**import** pandas **as** pd

**import** seaborn **as** sns

**import** numpy **as** np

**from** sklearn.preprocessing **import** LabelEncoder

**from** sklearn.model\_selection **import** KFold, cross\_val\_score

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

**from** imblearn.under\_sampling **import** RandomUnderSampler

**from** sklearn.metrics **import** RocCurveDisplay, confusion\_matrix, roc\_auc\_score, classificatio

**from** sklearn.linear\_model **import** LogisticRegression **from** sklearn.neighbors **import** KNeighborsClassifier **from** sklearn.ensemble **import** RandomForestClassifier **from** sklearn.ensemble **import** AdaBoostClassifier

In [27]:

date\_columns **=** ["Timestamp"]

df **=** pd.read\_csv("ddos\_dataset.csv", parse\_dates**=**date\_columns,index\_col**=None**) df.head()

C:\Users\ADMIN\AppData\Local\Temp\ipykernel\_7616\3645980268.py:2: UserWarnin g: Could not infer format, so each element will be parsed individually, fall ing back to `dateutil`. To ensure parsing is consistent and as-expected, ple ase specify a format.

df = pd.read\_csv("ddos\_dataset.csv", parse\_dates=date\_columns,index\_col=No ne)

Out[27]:

**Timestamp Fwd Pkt Len Mean**

|  |  |
| --- | --- |
|  | **Flow ID** |
| 172.31.69.28- |
| **0** | 18.216.200.189-80- |
|  | 52169-6 |
|  | 172.31.69.25- |
| **1** | 18.219.193.20-80- |
|  | 44588-6 |
|  | 172.31.69.25- |
| **2** | 18.219.193.20-80- |
|  | 43832-6 |
|  | 172.31.69.25- |
| **3** | 18.219.193.20-80- |
|  | 53346-6 |
|  | 172.31.69.28- |
| **4** | 18.218.55.126-80- |
|  | 57856-6 |

2018-02-22

**Fwd Seg Size Avg**

**Init Fwd Win Byts**

**Init Bwd Win Byts**

**Fwd Seg Size Min**

**Label**

00:27:57 233.750000 233.750000 -1 32768 0 ddos

2018-02-16

23:18:14 0.000000 0.000000 -1 225 0 ddos

2018-02-16

23:23:20 114.333333 114.333333 -1 219 0 ddos

2018-02-16

23:22:41 233.750000 233.750000 -1 211 0 ddos

2018-02-21

23:49:25 233.750000 233.750000 -1 32768 0 ddos

df.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 500000 entries, 0 to 499999 Data columns (total 8 columns):

# Column Non-Null Count Dtype

1. Flow ID 500000 non-null object
2. Timestamp 500000 non-null datetime64[ns]
3. Fwd Pkt Len Mean 500000 non-null float64
4. Fwd Seg Size Avg 500000 non-null float64
5. Init Fwd Win Byts 500000 non-null int64
6. Init Bwd Win Byts 500000 non-null int64
7. Fwd Seg Size Min 500000 non-null int64
8. Label 500000 non-null object

dtypes: datetime64[ns](1), float64(2), int64(3), object(2) memory usage: 30.5+ MB

In [29]:

df[['Source', 'Destination', 'Source Port', 'Dest Port', 'Other']] **=** df['Flow ID'].str.spli df.head()

Out[29]:

**Flow ID Timestamp Fwd Pkt**

**Len Mean**

**Fwd Seg Size Avg**

**Init Fwd Win Byts**

**Init Bwd Win Byts**

**Fwd Seg Size Min**

**Label Source**

172.31.69.28-

**0** 18.216.200.189-

80-52169-6

172.31.69.25-

**1** 18.219.193.20-

80-44588-6

172.31.69.25-

**2** 18.219.193.20-

80-43832-6

172.31.69.25-

**3** 18.219.193.20-

80-53346-6

172.31.69.28-

**4** 18.218.55.126-

80-57856-6

2018-02-22 233.750000 233.750000 -1 32768 0 ddos 172.31.69.28

00:27:57

2018-02-16 0.000000 0.000000 -1 225 0 ddos 172.31.69.25

23:18:14

2018-02-16 114.333333 114.333333 -1 219 0 ddos 172.31.69.25

23:23:20

2018-02-16 233.750000 233.750000 -1 211 0 ddos 172.31.69.25

23:22:41

2018-02-21 233.750000 233.750000 -1 32768 0 ddos 172.31.69.28

23:49:25

In [30]:

df **=** df.sort\_values("Timestamp")

*# Dropping Timestamp and ports (not sure about the data for ports hence deleting for safety*

df **=** df.drop(columns**=**["Timestamp", "Source Port", "Dest Port", "Other"]) df.head()

Out[31]:

**Init**

**Init**

**Fwd**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Flow ID** | **Fwd Pkt Len Mean** | **Fwd Seg Size Avg** | **Fwd Win Byts** | **Bwd Win Byts** | **Seg Size Min** | **Label** | **Source** | **Destinati** |
|  | 192.168.1.104- |  |  |  |  |  |  |  |  |
| **13318** | 203.73.24.75- | 0.000000 | 0.000000 | -1 | 17520 | 0 | ddos | 192.168.1.104 | 203.73.24. |
|  | 19754-80-6 |  |  |  |  |  |  |  |  |
|  | 192.168.1.104- |  |  |  |  |  |  |  |  |
| **21539** | 203.73.24.75- | 45.500000 | 45.500000 | -1 | 5840 | 0 | ddos | 192.168.1.104 | 203.73.24. |
|  | 19817-80-6 |  |  |  |  |  |  |  |  |
|  | 192.168.1.104- |  |  |  |  |  |  |  |  |
| **87259** | 203.73.24.75- | 0.000000 | 0.000000 | -1 | 17520 | 0 | ddos | 192.168.1.104 | 203.73.24. |
|  | 19824-80-6 |  |  |  |  |  |  |  |  |
|  | 192.168.1.104- |  |  |  |  |  |  |  |  |
| **61071** | 203.73.24.75- | 0.000000 | 0.000000 | -1 | 17520 | 0 | ddos | 192.168.1.104 | 203.73.24. |
|  | 19830-80-6 |  |  |  |  |  |  |  |  |
|  | 192.168.1.104- |  |  |  |  |  |  |  |  |
| **5143** | 203.73.24.75- | 39.333333 | 39.333333 | -1 | 5840 | 0 | ddos | 192.168.1.104 | 203.73.24. |
|  | 19881-80-6 |  |  |  |  |  |  |  |  |

In [32]:

df[['SourceIP\_1', 'SourceIP\_2', 'SourceIP\_3', 'SourceIP\_4']] **=** df.Source.str.split('.', exp df[['DestinationIP\_1', 'DestinationIP\_2', 'DestinationIP\_3', 'DestinationIP\_4']] **=** df.Desti df **=** df.drop(columns**=**["Source", "Destination", "Flow ID"])

df.head()

Out[32]:

**Label SourceIP\_1 SourceIP\_2 SourceIP\_3 S**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | | | **Init** | **Init** | **Fwd** |
|  | **Fwd Pkt** | **Fwd Seg** | **Fwd** | **Bwd** | **Seg** |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Len Mean** | **Size Avg** | **Win Byts** | **Win Byts** | **Size Min** |  | |
| **13318** | 0.000000 | 0.000000 | -1 | 17520 | 0 | ddos 192 168 | 1 |
| **21539** | 45.500000 | 45.500000 | -1 | 5840 | 0 | ddos 192 168 | 1 |
| **87259** | 0.000000 | 0.000000 | -1 | 17520 | 0 | ddos 192 168 | 1 |
| **61071** | 0.000000 | 0.000000 | -1 | 17520 | 0 | ddos 192 168 | 1 |
| **5143** | 39.333333 | 39.333333 | -1 | 5840 | 0 | ddos 192 168 | 1 |

df.isna().sum()

Out[33]:

Fwd Pkt Len Mean 0

Fwd Seg Size Avg 0

Init Fwd Win Byts 0

Init Bwd Win Byts 0

Fwd Seg Size Min 0

Label 0

SourceIP\_1 0

SourceIP\_2 0

SourceIP\_3 0

SourceIP\_4 0

DestinationIP\_1 0

DestinationIP\_2 0

DestinationIP\_3 0

DestinationIP\_4 0

dtype: int64

In [34]:

le **=** LabelEncoder()

df['Label'] **=** le.fit\_transform(df['Label']) df.head()

Out[34]:

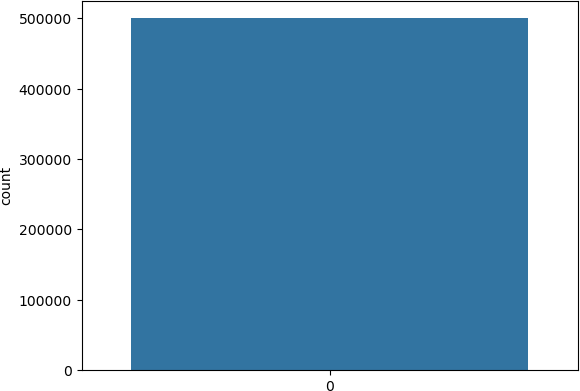
**Init**

**Init**

**Fwd**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Fwd Pkt Len Mean** | **Fwd Seg Size Avg** | **Fwd Win Byts** | **Bwd Win Byts** | **Seg Size Min** | **Label** | **SourceIP\_1** | **SourceIP\_2** | **SourceIP\_3 S** |
| **13318** | 0.000000 | 0.000000 | -1 | 17520 | 0 | 1 | 192 | 168 | 1 |
| **21539** | 45.500000 | 45.500000 | -1 | 5840 | 0 | 1 | 192 | 168 | 1 |
| **87259** | 0.000000 | 0.000000 | -1 | 17520 | 0 | 1 | 192 | 168 | 1 |
| **61071** | 0.000000 | 0.000000 | -1 | 17520 | 0 | 1 | 192 | 168 | 1 |
| **5143** | 39.333333 | 39.333333 | -1 | 5840 | 0 | 1 | 192 | 168 | 1 |

sns.countplot(df['Label']);

****

In [36]:

X **=** df.drop('Label' , axis **=** 1) y **=** df['Label']

RUS **=** RandomUnderSampler(random\_state**=**42) X\_rus, y\_rus **=** RUS.fit\_resample(X,y)

In [37]:

X\_train, X\_test, y\_train, y\_test **=** train\_test\_split(X\_rus, y\_rus, test\_size**=**0.3, random\_sta

models **=** {"Logistic Regression": LogisticRegression(), "Random Forest": RandomForestClassif "KNN": KNeighborsClassifier(), "AdaBoost": AdaBoostClassifier()}

cv **=** KFold(n\_splits**=**10)

**import** time

**def** fit\_and\_score(models, X\_train, X\_test, y\_train, y\_test): model\_scores **=** {}

model\_roc\_auc\_scores **=** {} model\_time **=** {}

**for** name, model **in** models.items(): start **=** time.process\_time()

model.fit(X\_train, y\_train)

model\_time[name] **=** time.process\_time() **-** start

scores **=** cross\_val\_score(model, X\_train, y\_train, scoring**=**'roc\_auc', cv**=**cv, n\_jobs**=** model\_roc\_auc\_scores[name] **=** roc\_auc\_score(y\_test, model.predict\_proba(X\_test)[:,1] model\_scores[name] **=** model.score(X\_test, y\_test)

**return** model\_scores, model\_roc\_auc\_scores, model\_time

In [39]:

model\_scores, model\_roc\_auc\_scores, model\_time **=** fit\_and\_score(models, X\_train, X\_test, y\_t print("ACCURACIES : ")

model\_scores

C:\Users\ADMIN\anaconda3\Lib\site-packages\sklearn\linear\_model\\_logistic.p y:458: ConvergenceWarning: lbfgs failed to converge (status=1):

STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max\_iter) or scale the data as shown in: [https://scikit-learn.org/stable/modules/preprocessing.html (https://scik](https://scikit-learn.org/stable/modules/preprocessing.html)

[it-learn.org/stable/modules/preprocessing.html)](https://scikit-learn.org/stable/modules/preprocessing.html)

Please also refer to the documentation for alternative solver options:

[https://scikit-learn.org/stable/modules/linear\_model.html#logistic-regre ssion (https://scikit-learn.org/stable/modules/linear\_model.html#logistic-re gression)](https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)

n\_iter\_i = \_check\_optimize\_result( ACCURACIES :

Out[39]:

{'Logistic Regression': 0.95815,

'Random Forest': 0.9999666666666667,

'KNN': 0.9987833333333334,

'AdaBoost': 0.9997}

print("ROC AUC SCORES : ")

model\_roc\_auc\_scores

ROC AUC SCORES :

Out[40]:

{'Logistic Regression': 0.990470871604245,

'Random Forest': 0.9999999299982733,

'KNN': 0.9996129115624995,

'AdaBoost': 0.9999994033186146}

In [41]:

print("Time : ") model\_time

Time :

Out[41]:

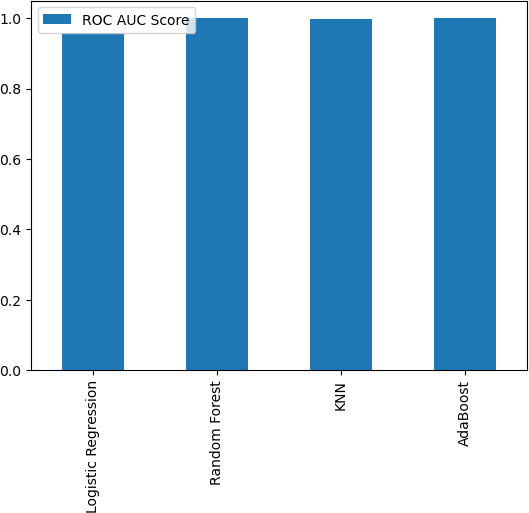
{'Logistic Regression': 3.859375,

'Random Forest': 4.078125,

'KNN': 0.421875,

'AdaBoost': 19.125}

model\_compare **=** pd.DataFrame(model\_scores, index**=**['ROC AUC Score']) model\_compare.T.plot.bar();



In [43]:

clf **=** RandomForestClassifier() clf.fit(X\_train, y\_train)

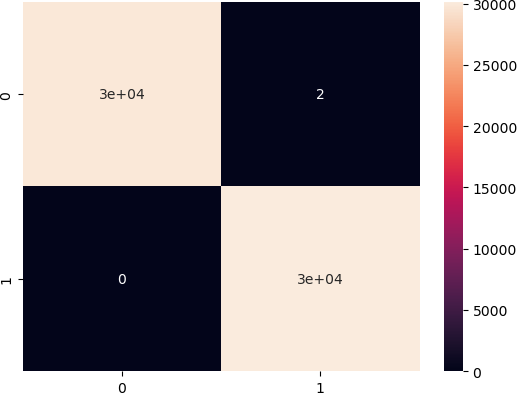
clf.score(X\_test, y\_test)

Out[43]:

0.9999666666666667

y\_preds **=** clf.predict(X\_test)

conf\_mat **=** confusion\_matrix(y\_test, y\_preds) sns.heatmap(conf\_mat, annot**=True**);



In [48]:

print(classification\_report(y\_test, y\_preds))

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | precision | recall | f1-score | support |
| 0 | 1.00 | 1.00 | 1.00 | 29851 |
| 1 | 1.00 | 1.00 | 1.00 | 30149 |
| accuracy |  |  | 1.00 | 60000 |
| macro avg | 1.00 | 1.00 | 1.00 | 60000 |
| weighted avg | 1.00 | 1.00 | 1.00 | 60000 |

average\_precision **=** average\_precision\_score(y\_test, clf.predict\_proba(X\_test)[:,1]) print('Average precision-recall score: {0:0.2f}'.format(average\_precision))

Average precision-recall score: 1.00

In [ ]: