions....***********")
predictions = logmodel.predict(x_test)
print("***********printing classication report and confusion matrix....*******
****")
print(classification_report(y_test, predictions))
print(confusion_matrix(y_test, predictions))
```

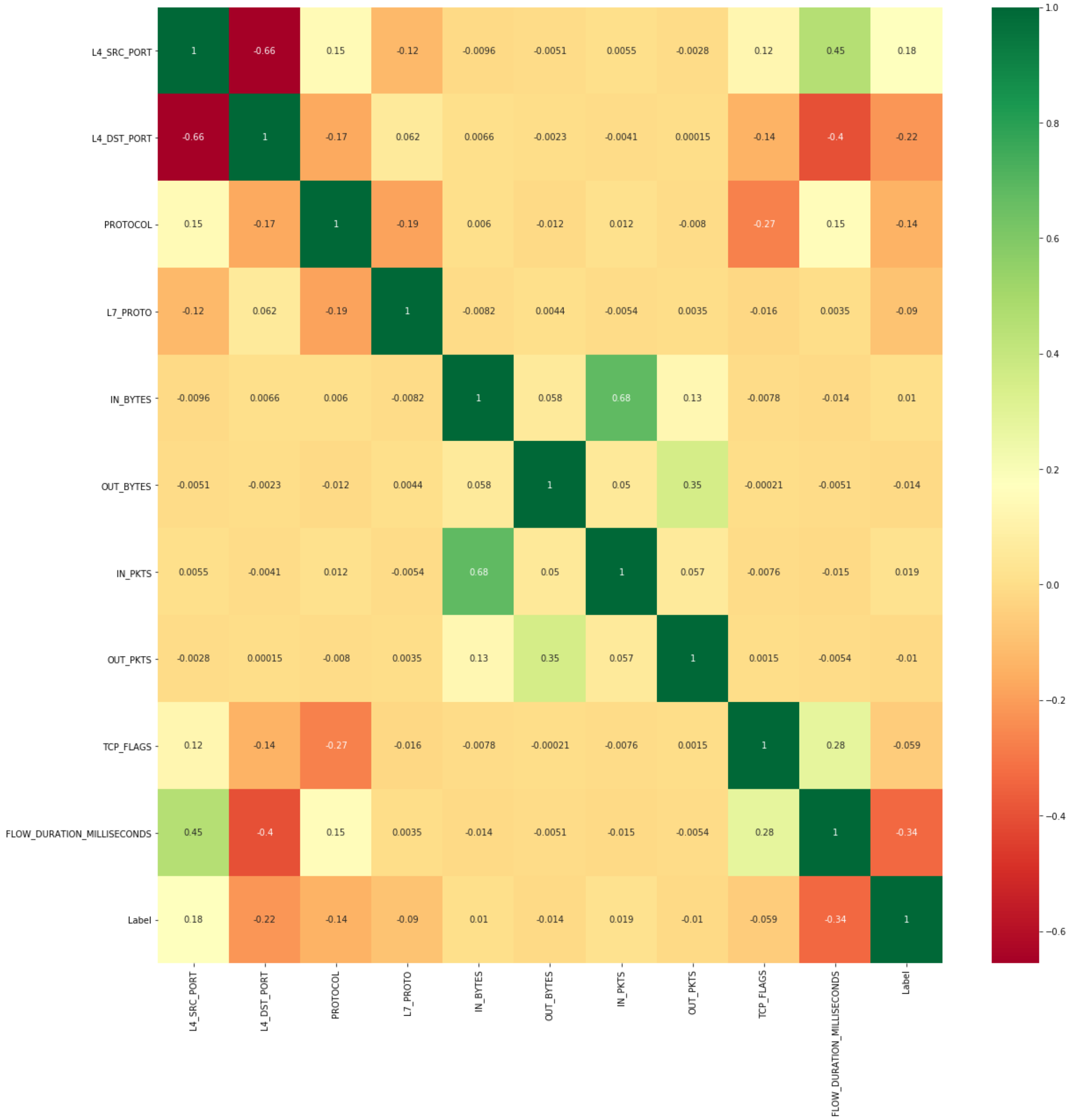
```
************the log regression was used*************
***********making the model....***********
***********training the model....***********
***********making the predictions....***********
***********printing classication report and confusion matrix....***********
              precision    recall  f1-score   support

           0       0.93      0.94      0.94   2762496
           1       0.80      0.78      0.79    835972

    accuracy                           0.90   3598468
   macro avg       0.87      0.86      0.86   3598468
weighted avg       0.90      0.90      0.90   3598468

[[2602019  160477]
 [ 183381  652591]]
```

In [ ]:

```
classifier=Sequential()
classifier.add(Dense(output_dim=49,init='he_uniform',activation='relu',input_d
im=41))
classifier.add(Dense(output_dim=24,init='he_uniform',activation='relu'))
classifier.add(Dense(output_dim=13,init='he_uniform',activation='relu'))
classifier.add(Dense(output_dim=1,init='glorot_uniforms',activation='sigmoid'
))
classifier.compile(optimizer='Adamax',loss='binary_crossentropy',metrics=['acc
uracy'])
model_history=classifier.fit(x_train,y_train,validation_split=0.33,batch_size=
1000,nb_epoch=10)
y_pred=classifier.predict(x_test)
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test,y_pred))
```

# correlation heatmap for feature selection

```
In [27]: import seaborn as sns
         X = fdata.drop(['Label'],axis=1)  #independent columns
         y = fdata['Label']    #target column i.e price range
         #get correlations of each features in dataset
         corrmat = data.corr()
         top_corr_features = corrmat.index
         plt.figure(figsize=(20,20))
         #plot heat map
         g=sns.heatmap(data[top_corr_features].corr(),annot=True,cmap="RdYlGn")
```



In [28]:
```
cor=fdata.corr()
cor_target=abs(cor["Label"])
relevant_features=cor_target[cor_target>-1]
relevant_features
```

Out[28]:
```
L4_SRC_PORT                    0.178658
L4_DST_PORT                    0.219108
PROTOCOL                       0.140932
L7_PROTO                       0.090210
IN_BYTES                       0.010478
OUT_BYTES                      0.013647
IN_PKTS                        0.018832
OUT_PKTS                       0.009963
TCP_FLAGS                      0.059306
FLOW_DURATION_MILLISECONDS     0.337871
Label                          1.000000
IPV4_SRC_ADDR                  0.032354
IPV4_DST_ADDR                  0.044257
Attack                         0.787343
Name: Label, dtype: float64
```

# selecting features with correlation>0.3

In [29]:
```
ndata=fdata[['FLOW_DURATION_MILLISECONDS','Attack']]
ndata
```

Out[29]:

| | FLOW_DURATION_MILLISECONDS | Attack |
|---|---|---|
| **0** | 15 | 2 |
| **1** | 0 | 2 |
| **2** | 1111 | 2 |
| **3** | 124 | 2 |
| **4** | 1459 | 2 |
| **...** | ... | ... |
| **11994888** | 4263037 | 2 |
| **11994889** | 4263062 | 2 |
| **11994890** | 4263062 | 2 |
| **11994891** | 4264935 | 2 |
| **11994892** | 4270068 | 13 |

11994893 rows × 2 columns

In [30]:
```
x=ndata
y=fdata['Label']
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.30,random_state
=0)
x_train.shape,y_train.shape,x_test.shape,y_test.shape
```

Out[30]:
```
((8396425, 2), (8396425,), (3598468, 2), (3598468,))
```

In [31]:
```
#multiple linear regression
print("******************** improved multiple linear regression**************
****************")
from sklearn.linear_model import  LinearRegression
from sklearn.metrics import classification_report,confusion_matrix
from sklearn.metrics import accuracy_score, r2_score, mean_squared_error
print("***********making the model....***********")
regressor = LinearRegression()
print("***********training the model....***********")
regressor=regressor.fit(x_train, y_train)
print("***********making prediction please wait....***********")
y_pred = regressor.predict(x_test)
y_pred
print("***********printing confusion matrix and classification report....*****
******")
print(confusion_matrix(y_test, y_pred.round()))
print(classification_report(y_test, y_pred.round()))
```

```
******************** improved multiple linear regression********************
**********
***********making the model....***********
***********training the model....***********
***********making prediction please wait....***********
***********printing confusion matrix and classification report....***********
[[2762496       0       0]
 [ 432728  366815   36429]
 [      0       0       0]]
              precision    recall  f1-score   support

         0.0       0.86      1.00      0.93   2762496
         1.0       1.00      0.44      0.61    835972
         2.0       0.00      0.00      0.00         0

    accuracy                           0.87   3598468
   macro avg       0.62      0.48      0.51   3598468
weighted avg       0.90      0.87      0.85   3598468
```

In [32]:

```
from sklearn import metrics
print("**************the improved naive bayes algorithm used*****************"
)
gnb=GaussianNB()
print("***********training the model....***********")
gnb.fit(x_train,y_train)
print("***********making the predictions....***********")
y_pred=gnb.predict(x_test)
y_test.value_counts()
print("***********printing confusion matrix and classification report....*****
******")
cm=confusion_matrix(y_test,y_pred)

print(classification_report(y_test,y_pred))
print(confusion_matrix(y_test,y_pred))
```

```
**************the improved naive bayes algorithm used*****************
***********training the model....***********
***********making the predictions....***********
***********printing confusion matrix and classification report....***********
              precision    recall  f1-score   support

           0       0.77      1.00      0.87   2762496
           1       0.00      0.00      0.00    835972

    accuracy                           0.77   3598468
   macro avg       0.38      0.50      0.43   3598468
weighted avg       0.59      0.77      0.67   3598468

[[2762496       0]
 [ 835972       0]]
```

In [33]:

```
print("************the modified log regression was used*************")
from sklearn.linear_model import LogisticRegression
print("***********making the model....***********")
logmodel = LogisticRegression(max_iter=18000)
print("***********training the model....***********")
logmodel.fit(x_train,y_train)
print("***********making the predictions....***********")
predictions = logmodel.predict(x_test)
print("***********printing classication report and confusion matrix....***********")
print(classification_report(y_test, predictions))
print(confusion_matrix(y_test, predictions))
```

```
************the modified log regression was used*************
***********making the model....***********
***********training the model....***********
***********making the predictions....***********
***********printing classication report and confusion matrix....***********
              precision    recall  f1-score   support

           0       0.82      0.73      0.77   2762496
           1       0.35      0.47      0.40    835972

    accuracy                           0.67   3598468
   macro avg       0.58      0.60      0.59   3598468
weighted avg       0.71      0.67      0.69   3598468

[[2014931  747565]
 [ 439370  396602]]
```

In [ ]:

The above was the over all model that we have made rather than a single machine learning model running for IDS. We felt that two models out of which one predicts the attack and the other does the prediction of the intrusion which help us to provide extra layer of security and we can even stop the intrusion before the attack is made by those predictions.

The processing time of algorithms that we had used

[11994893 rows almost 1.3 Gb data for predictions]

| | |
|---|---|
| Multiple linear regression<br>acc=94 | 32 sec |
| Naïve bayes algorithm<br>acc=92 | 30 sec |
| Decision tree algorithm<br>acc=1 | 34 sec |
| Ada-boost algorithm<br>acc=1 | 8-10 min |
| Random forest algorithm<br>acc=1 | 10-14 min |
| Logistic regression algorithm<br>acc=90 | 5-8 min |
| Hyper parameter tuned logistic regression<br>acc=1 | 1-2 hrs |

We had calculated the ROC-AUC curves for the prediction models that are attached below

After that the attack prediction model and then the label prediction model are attached

# ROC and AUC curves for the models developed above

In [1]:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn.metrics import classification_report
from sklearn import metrics
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import  (precision_score, recall_score, f1_score, accurac
y_score, mean_squared_error, mean_absolute_error)
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
```

In [2]:

```
data=pd.read_csv(r'C:\Users\asus\Downloads\NF-UQ-NIDS.csv')
data.describe()
```

Out[2]:

| | L4_SRC_PORT | L4_DST_PORT | PROTOCOL | L7_PROTO | IN_BYTES | OUT_BYTES |
|---|---|---|---|---|---|---|
| **count** | 1.199489e+07 | 1.199489e+07 | 1.199489e+07 | 1.199489e+07 | 1.199489e+07 | 1.199489e+07 |
| **mean** | 4.124519e+04 | 8.964758e+03 | 8.661742e+00 | 1.932050e+01 | 3.983727e+03 | 9.489449e+03 |
| **std** | 2.108654e+04 | 1.772082e+04 | 6.404263e+00 | 3.504857e+01 | 1.662256e+05 | 2.933544e+05 |
| **min** | 0.000000e+00 | 0.000000e+00 | 0.000000e+00 | 0.000000e+00 | 0.000000e+00 | 0.000000e+00 |
| **25%** | 3.299400e+04 | 5.300000e+01 | 6.000000e+00 | 0.000000e+00 | 6.800000e+01 | 0.000000e+00 |
| **50%** | 5.061000e+04 | 4.430000e+02 | 6.000000e+00 | 0.000000e+00 | 2.320000e+02 | 1.560000e+02 |
| **75%** | 5.581700e+04 | 3.389000e+03 | 6.000000e+00 | 7.000000e+00 | 1.440000e+03 | 1.873000e+03 |
| **max** | 6.553500e+04 | 6.553500e+04 | 2.550000e+02 | 2.510000e+02 | 2.282235e+08 | 2.432197e+08 |

In [3]:
```
data
```

Out[3]:

| | IPV4_SRC_ADDR | L4_SRC_PORT | IPV4_DST_ADDR | L4_DST_PORT | PROTOCOL | L7_P |
|---|---|---|---|---|---|---|
| **0** | 149.171.126.0 | 62073 | 59.166.0.5 | 56082 | 6 | |
| **1** | 149.171.126.2 | 32284 | 59.166.0.5 | 1526 | 6 | |
| **2** | 149.171.126.0 | 21 | 59.166.0.1 | 21971 | 6 | |
| **3** | 59.166.0.1 | 23800 | 149.171.126.0 | 46893 | 6 | |
| **4** | 59.166.0.5 | 63062 | 149.171.126.2 | 21 | 6 | |
| **...** | ... | ... | ... | ... | ... | |
| **11994888** | 192.168.100.46 | 80 | 192.168.100.5 | 80 | 6 | |
| **11994889** | 192.168.100.5 | 0 | 192.168.100.3 | 0 | 6 | |
| **11994890** | 192.168.100.7 | 365 | 192.168.100.3 | 565 | 17 | |
| **11994891** | 192.168.100.3 | 50850 | 13.54.166.67 | 8883 | 6 | 22 |
| **11994892** | 192.168.100.6 | 49160 | 192.168.100.149 | 4444 | 6 | |

11994893 rows × 15 columns

# data information about datatypes

In [4]:

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11994893 entries, 0 to 11994892
Data columns (total 15 columns):
IPV4_SRC_ADDR                 object
L4_SRC_PORT                   int64
IPV4_DST_ADDR                 object
L4_DST_PORT                   int64
PROTOCOL                      int64
L7_PROTO                      float64
IN_BYTES                      int64
OUT_BYTES                     int64
IN_PKTS                       int64
OUT_PKTS                      int64
TCP_FLAGS                     int64
FLOW_DURATION_MILLISECONDS    int64
Label                         int64
Attack                        object
Dataset                       object
dtypes: float64(1), int64(10), object(4)
memory usage: 1.3+ GB
```

# dropping unwanted columns

In [5]:

```
data=data.drop(['Dataset'],axis=1)
```

# categorical data into a seperate dataframe

In [6]:
```
cdata=data[['IPV4_SRC_ADDR','IPV4_DST_ADDR','Attack']]
cdata
```

Out[6]:

| | IPV4_SRC_ADDR | IPV4_DST_ADDR | Attack |
|---|---|---|---|
| **0** | 149.171.126.0 | 59.166.0.5 | Benign |
| **1** | 149.171.126.2 | 59.166.0.5 | Benign |
| **2** | 149.171.126.0 | 59.166.0.1 | Benign |
| **3** | 59.166.0.1 | 149.171.126.0 | Benign |
| **4** | 59.166.0.5 | 149.171.126.2 | Benign |
| **...** | ... | ... | ... |
| **11994888** | 192.168.100.46 | 192.168.100.5 | Benign |
| **11994889** | 192.168.100.5 | 192.168.100.3 | Benign |
| **11994890** | 192.168.100.7 | 192.168.100.3 | Benign |
| **11994891** | 192.168.100.3 | 13.54.166.67 | Benign |
| **11994892** | 192.168.100.6 | 192.168.100.149 | Theft |

11994893 rows × 3 columns

# label encoding the categorical data

In [7]:
```
from sklearn.preprocessing import LabelEncoder
cdata.columns
for label in cdata.columns:
    cdata[label]=LabelEncoder().fit(cdata[label]).transform(cdata[label])
```

In [8]:
```
data=data.drop(['IPV4_SRC_ADDR','IPV4_DST_ADDR','Attack'],axis=1)
data
```

Out[8]:
```
          L4_SRC_PORT  L4_DST_PORT  PROTOCOL  L7_PROTO  IN_BYTES  OUT_BYTES  IN_P
       0        62073        56082         6     0.000      9672        416
       1        32284         1526         6     0.000      1776        104
       2           21        21971         6     1.000      1842       1236
       3        23800        46893         6     0.000       528       8824
       4        63062           21         6     1.000      1786       2340
     ...          ...          ...       ...       ...       ...        ...
11994888           80           80         6     7.000   2330065          0     :
11994889            0            0         6     0.000   1054423          0
11994890          365          565        17     0.000     62422          0
11994891        50850         8883         6   222.178     11300       1664
11994892        49160         4444         6     0.000  40102320      37280

11994893 rows × 11 columns
```

In [9]:
```
fdata = pd.concat([data, cdata], axis=1)
fdata.head()
```

Out[9]:
```
   L4_SRC_PORT  L4_DST_PORT  PROTOCOL  L7_PROTO  IN_BYTES  OUT_BYTES  IN_PKTS  O
0        62073        56082         6       0.0      9672        416       11
1        32284         1526         6       0.0      1776        104        6
2           21        21971         6       1.0      1842       1236       26
3        23800        46893         6       0.0       528       8824       10
4        63062           21         6       1.0      1786       2340       32
```

In [10]:
```
fdata.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11994893 entries, 0 to 11994892
Data columns (total 14 columns):
L4_SRC_PORT                 int64
L4_DST_PORT                 int64
PROTOCOL                    int64
L7_PROTO                    float64
IN_BYTES                    int64
OUT_BYTES                   int64
IN_PKTS                     int64
OUT_PKTS                    int64
TCP_FLAGS                   int64
FLOW_DURATION_MILLISECONDS  int64
Label                       int64
IPV4_SRC_ADDR               int32
IPV4_DST_ADDR               int32
Attack                      int32
dtypes: float64(1), int32(3), int64(10)
memory usage: 1.1 GB
```

In [11]:
```
x=fdata.drop(['Label'],axis=1)
y=fdata['Label']
```

# train test split

In [12]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.30,random_state
=0)
x_train.shape,y_train.shape,x_test.shape,y_test.shape
```

Out[12]:
```
((8396425, 13), (8396425,), (3598468, 13), (3598468,))
```

# naive bayes algorithm

In [13]:

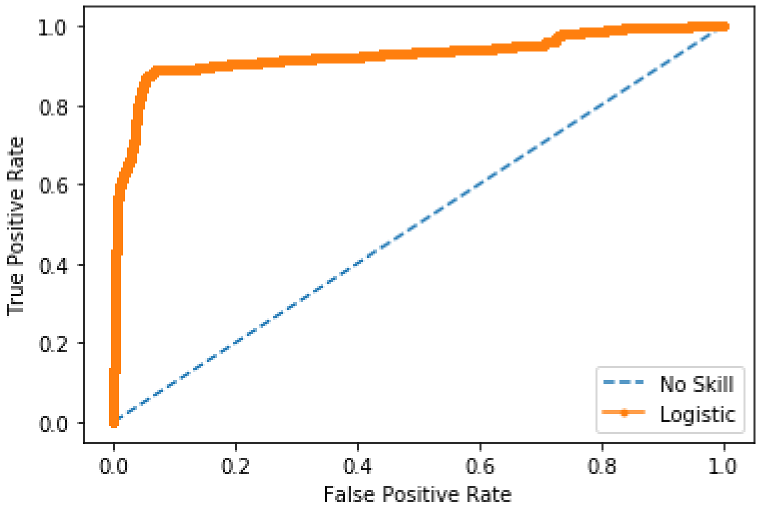
```
from sklearn.metrics import classification_report,confusion_matrix
from sklearn.metrics import accuracy_score, r2_score, mean_squared_error
from sklearn import metrics
print("**************the naive bayes algorithm used*****************")
gnb=GaussianNB()
print("***********training the model....***********")
gnb.fit(x_train,y_train)
print("***********making the predictions....***********")
y_pred=gnb.predict(x_test)
y_test.value_counts()
print("***********printing confusion matrix and classification report....*****
******")
cm=confusion_matrix(y_test,y_pred)

print(classification_report(y_test,y_pred))
print(confusion_matrix(y_test,y_pred))
```

```
**************the naive bayes algorithm used*****************
***********training the model....***********
***********making the predictions....***********
***********printing confusion matrix and classification report....***********
              precision    recall  f1-score   support

           0       0.94      0.96      0.95   2762496
           1       0.86      0.78      0.82    835972

    accuracy                           0.92   3598468
   macro avg       0.90      0.87      0.88   3598468
weighted avg       0.92      0.92      0.92   3598468

[[2658257  104239]
 [ 184495  651477]]
```

In [29]:

```
# roc curve and auc
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from matplotlib import pyplot


# generate a no skill prediction (majority class)
ns_probs = [0 for _ in range(len(y_test))]
print("predicting the probabilities")
# predict probabilities
lr_probs = gnb.predict_proba(x_test)
# keep probabilities for the positive outcome only
lr_probs = lr_probs[:, 1]
print("calculating scores")
# calculate scores
ns_auc = roc_auc_score(y_test, ns_probs)
lr_auc = roc_auc_score(y_test, lr_probs)
print("summerizing the scores")
# summarize scores
print('No Skill: ROC AUC=%.3f' % (ns_auc))
print('naive bayesan: ROC AUC=%.3f' % (lr_auc))
print("calculating roc curve")
# calculate roc curves
ns_fpr, ns_tpr, _ = roc_curve(y_test, ns_probs)
lr_fpr, lr_tpr, _ = roc_curve(y_test, lr_probs)
print("plotting roc curve")
# plot the roc curve for the model
pyplot.plot(ns_fpr, ns_tpr, linestyle='--', label='No Skill')
pyplot.plot(lr_fpr, lr_tpr, marker='.', label='Logistic')
# axis labels
pyplot.xlabel('False Positive Rate')
pyplot.ylabel('True Positive Rate')
# show the legend
pyplot.legend()
# show the plot
pyplot.show()
```

```
predicting the probabilities
calculating scores
summerizing the scores
No Skill: ROC AUC=0.500
naive bayesan: ROC AUC=0.927
calculating roc curve
plotting roc curve
```



# decision tree algorithm

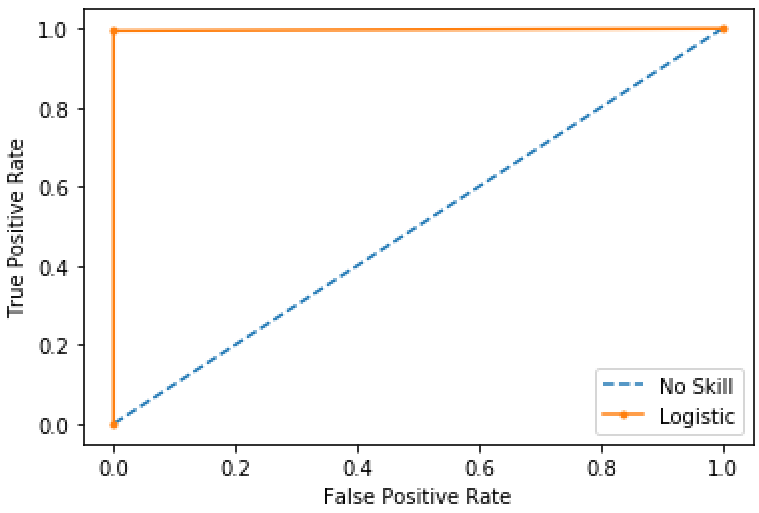
In [17]:
```python
from sklearn import tree
import matplotlib.pyplot as plt
print("**************the decision tree was used*****************")
clf=DecisionTreeClassifier(criterion='gini',splitter='random',max_leaf_nodes=10,min_samples_leaf=5,max_depth=5,random_state=0)
print("***********training the model....***********")
clf.fit(x_train,y_train)
print("***********making the predictions....***********")
y_pred=clf.predict(x_test)
y_pred
print("***********printing the classification report please wait....***********")
print(classification_report(y_test,y_pred))
```

```
**************the decision tree was used*****************
***********training the model....***********
***********making the predictions....***********
***********printing the classification report please wait....***********
              precision    recall  f1-score   support

           0       1.00      1.00      1.00   2762496
           1       1.00      0.99      1.00    835972

    accuracy                           1.00   3598468
   macro avg       1.00      1.00      1.00   3598468
weighted avg       1.00      1.00      1.00   3598468
```

In [28]:

```
# generate a no skill prediction (majority class)
ns_probs = [0 for _ in range(len(y_test))]

print("predicting the probabilities")
# predict probabilities
lr_probs = clf.predict_proba(x_test)
# keep probabilities for the positive outcome only
lr_probs = lr_probs[:, 1]
print("calculating scores")
# calculate scores
ns_auc = roc_auc_score(y_test, ns_probs)
lr_auc = roc_auc_score(y_test, lr_probs)
print("summerizing the scores")
# summarize scores
print('No Skill: ROC AUC=%.3f' % (ns_auc))
print('decision tree: ROC AUC=%.3f' % (lr_auc))
print("calculating roc curve")
# calculate roc curves
ns_fpr, ns_tpr, _ = roc_curve(y_test, ns_probs)
lr_fpr, lr_tpr, _ = roc_curve(y_test, lr_probs)
print("plotting roc curve")
# plot the roc curve for the model
pyplot.plot(ns_fpr, ns_tpr, linestyle='--', label='No Skill')
pyplot.plot(lr_fpr, lr_tpr, marker='.', label='Logistic')
# axis labels
pyplot.xlabel('False Positive Rate')
pyplot.ylabel('True Positive Rate')
# show the legend
pyplot.legend()
# show the plot
pyplot.show()
```

```
predicting the probabilities
calculating scores
summerizing the scores
No Skill: ROC AUC=0.500
decision tree: ROC AUC=0.997
calculating roc curve
plotting roc curve
```



# adaboost algorithm

In [19]:

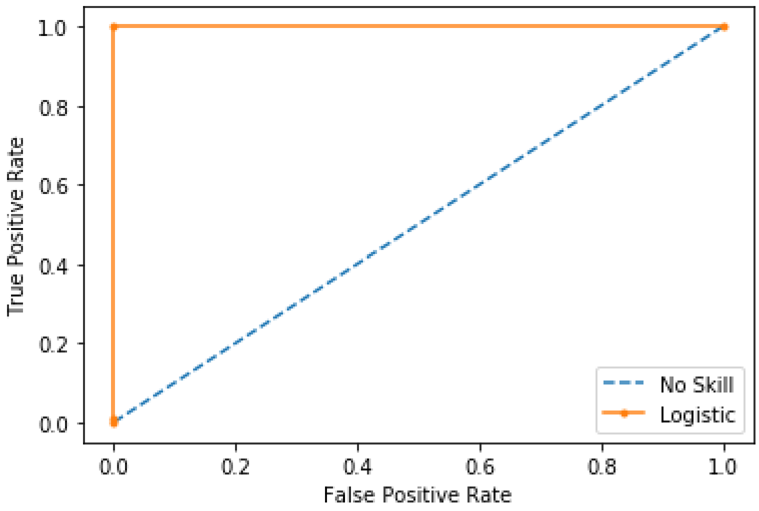
```
print("************the ensemble ADABOOST was used*************")
from sklearn.ensemble import AdaBoostClassifier
model=DecisionTreeClassifier(criterion='entropy',max_depth=1,random_state=0)
print("***********making the model....***********")
Adaboost=AdaBoostClassifier(base_estimator=model,n_estimators=20,random_state=
0)
print("***********training the model....***********")
boostmodel=Adaboost.fit(x_train,y_train)
print("***********making the predictions please wait....***********")
y_pred=boostmodel.predict(x_test)
print("***********printing the accuracy of the model please wait....**********
*")
predictions=metrics.accuracy_score(y_test,y_pred)
print("accuracy: ",predictions)
```

```
************the ensemble ADABOOST was used*************
***********making the model....***********
***********training the model....***********
***********making the predictions please wait....***********
***********printing the accuracy of the model please wait....***********
accuracy:  1.0
```

In [27]:

```
# generate a no skill prediction (majority class)
ns_probs = [0 for _ in range(len(y_test))]
# fit a model

print("predicting the probabilities")
# predict probabilities
lr_probs = boostmodel.predict_proba(x_test)
# keep probabilities for the positive outcome only
lr_probs = lr_probs[:, 1]
print("calculating scores")
# calculate scores
ns_auc = roc_auc_score(y_test, ns_probs)
lr_auc = roc_auc_score(y_test, lr_probs)
print("summerizing the scores")
# summarize scores
print('No Skill: ROC AUC=%.3f' % (ns_auc))
print('adaboost: ROC AUC=%.3f' % (lr_auc))
print("calculating roc curve")
# calculate roc curves
ns_fpr, ns_tpr, _ = roc_curve(y_test, ns_probs)
lr_fpr, lr_tpr, _ = roc_curve(y_test, lr_probs)
print("plotting roc curve")
# plot the roc curve for the model
pyplot.plot(ns_fpr, ns_tpr, linestyle='--', label='No Skill')
pyplot.plot(lr_fpr, lr_tpr, marker='.', label='Logistic')
# axis labels
pyplot.xlabel('False Positive Rate')
pyplot.ylabel('True Positive Rate')
# show the legend
pyplot.legend()
# show the plot
pyplot.show()
```

```
predicting the probabilities
calculating scores
summerizing the scores
No Skill: ROC AUC=0.500
adaboost: ROC AUC=1.000
calculating roc curve
plotting roc curve
```



In [21]:

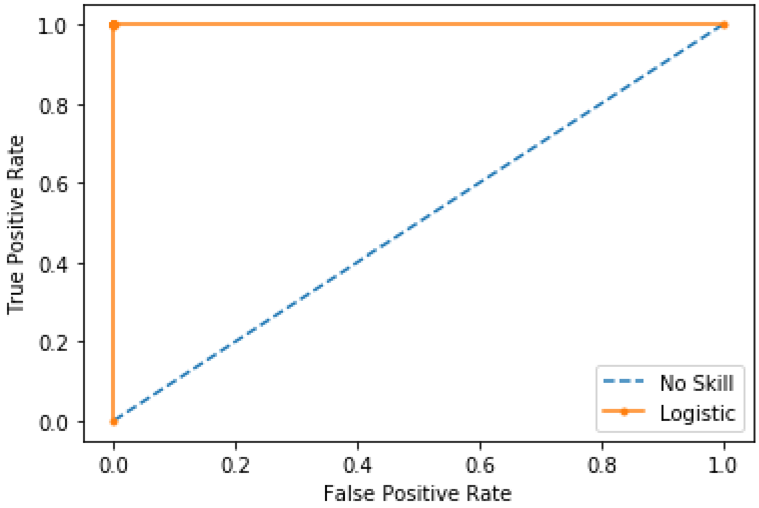
```
#random forest
print("************the random forest was used*************")
from sklearn.ensemble import RandomForestClassifier
print("***********making the model....***********")
classifier = RandomForestClassifier(n_estimators = 80, criterion = 'entropy',
random_state = 12)
print("***********training the model....***********")
classifier.fit(x_train, y_train)
# predict
print("***********making the predictions....***********")
y_pred = classifier.predict(x_test)
print("***********printing the accuracy....***********")
accuracy = accuracy_score(y_test, y_pred)
print("accuracy: ",accuracy)
```

```
************the random forest was used*************
***********making the model....***********
***********training the model....***********
***********making the predictions....***********
***********printing the accuracy....***********
accuracy:  1.0
```

In [26]:

```
# generate a no skill prediction (majority class)
ns_probs = [0 for _ in range(len(y_test))]
print("predicting the probabilities")
# predict probabilities
lr_probs = classifier.predict_proba(x_test)
# keep probabilities for the positive outcome only
lr_probs = lr_probs[:, 1]
print("calculating scores")
# calculate scores
ns_auc = roc_auc_score(y_test, ns_probs)
lr_auc = roc_auc_score(y_test, lr_probs)
print("summerizing the scores")
# summarize scores
print('No Skill: ROC AUC=%.3f' % (ns_auc))
print('random forest: ROC AUC=%.3f' % (lr_auc))
print("calculating roc curve")
# calculate roc curves
ns_fpr, ns_tpr, _ = roc_curve(y_test, ns_probs)
lr_fpr, lr_tpr, _ = roc_curve(y_test, lr_probs)
print("plotting roc curve")
# plot the roc curve for the model
pyplot.plot(ns_fpr, ns_tpr, linestyle='--', label='No Skill')
pyplot.plot(lr_fpr, lr_tpr, marker='.', label='Logistic')
# axis labels
pyplot.xlabel('False Positive Rate')
pyplot.ylabel('True Positive Rate')
# show the legend
pyplot.legend()
# show the plot
pyplot.show()
```

```
predicting the probabilities
calculating scores
summerizing the scores
No Skill: ROC AUC=0.500
random forest: ROC AUC=1.000
calculating roc curve
plotting roc curve
```



# logistic regression algorithm

In [24]:

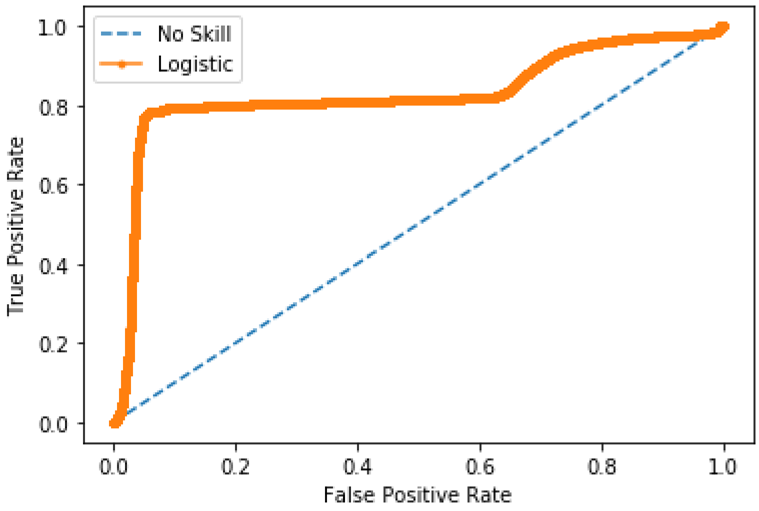
```
print("************the log regression was used*************")
from sklearn.linear_model import LogisticRegression
print("***********making the model....***********")
logmodel = LogisticRegression(max_iter=18000)
print("***********training the model....***********")
logmodel.fit(x_train,y_train)
print("***********making the predictions....***********")
predictions = logmodel.predict(x_test)
print("***********printing classication report and confusion matrix....***********")
print(classification_report(y_test, predictions))
print(confusion_matrix(y_test, predictions))
```

```
************the log regression was used*************
***********making the model....***********
***********training the model....***********
***********making the predictions....***********
***********printing classication report and confusion matrix....***********
              precision    recall  f1-score   support

           0       0.93      0.94      0.94   2762496
           1       0.80      0.78      0.79    835972

    accuracy                           0.90   3598468
   macro avg       0.87      0.86      0.86   3598468
weighted avg       0.90      0.90      0.90   3598468

[[2602019  160477]
 [ 183381  652591]]
```

In [25]:

```
# generate a no skill prediction (majority class)
ns_probs = [0 for _ in range(len(y_test))]
print("predicting the probabilities")
# predict probabilities
lr_probs = logmodel.predict_proba(x_test)
# keep probabilities for the positive outcome only
lr_probs = lr_probs[:, 1]
print("calculating scores")
# calculate scores
ns_auc = roc_auc_score(y_test, ns_probs)
lr_auc = roc_auc_score(y_test, lr_probs)
print("summerizing the scores")
# summarize scores
print('No Skill: ROC AUC=%.3f' % (ns_auc))
print('logistic: ROC AUC=%.3f' % (lr_auc))
print("calculating roc curve")
# calculate roc curves
ns_fpr, ns_tpr, _ = roc_curve(y_test, ns_probs)
lr_fpr, lr_tpr, _ = roc_curve(y_test, lr_probs)
print("plotting roc curve")
# plot the roc curve for the model
pyplot.plot(ns_fpr, ns_tpr, linestyle='--', label='No Skill')
pyplot.plot(lr_fpr, lr_tpr, marker='.', label='Logistic')
# axis labels
pyplot.xlabel('False Positive Rate')
pyplot.ylabel('True Positive Rate')
# show the legend
pyplot.legend()
# show the plot
pyplot.show()
```

```
predicting the probabilities
calculating scores
summerizing the scores
No Skill: ROC AUC=0.500
logistic: ROC AUC=0.831
calculating roc curve
plotting roc curve
```



# attack prediction model

In [1]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn.metrics import classification_report
from sklearn import metrics
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import  (precision_score, recall_score, f1_score, accurac
y_score, mean_squared_error, mean_absolute_error)
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
```

In [2]:
```
data=pd.read_csv(r'C:\Users\asus\Downloads\NF-UQ-NIDS.csv')
data.describe()
data=data.drop(['Dataset'],axis=1)
data=data.drop(['Label'],axis=1)
data
```

Out[2]:
```
          IPV4_SRC_ADDR  L4_SRC_PORT  IPV4_DST_ADDR  L4_DST_PORT  PROTOCOL  L7_PF
       0  149.171.126.0        62073     59.166.0.5        56082         6
       1  149.171.126.2        32284     59.166.0.5         1526         6
       2  149.171.126.0           21     59.166.0.1        21971         6
       3     59.166.0.1        23800  149.171.126.0        46893         6
       4     59.166.0.5        63062  149.171.126.2           21         6
     ...            ...          ...            ...          ...       ...
11994888 192.168.100.46           80  192.168.100.5           80         6
11994889  192.168.100.5            0  192.168.100.3            0         6
11994890  192.168.100.7          365  192.168.100.3          565        17
11994891  192.168.100.3        50850   13.54.166.67         8883         6    22.
11994892  192.168.100.6        49160 192.168.100.149         4444         6

11994893 rows × 13 columns
```

In [3]:
```
import warnings
warnings.filterwarnings('ignore')
numda=data[['Attack']]
numda.head()
```

Out[3]:

| | Attack |
|---|---|
| **0** | Benign |
| **1** | Benign |
| **2** | Benign |
| **3** | Benign |
| **4** | Benign |

In [4]:
```
cdata=data[['IPV4_SRC_ADDR','IPV4_DST_ADDR','Attack']]
cdata
```

Out[4]:

| | IPV4_SRC_ADDR | IPV4_DST_ADDR | Attack |
|---|---|---|---|
| **0** | 149.171.126.0 | 59.166.0.5 | Benign |
| **1** | 149.171.126.2 | 59.166.0.5 | Benign |
| **2** | 149.171.126.0 | 59.166.0.1 | Benign |
| **3** | 59.166.0.1 | 149.171.126.0 | Benign |
| **4** | 59.166.0.5 | 149.171.126.2 | Benign |
| **...** | ... | ... | ... |
| **11994888** | 192.168.100.46 | 192.168.100.5 | Benign |
| **11994889** | 192.168.100.5 | 192.168.100.3 | Benign |
| **11994890** | 192.168.100.7 | 192.168.100.3 | Benign |
| **11994891** | 192.168.100.3 | 13.54.166.67 | Benign |
| **11994892** | 192.168.100.6 | 192.168.100.149 | Theft |

11994893 rows × 3 columns

In [5]:
```
from sklearn.preprocessing import LabelEncoder
cdata.columns
for label in cdata.columns:
    cdata[label]=LabelEncoder().fit(cdata[label]).transform(cdata[label])
```

In [6]:

```
data=data.drop(['IPV4_SRC_ADDR','IPV4_DST_ADDR','Attack'],axis=1)
data
```

Out[6]:

| | L4_SRC_PORT | L4_DST_PORT | PROTOCOL | L7_PROTO | IN_BYTES | OUT_BYTES | IN_P |
|---|---|---|---|---|---|---|---|
| **0** | 62073 | 56082 | 6 | 0.000 | 9672 | 416 | |
| **1** | 32284 | 1526 | 6 | 0.000 | 1776 | 104 | |
| **2** | 21 | 21971 | 6 | 1.000 | 1842 | 1236 | |
| **3** | 23800 | 46893 | 6 | 0.000 | 528 | 8824 | |
| **4** | 63062 | 21 | 6 | 1.000 | 1786 | 2340 | |
| **...** | ... | ... | ... | ... | ... | ... | |
| **11994888** | 80 | 80 | 6 | 7.000 | 2330065 | 0 | |
| **11994889** | 0 | 0 | 6 | 0.000 | 1054423 | 0 | |
| **11994890** | 365 | 565 | 17 | 0.000 | 62422 | 0 | |
| **11994891** | 50850 | 8883 | 6 | 222.178 | 11300 | 1664 | |
| **11994892** | 49160 | 4444 | 6 | 0.000 | 40102320 | 37280 | |

11994893 rows × 10 columns

In [7]:

```
fdata = pd.concat([data, cdata], axis=1)
fdata.head()
```

Out[7]:

| | L4_SRC_PORT | L4_DST_PORT | PROTOCOL | L7_PROTO | IN_BYTES | OUT_BYTES | IN_PKTS | O |
|---|---|---|---|---|---|---|---|---|
| **0** | 62073 | 56082 | 6 | 0.0 | 9672 | 416 | 11 | |
| **1** | 32284 | 1526 | 6 | 0.0 | 1776 | 104 | 6 | |
| **2** | 21 | 21971 | 6 | 1.0 | 1842 | 1236 | 26 | |
| **3** | 23800 | 46893 | 6 | 0.0 | 528 | 8824 | 10 | |
| **4** | 63062 | 21 | 6 | 1.0 | 1786 | 2340 | 32 | |

In [8]:

```
numda['numdata']=fdata[['Attack']]
numda.head()
```

Out[8]:

| | Attack | numdata |
|---|---|---|
| **0** | Benign | 2 |
| **1** | Benign | 2 |
| **2** | Benign | 2 |
| **3** | Benign | 2 |
| **4** | Benign | 2 |

In [9]:

```
numda = numda.drop_duplicates('Attack')
print(numda)
```

```
                  Attack  numdata
0                 Benign        2
29              Exploits        7
67        Reconnaissance       11
93                   DoS        6
548              Generic        9
600            Shellcode       12
753             Backdoor        1
1858             Fuzzers        8
2663               Worms       14
35786           Analysis        0
1623130        injection       15
1623131             DDoS        5
1623136         scanning       19
1623264         password       17
1624507             mitm       16
1816782              xss       20
2403109       ransomware       18
3042976    Infilteration       10
4620416              Bot        3
4919431      Brute Force        4
11394794           Theft       13
```

In [10]:

```
x=fdata.drop(['Attack'],axis=1)
y=fdata['Attack']
```

In [11]:

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.30,random_state
=0)
x_train.shape,y_train.shape,x_test.shape,y_test.shape
```

Out[11]:

```
((8396425, 12), (8396425,), (3598468, 12), (3598468,))
```

In [12]:

```
print("************the ensemble ADABOOST was used*************")
from sklearn.ensemble import AdaBoostClassifier
model=DecisionTreeClassifier(criterion='entropy',max_depth=1,random_state=0)
print("***********making the model....***********")
Adaboost=AdaBoostClassifier(base_estimator=model,n_estimators=20,random_state=
0)
print("***********training the model....***********")
boostmodel=Adaboost.fit(x_train,y_train)
print("***********making the predictions please wait....***********")
y_pred=boostmodel.predict(x_test)
print("***********printing the accuracy of the model please wait....**********
*")
predictions=metrics.accuracy_score(y_test,y_pred)
print("accuracy: ",predictions)
```

```
************the ensemble ADABOOST was used*************
***********making the model....***********
***********training the model....***********
***********making the predictions please wait....***********
***********printing the accuracy of the model please wait....***********
accuracy:  0.6075938427130657
```

In [13]:

```
#random forest
print("************the random forest was used*************")
from sklearn.ensemble import RandomForestClassifier
print("***********making the model....***********")
classifier = RandomForestClassifier(n_estimators = 80, criterion = 'entropy',
random_state = 12)
print("***********training the model....***********")
classifier.fit(x_train, y_train)
# predict
print("***********making the predictions....***********")
y_pred = classifier.predict(x_test)
print("***********printing the accuracy....***********")
accuracy = accuracy_score(y_test, y_pred)
print("accuracy: ",accuracy)
```

```
************the random forest was used*************
***********making the model....***********
***********training the model....***********
***********making the predictions....***********
***********printing the accuracy....***********
accuracy:  0.9320055090110569
```

**label prediction model**

In [1]:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn.metrics import classification_report
from sklearn import metrics
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import  (precision_score, recall_score, f1_score, accurac
y_score, mean_squared_error, mean_absolute_error)
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
```

In [2]:

```
data=pd.read_csv(r'C:\Users\asus\Downloads\NF-UQ-NIDS.csv')
data.describe()
data=data.drop(['Dataset'],axis=1)
data
```

Out[2]:

| | IPV4_SRC_ADDR | L4_SRC_PORT | IPV4_DST_ADDR | L4_DST_PORT | PROTOCOL | L7_PF |
|---|---|---|---|---|---|---|
| **0** | 149.171.126.0 | 62073 | 59.166.0.5 | 56082 | 6 | |
| **1** | 149.171.126.2 | 32284 | 59.166.0.5 | 1526 | 6 | |
| **2** | 149.171.126.0 | 21 | 59.166.0.1 | 21971 | 6 | |
| **3** | 59.166.0.1 | 23800 | 149.171.126.0 | 46893 | 6 | |
| **4** | 59.166.0.5 | 63062 | 149.171.126.2 | 21 | 6 | |
| **...** | ... | ... | ... | ... | ... | |
| **11994888** | 192.168.100.46 | 80 | 192.168.100.5 | 80 | 6 | |
| **11994889** | 192.168.100.5 | 0 | 192.168.100.3 | 0 | 6 | |
| **11994890** | 192.168.100.7 | 365 | 192.168.100.3 | 565 | 17 | |
| **11994891** | 192.168.100.3 | 50850 | 13.54.166.67 | 8883 | 6 | 22 |
| **11994892** | 192.168.100.6 | 49160 | 192.168.100.149 | 4444 | 6 | |

11994893 rows × 14 columns

In [3]:

```
import warnings
warnings.filterwarnings('ignore')
data=data.drop(['Attack'],axis=1)
```

In [4]:
```
cdata=data[['IPV4_SRC_ADDR','IPV4_DST_ADDR']]
cdata
```

Out[4]:

| | IPV4_SRC_ADDR | IPV4_DST_ADDR |
|---|---|---|
| **0** | 149.171.126.0 | 59.166.0.5 |
| **1** | 149.171.126.2 | 59.166.0.5 |
| **2** | 149.171.126.0 | 59.166.0.1 |
| **3** | 59.166.0.1 | 149.171.126.0 |
| **4** | 59.166.0.5 | 149.171.126.2 |
| **...** | ... | ... |
| **11994888** | 192.168.100.46 | 192.168.100.5 |
| **11994889** | 192.168.100.5 | 192.168.100.3 |
| **11994890** | 192.168.100.7 | 192.168.100.3 |
| **11994891** | 192.168.100.3 | 13.54.166.67 |
| **11994892** | 192.168.100.6 | 192.168.100.149 |

11994893 rows × 2 columns

In [5]:
```
from sklearn.preprocessing import LabelEncoder
cdata.columns
for label in cdata.columns:
    cdata[label]=LabelEncoder().fit(cdata[label]).transform(cdata[label])
```

In [6]:
```
data=data.drop(['IPV4_SRC_ADDR','IPV4_DST_ADDR'],axis=1)
data
```

Out[6]:

| | L4_SRC_PORT | L4_DST_PORT | PROTOCOL | L7_PROTO | IN_BYTES | OUT_BYTES | IN_P |
|---|---|---|---|---|---|---|---|
| **0** | 62073 | 56082 | 6 | 0.000 | 9672 | 416 | |
| **1** | 32284 | 1526 | 6 | 0.000 | 1776 | 104 | |
| **2** | 21 | 21971 | 6 | 1.000 | 1842 | 1236 | |
| **3** | 23800 | 46893 | 6 | 0.000 | 528 | 8824 | |
| **4** | 63062 | 21 | 6 | 1.000 | 1786 | 2340 | |
| **...** | ... | ... | ... | ... | ... | ... | |
| **11994888** | 80 | 80 | 6 | 7.000 | 2330065 | 0 | : |
| **11994889** | 0 | 0 | 6 | 0.000 | 1054423 | 0 | |
| **11994890** | 365 | 565 | 17 | 0.000 | 62422 | 0 | |
| **11994891** | 50850 | 8883 | 6 | 222.178 | 11300 | 1664 | |
| **11994892** | 49160 | 4444 | 6 | 0.000 | 40102320 | 37280 | |

11994893 rows × 11 columns

In [7]:
```
fdata = pd.concat([data, cdata], axis=1)
fdata.head()
```

Out[7]:

|   | L4_SRC_PORT | L4_DST_PORT | PROTOCOL | L7_PROTO | IN_BYTES | OUT_BYTES | IN_PKTS | O |
|---|---|---|---|---|---|---|---|---|
| **0** | 62073 | 56082 | 6 | 0.0 | 9672 | 416 | 11 |  |
| **1** | 32284 | 1526 | 6 | 0.0 | 1776 | 104 | 6 |  |
| **2** | 21 | 21971 | 6 | 1.0 | 1842 | 1236 | 26 |  |
| **3** | 23800 | 46893 | 6 | 0.0 | 528 | 8824 | 10 |  |
| **4** | 63062 | 21 | 6 | 1.0 | 1786 | 2340 | 32 |  |

In [8]:
```
x=fdata.drop(['Label'],axis=1)
y=fdata['Label']
```

In [9]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.30,random_state
=0)
x_train.shape,y_train.shape,x_test.shape,y_test.shape
```

Out[9]:
```
((8396425, 12), (8396425,), (3598468, 12), (3598468,))
```

In [10]:

```
#multiple linear regression
print("******************** multiple linear regression************************
*******")
from sklearn.linear_model import  LinearRegression
from sklearn.metrics import classification_report,confusion_matrix
from sklearn.metrics import accuracy_score, r2_score, mean_squared_error
print("***********making the model....***********")
regressor = LinearRegression()
print("***********training the model....***********")
regressor=regressor.fit(x_train, y_train)
print("***********making prediction please wait....***********")
y_pred = regressor.predict(x_test)
y_pred
print("***********printing confusion matrix and classification report....*****
******")
print(confusion_matrix(y_test, y_pred.round()))
print(classification_report(y_test, y_pred.round()))
```

```
******************** multiple linear regression******************************
*
***********making the model....***********
***********training the model....***********
***********making prediction please wait....***********
***********printing confusion matrix and classification report....***********
[[      0       0       0       0       0       0       0       0]
 [      0       0       0       0       0       0       0       0]
 [      0       0       0       0       0       0       0       0]
 [      0       0       0       0       0       0       0       0]
 [      0       0       4   12794 2584447  165246       4       1]
 [      1       1     948    1568  234021  599433       0       0]
 [      0       0       0       0       0       0       0       0]
 [      0       0       0       0       0       0       0       0]]
              precision    recall  f1-score   support

        -5.0       0.00      0.00      0.00         0
        -4.0       0.00      0.00      0.00         0
        -2.0       0.00      0.00      0.00         0
        -1.0       0.00      0.00      0.00         0
         0.0       0.92      0.94      0.93   2762496
         1.0       0.78      0.72      0.75    835972
         2.0       0.00      0.00      0.00         0
         3.0       0.00      0.00      0.00         0

    accuracy                           0.88   3598468
   macro avg       0.21      0.21      0.21   3598468
weighted avg       0.89      0.88      0.89   3598468
```

In [11]:

```
from sklearn import metrics
print("**************the naive bayes algorithm used*****************")
gnb=GaussianNB()
print("***********training the model....***********")
gnb.fit(x_train,y_train)
print("***********making the predictions....***********")
y_pred=gnb.predict(x_test)
y_test.value_counts()
print("***********printing confusion matrix and classification report....*****
******")
cm=confusion_matrix(y_test,y_pred)

print(classification_report(y_test,y_pred))
print(confusion_matrix(y_test,y_pred))
```

```
**************the naive bayes algorithm used*****************
***********training the model....***********
***********making the predictions....***********
***********printing confusion matrix and classification report....***********
              precision    recall  f1-score   support

           0       0.93      0.96      0.95   2762496
           1       0.86      0.78      0.82    835972

    accuracy                           0.92   3598468
   macro avg       0.90      0.87      0.88   3598468
weighted avg       0.92      0.92      0.92   3598468

[[2657889  104607]
 [ 184952  651020]]
```

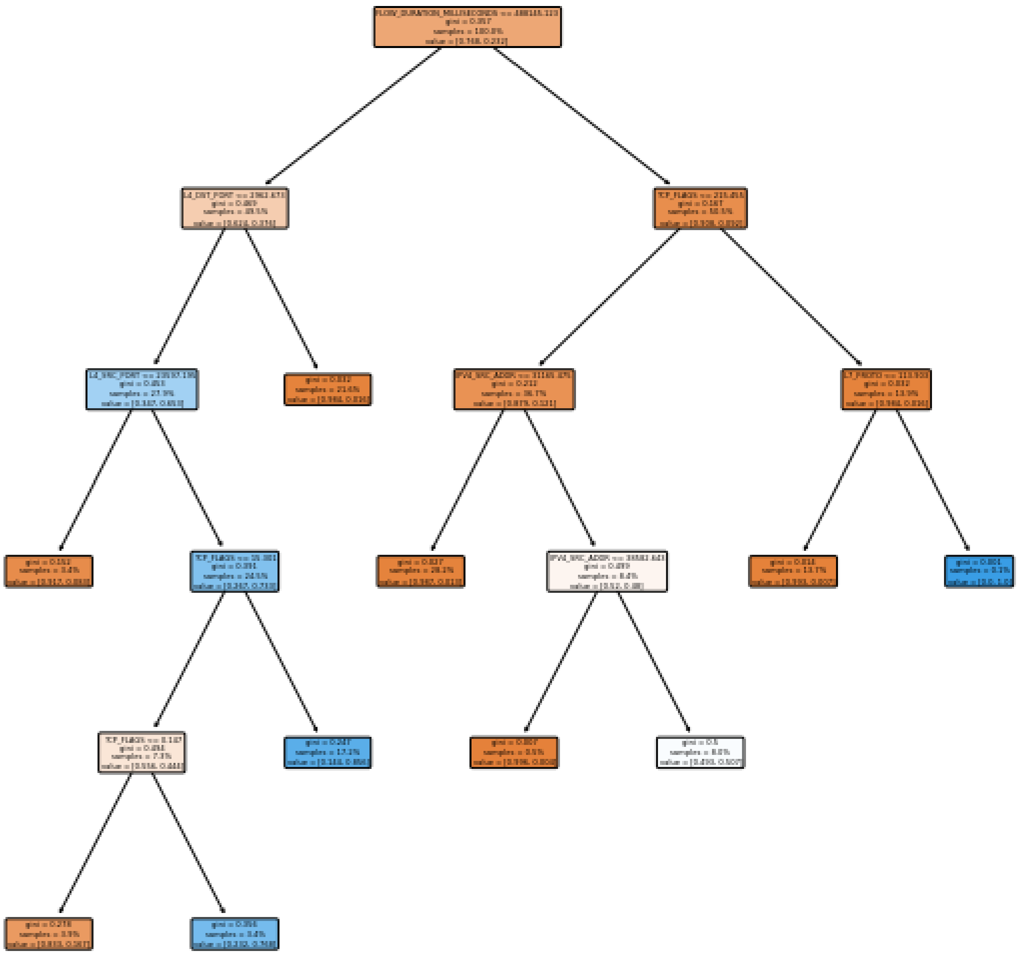
In [12]:

```
from sklearn import tree
import matplotlib.pyplot as plt
print("**************the decision tree was used*****************")
clf=DecisionTreeClassifier(criterion='gini',splitter='random',max_leaf_nodes=1
0,min_samples_leaf=5,max_depth=5,random_state=0)
print("***********training the model....***********")
clf.fit(x_train,y_train)
print("***********making the predictions....***********")
y_pred=clf.predict(x_test)
y_pred
print("***********printing the classification report please wait....**********
*")
print(classification_report(y_test,y_pred))
```

```
**************the decision tree was used*****************
***********training the model....***********
***********making the predictions....***********
***********printing the classification report please wait....***********
              precision    recall  f1-score   support

           0       0.98      0.96      0.97   2762496
           1       0.88      0.92      0.90    835972

    accuracy                           0.95   3598468
   macro avg       0.93      0.94      0.93   3598468
weighted avg       0.95      0.95      0.95   3598468
```

In [13]:
```
print("***********making the decision tree....***********")
cols=list(x.columns.values)
plt.figure(figsize=(10,10))
tree.plot_tree(clf.fit(x,y),feature_names=cols,filled=True,precision=3,proport
ion=True,rounded=True)
plt.show()
```

```
***********making the decision tree....***********
```



In [14]:

```
print("************the ensemble ADABOOST was used*************")
from sklearn.ensemble import AdaBoostClassifier
model=DecisionTreeClassifier(criterion='entropy',max_depth=1,random_state=0)
print("***********making the model....***********")
Adaboost=AdaBoostClassifier(base_estimator=model,n_estimators=20,random_state=
0)
print("***********training the model....***********")
boostmodel=Adaboost.fit(x_train,y_train)
print("***********making the predictions please wait....***********")
y_pred=boostmodel.predict(x_test)
print("***********printing the accuracy of the model please wait....**********
*")
predictions=metrics.accuracy_score(y_test,y_pred)
print("accuracy: ",predictions)
```

```
************the ensemble ADABOOST was used*************
***********making the model....***********
***********training the model....***********
***********making the predictions please wait....***********
***********printing the accuracy of the model please wait....***********
accuracy:  0.9831884012863252
```

In [15]:

```
#random forest
print("************the random forest was used*************")
from sklearn.ensemble import RandomForestClassifier
print("***********making the model....***********")
classifier = RandomForestClassifier(n_estimators = 80, criterion = 'entropy',
random_state = 12)
print("***********training the model....***********")
classifier.fit(x_train, y_train)
# predict
print("***********making the predictions....***********")
y_pred = classifier.predict(x_test)
print("***********printing the accuracy....***********")
accuracy = accuracy_score(y_test, y_pred)
print("accuracy: ",accuracy)
```

```
************the random forest was used*************
***********making the model....***********
***********training the model....***********
***********making the predictions....***********
***********printing the accuracy....***********
accuracy:  0.9948883802773847
```

In [16]:

```
print("************the log regression was used*************")
from sklearn.linear_model import LogisticRegression
print("***********making the model....***********")
logmodel = LogisticRegression(max_iter=18000)
print("***********training the model....***********")
logmodel.fit(x_train,y_train)
print("***********making the predictions....***********")
predictions = logmodel.predict(x_test)
print("***********printing classication report and confusion matrix....*******
****")
print(classification_report(y_test, predictions))
print(confusion_matrix(y_test, predictions))
```

```
************the log regression was used*************
***********making the model....***********
***********training the model....***********
***********making the predictions....***********
***********printing classication report and confusion matrix....***********
              precision    recall  f1-score   support

           0       0.93      0.94      0.94   2762496
           1       0.80      0.78      0.79    835972

    accuracy                           0.90   3598468
   macro avg       0.87      0.86      0.86   3598468
weighted avg       0.90      0.90      0.90   3598468

[[2602005  160491]
 [ 183376  652596]]
```

In [ ]:

```
from sklearn.model_selection import GridSearchCV
logistic = LogisticRegression(max_iter=30000)
penalty = [ 'l2']
C = np.logspace(0, 4, 10)
hyperparameters = dict(C=C, penalty=penalty)
clf = GridSearchCV(logistic, hyperparameters, cv=5, verbose=0)
best_model = clf.fit(x_train, y_train)
print('Best Penalty:', best_model.best_estimator_.get_params()['penalty'])
print('Best C:', best_model.best_estimator_.get_params()['C'])
```

```
      ************the log regression was used*************
      ***********making the model....***********
       ***********training the model....***********
   ***********making the predictions....***********
      ***********printing classication report and confusion matrix....***********
          precision    recall  f1-score   support

           0       1.00      1.00      1.00   2762496
           1       1.00      1.00      1.00    835972

    accuracy                           1.00   3598468
   macro avg       1.00      1.00      1.00   3598468
weighted avg       1.00      1.00      1.00   3598468

    [[2602005  160491]
    [ 183376  652596]]
```

```
In [ ]:
```