import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from mpl\_toolkits.mplot3d import Axes3D

from sklearn.preprocessing import StandardScaler

import seaborn as sns

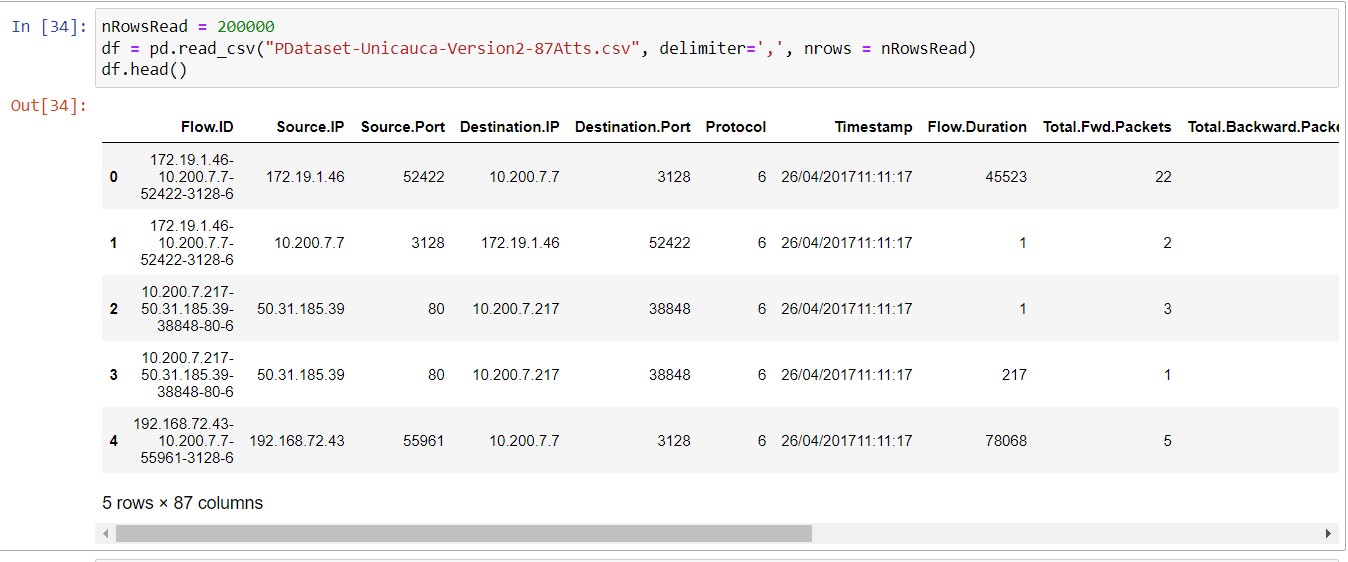
from collections import Counter

nRowsRead = 200000

df = pd.read\_csv("PDataset-Unicauca-Version2-87Atts.csv", delimiter=',', nrows = nRowsRead)

df.head()

**OUTPUT**



print(sorted(Counter(df['ProtocolName']).items()))

**output**

****

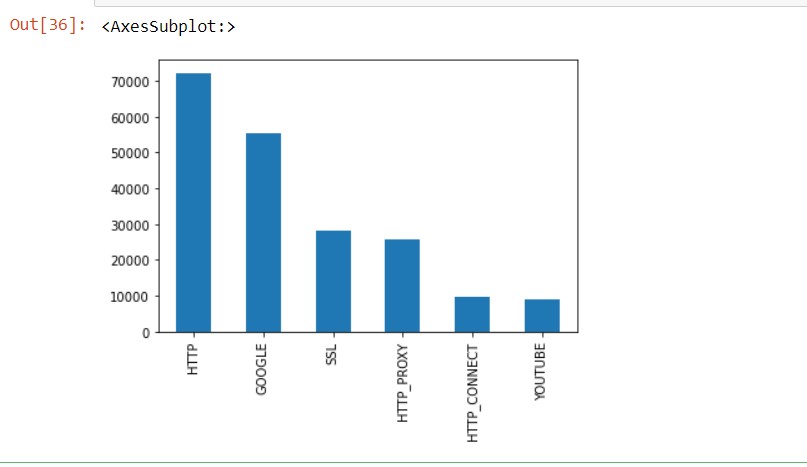
top =df['ProtocolName'].value\_counts()

top.sort\_values(ascending = False)

top = top[ : 6]

top.plot(kind = "bar")

**OUTPUT**

****

**UNDER SAMPLING**

X = df

Y = df['ProtocolName']

from imblearn.under\_sampling import RandomUnderSampler

rus = RandomUnderSampler(random\_state = 0)

X\_resampled,y\_resampled = rus.fit\_resample(X, Y)

print(sorted(Counter(y\_resampled).items()), y\_resampled.shape)

**OUTPUT**

****

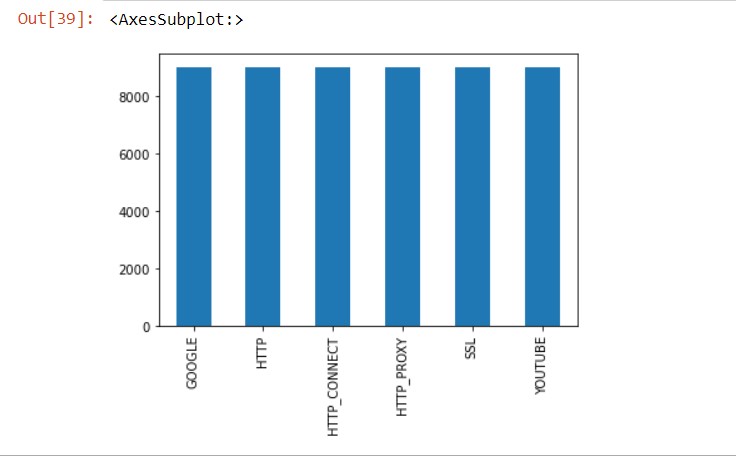
top = y\_resampled.value\_counts()

top.sort\_values(ascending = False)

top = top[ : 6]

top.plot(kind = "bar")

**OUTPUT**

****

df.shape

**OUTPUT**

****

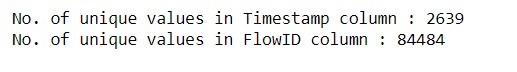
**DATA PREPROCESSING**

ipdata = df.copy()

print("No. of unique values in Timestamp column :",ipdata['Timestamp'].nunique())

print("No. of unique values in FlowID column :",ipdata['Flow.ID'].nunique())

**OUTPUT**

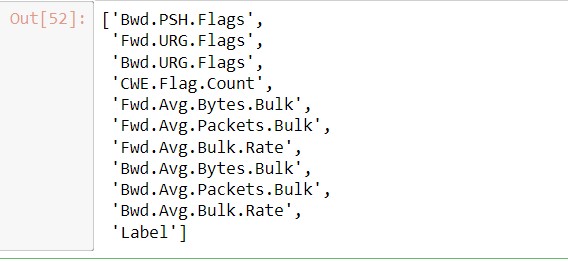


ipdata.drop(['Timestamp', 'Flow.ID'], axis = 1, inplace = True)

single\_unique\_cols = [col for col in ipdata.columns if ipdata[col].nunique() == 1]

single\_unique\_cols

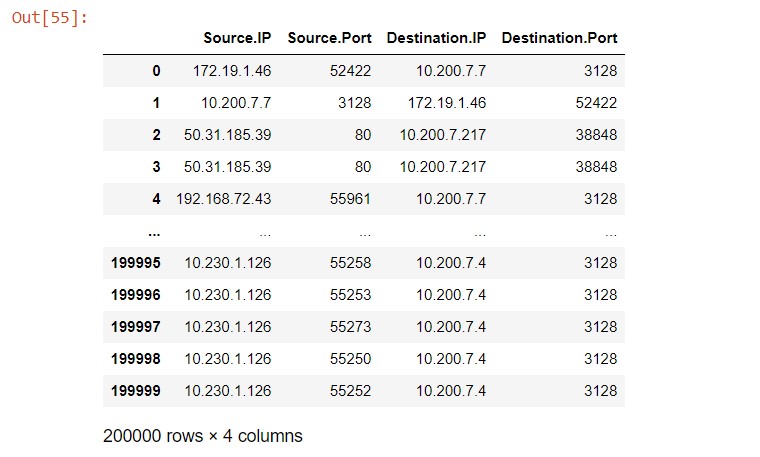
**OUTPUT**



ipdata.drop(single\_unique\_cols, axis = 1, inplace = True)

ip\_add\_cols = ['Source.IP', 'Source.Port', 'Destination.IP', 'Destination.Port']

ipdata[ip\_add\_cols]



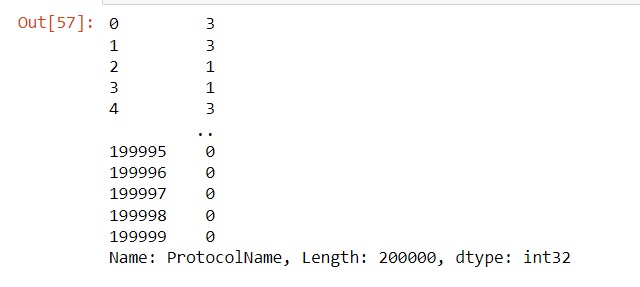
ipdata.drop(ip\_add\_cols, axis = 1, inplace = True)

from sklearn.preprocessing import LabelEncoder

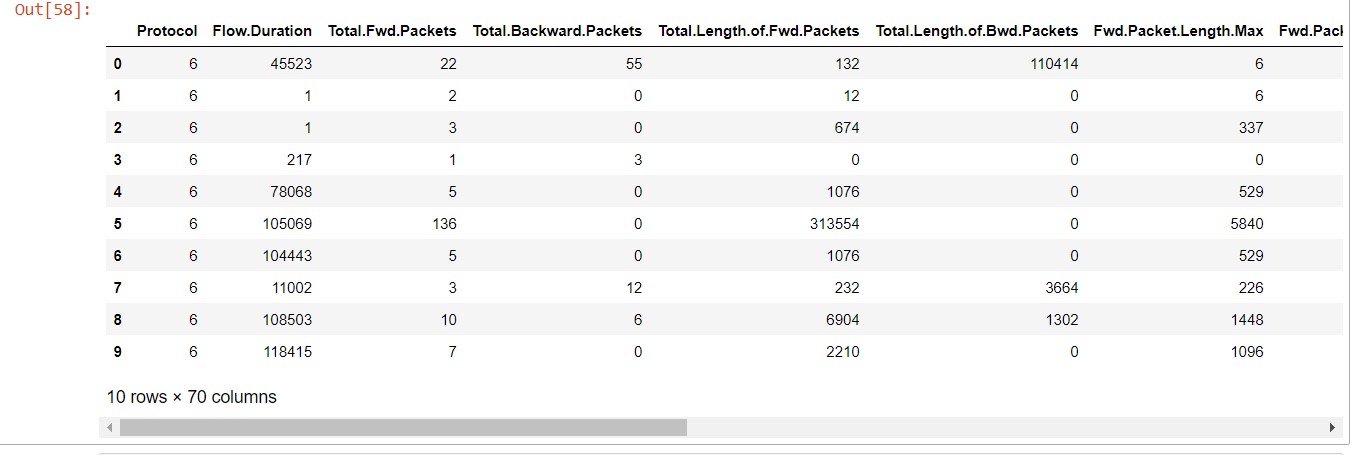
encoder = LabelEncoder().fit(ipdata['ProtocolName'])

ipdata['ProtocolName'] = encoder.fit\_transform(ipdata['ProtocolName'])

ipdata['ProtocolName']



ipdata.head(10)

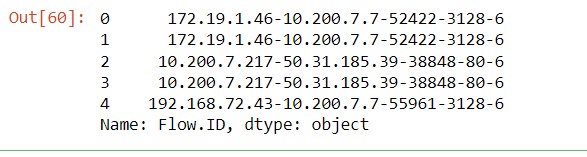


ipdata.shape



df1 = pd.Series(df['Flow.ID']).head()

df1



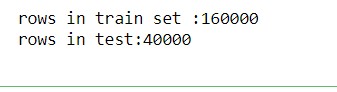
X1 = ipdata.iloc[:, : 69].values

Y1 = ipdata.iloc[: , 69].values

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X1, Y1, test\_size = 0.2, random\_state = 77)

print(f"rows in train set :{len(X\_train)}\nrows in test:{len(X\_test)}\n")



from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train)

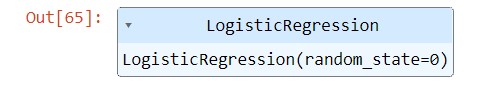
X\_test = sc.transform(X\_test)

**LOGISTIC REGRESSION**

from sklearn.linear\_model import LogisticRegression

classifier = LogisticRegression(random\_state = 0)

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

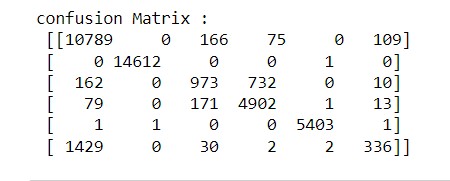


y\_pred = classifier.predict(X\_test)9

from sklearn.metrics import confusion\_matrix, accuracy\_score, classification\_report

cm = confusion\_matrix(y\_test, y\_pred)

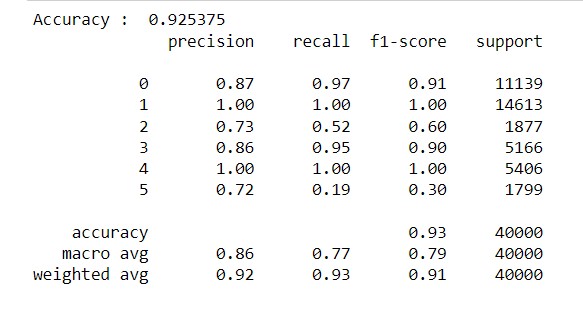
print("confusion Matrix : \n", cm)



from sklearn.metrics import accuracy\_score

print("Accuracy : ", accuracy\_score( y\_pred,y\_test))

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



**SVM**

from sklearn.svm import SVC

classifier = SVC(kernel = 'linear', random\_state = 0)

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

y\_pred = classifier.predict(X\_test)

from sklearn.metrics import accuracy\_score

print("Accuracy : ", accuracy\_score( y\_pred,y\_test))

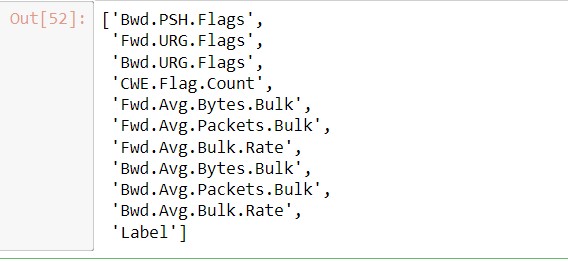
print(classification\_report(y\_test,y\_pred))

**RANDOM FOREST**

from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier(n\_estimators = 10, criterion= 'entropy', random\_state= 0)

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

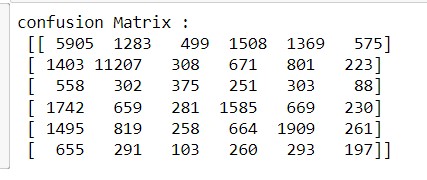


y\_pred = classifier.predict(X\_test)

from sklearn.metrics import confusion\_matrix, accuracy\_score, classification\_report

cm = confusion\_matrix(y\_test, y\_pred)

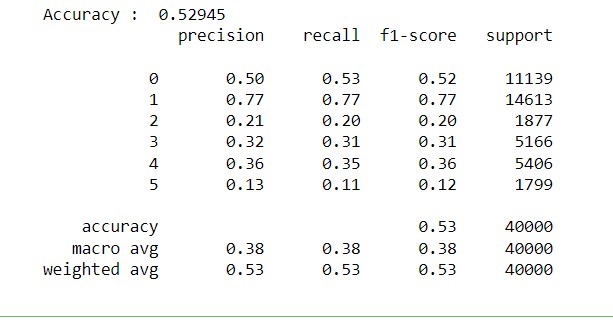
print("confusion Matrix : \n", cm)



from sklearn.metrics import accuracy\_score

print("Accuracy : ", accuracy\_score( y\_pred,y\_test))

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



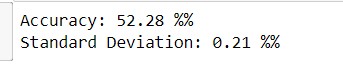
**# CROSS VALIDATION**

from sklearn.model\_selection import cross\_val\_score

accuracies = cross\_val\_score (estimator = classifier, X = X\_train, y = y\_train, cv = 4)

print("Accuracy: {:.2f} %%".format(accuracies.mean()\*100))

print("Standard Deviation: {:.2f} %%". format (accuracies.std ()\*100))



**K-NN**

from sklearn.decomposition import PCA

pca = PCA(n\_components = 1 )

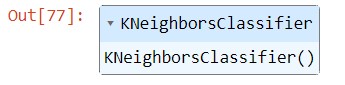
X\_train = pca.fit\_transform(X\_train)

X\_test = pca.transform(X\_test)

from sklearn.neighbors import KNeighborsClassifier

classifier = KNeighborsClassifier(n\_neighbors = 5, metric = 'minkowski', p = 2)

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



y\_pred= classifier.predict(X\_test)

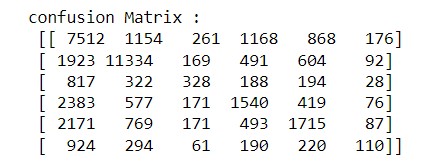
cm=confusion\_matrix(y\_test,y\_pred)

print("confusion Matrix : \n", cm)

from sklearn.metrics import accuracy\_score

print("Accuracy : ", accuracy\_score( y\_pred,y\_test))

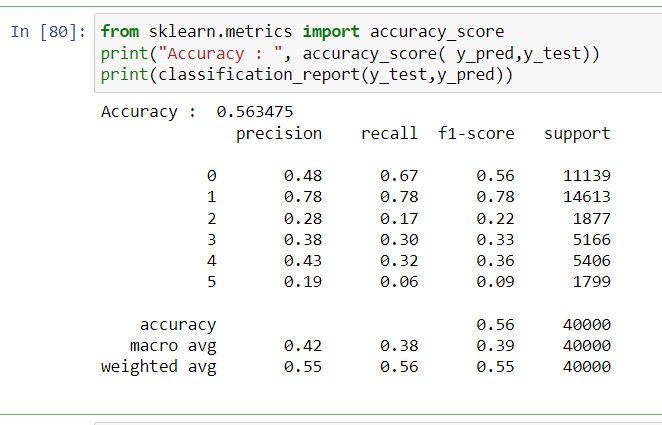
print(classification\_report(y\_test,y\_pred))



from sklearn.metrics import accuracy\_score

print("Accuracy : ", accuracy\_score( y\_pred,y\_test))

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



**# CROSS VALIDATION**

from sklearn.model\_selection import cross\_val\_score

accuracies = cross\_val\_score (estimator = classifier, X = X\_train, y = y\_train, cv = 4)

print("Accuracy: {:.2f} %%".format(accuracies.mean()\*100))

print("Standard Deviation: {:.2f} %%". format (accuracies.std ()\*100))

