```
"""
● This project will test your data science abilities as well as
Machine learning modeling abilities
● you will find the dataset which is related to DDoS Attacks by
following this link
(https://www.kaggle.com/siddharthm1698/ddos-botnet-attack-on-iot-
devices?select=DDoSdata.csv)
Download this data
● The data is highly undistributed
● Convert every attribute data type into float data type so attributes
require one hot encoding (label encoder)
● You need to apply correlation and variance concepts so that you will
take only important columns into consideration
● Use this data for modeling
o Create
Logistic regression
Random Forest
Decision Tree
Note : Accuracy should be above 90%
"""

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import svm
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report
df=pd.read_csv("E:\DDoSdata.csv")
df.head(2)
```

```
C:\Users\Renuka\AppData\Local\Temp\ipykernel_694780\2316441074.py:7:
DtypeWarning: Columns (8,10) have mixed types. Specify dtype option on
import or set low_memory=False.
  df=pd.read_csv("E:\DDoSdata.csv")
```

	Unnamed: 0	pkSeqID	stime	flgs	flgs_number	proto	proto_number	\
0	1650261	1650261	1.528103e+09	e	1	tcp	1	
1	1650262	1650262	1.528103e+09	e	1	tcp	1	

	saddr	sport	daddr	...	AR_P_Proto_P_DstIP	\
0	192.168.100.150	54110	192.168.100.3	...	1.21662	
1	192.168.100.150	54112	192.168.100.3	...	1.21662	

	N_IN_Conn_P_DstIP	N_IN_Conn_P_SrcIP	AR_P_Proto_P_Sport	\
0	40	38	1.56093	
1	40	38	1.56107	

```
   AR_P_Proto_P_Dport  Pkts_P_State_P_Protocol_P_DestIP  \
0             1.21662                               328
1             1.21662                               328

   Pkts_P_State_P_Protocol_P_SrcIP  attack  category  subcategory
0                              308       1      DDoS         HTTP
1                              308       1      DDoS         HTTP

[2 rows x 47 columns]
```

```
df.shape
```

```
(1927101, 47)
```

```
mb = df.memory_usage().sum() / 1024**2
print('Memory usage of dataframe is {:.2f} MB'.format(mb))
```

```
Memory usage of dataframe is 691.02 MB
```

```
from sklearn import preprocessing

# label_encoder object knows how to understand word labels.
label_encoder = preprocessing.LabelEncoder()

# Encode labels in column
df['subcategory']= label_encoder.fit_transform(df['subcategory'])

df['subcategory'].unique()
```

```
array([0, 2, 3, 1])
```

```
label_encoder = preprocessing.LabelEncoder()

# Encode labels in column
df['category']= label_encoder.fit_transform(df['category'])

df['category'].unique()
```

```
array([0, 1])
```

```
label_encoder = preprocessing.LabelEncoder()

# Encode labels in column
df['proto']= label_encoder.fit_transform(df['proto'])

df['proto'].unique()
```

```
array([3, 0, 4, 1, 2])
```

```
df.select_dtypes('object').columns
```

```
Index(['flgs', 'saddr', 'sport', 'daddr', 'dport', 'state'],
dtype='object')
```

```
df.drop(df.select_dtypes('object').columns,inplace=True,axis=1)
```

```
df.head(2)
```

```
   Unnamed: 0  pkSeqID         stime  flgs_number  proto  proto_number
pkts  \
0     1650261  1650261  1.528103e+09            1      3             1
10
1     1650262  1650262  1.528103e+09            1      3             1
10

   bytes  state_number         ltime  ...  AR_P_Proto_P_DstIP  \
0   1729             1  1.528103e+09  ...             1.21662
1   1604             1  1.528103e+09  ...             1.21662

   N_IN_Conn_P_DstIP  N_IN_Conn_P_SrcIP  AR_P_Proto_P_Sport  \
0                 40                 38             1.56093
1                 40                 38             1.56107

   AR_P_Proto_P_Dport  Pkts_P_State_P_Protocol_P_DestIP  \
0             1.21662                               328
1             1.21662                               328

   Pkts_P_State_P_Protocol_P_SrcIP  attack  category  subcategory
0                              308       1         0            0
1                              308       1         0            0

[2 rows x 41 columns]
```

```
from sklearn.feature_selection import VarianceThreshold
var_thres=VarianceThreshold(threshold=0.5)
var_thres.fit(df)
```

```
VarianceThreshold(threshold=0.5)
```

```
var_thres.get_support
```

```
<bound method SelectorMixin.get_support of
VarianceThreshold(threshold=0.5)>
```

```
df.columns[var_thres.get_support() == True]
```

```
Index(['Unnamed: 0', 'pkSeqID', 'stime', 'proto_number', 'pkts',
'bytes',
       'state_number', 'ltime', 'seq', 'dur', 'mean', 'stddev', 'sum',
'min',
       'max', 'spkts', 'dpkts', 'sbytes', 'dbytes', 'rate', 'srate',
'drate',
       'TnBPSrcIP', 'TnBPDstIP', 'TnP_PSrcIP', 'TnP_PDstIP',
```

```
'TnP_PerProto',
       'TnP_Per_Dport', 'AR_P_Proto_P_SrcIP', 'AR_P_Proto_P_DstIP',
       'N_IN_Conn_P_DstIP', 'N_IN_Conn_P_SrcIP', 'AR_P_Proto_P_Sport',
       'AR_P_Proto_P_Dport', 'Pkts_P_State_P_Protocol_P_DestIP',
       'Pkts_P_State_P_Protocol_P_SrcIP'],
      dtype='object')
```

```
columns_having_var_more_than_50 = df.columns[var_thres.get_support()
== True]
```

```
df.columns[var_thres.get_support() == False]
```

```
Index(['flgs_number', 'proto', 'attack', 'category', 'subcategory'],
dtype='object')
```

```
columns_having_var_less_than_50 = df.columns[var_thres.get_support()
== False]
```

```
len(columns_having_var_more_than_50)
```

```
36
```

```
len(df.columns)
```

```
41
```

```
len(columns_having_var_less_than_50)
```

```
5
```

```
df.drop(columns_having_var_less_than_50,inplace = True,axis= 1)
```

```
df.columns
```

```
Index(['Unnamed: 0', 'pkSeqID', 'stime', 'proto_number', 'pkts',
'bytes',
       'state_number', 'ltime', 'seq', 'dur', 'mean', 'stddev', 'sum',
'min',
       'max', 'spkts', 'dpkts', 'sbytes', 'dbytes', 'rate', 'srate',
'drate',
       'TnBPSrcIP', 'TnBPDstIP', 'TnP_PSrcIP', 'TnP_PDstIP',
'TnP_PerProto',
       'TnP_Per_Dport', 'AR_P_Proto_P_SrcIP', 'AR_P_Proto_P_DstIP',
       'N_IN_Conn_P_DstIP', 'N_IN_Conn_P_SrcIP', 'AR_P_Proto_P_Sport',
       'AR_P_Proto_P_Dport', 'Pkts_P_State_P_Protocol_P_DestIP',
       'Pkts_P_State_P_Protocol_P_SrcIP'],
      dtype='object')
```

```
matrix = df.corr()
```

```
def correlation(dataset, threshold):#X_train,0.5
    col_corr = set()  # Set of all the names of correlated columns
    col_corr_lst = []
```

```
    print(f"set initial {col_corr}")
    print(f"list initial {col_corr_lst}")
    corr_arr = df.corr() #corr_arr is my correlaion matrix which is 2d
    for row in range(len(corr_arr)):
        for col in range(row):
            if abs(corr_arr.iloc[row, col]) > threshold: # we are
interested in absolute coeff value
                colname = corr_arr.columns[row]  # getting the name of
column
                col_corr_lst.append(colname)
                col_corr.add(colname)
                print(f"colname name which is correlated is
{colname}")
                print(f"set {col_corr}")
                print(f"lst {col_corr_lst}")


    print(f"list is {col_corr_lst}")
    return col_corr

df.head(2)
```

```
   Unnamed: 0  pkSeqID         stime  proto_number  pkts  bytes
state_number  \
0     1650261  1650261  1.528103e+09             1    10   1729
1
1     1650262  1650262  1.528103e+09             1    10   1604
1

          ltime  seq       dur  ...  TnP_PerProto  TnP_Per_Dport  \
0  1.528103e+09   20  6.406424  ...           328            700
1  1.528103e+09   21  6.405851  ...           328            700

   AR_P_Proto_P_SrcIP  AR_P_Proto_P_DstIP  N_IN_Conn_P_DstIP  \
0             1.26889             1.21662                 40
1             1.26889             1.21662                 40

   N_IN_Conn_P_SrcIP  AR_P_Proto_P_Sport  AR_P_Proto_P_Dport  \
0                 38             1.56093             1.21662
1                 38             1.56107             1.21662

   Pkts_P_State_P_Protocol_P_DestIP  Pkts_P_State_P_Protocol_P_SrcIP
0                               328                              308
1                               328                              308

[2 rows x 36 columns]
```

```
x=np.asarray(df[['pkSeqID', 'stime', 'pkts', 'bytes',
       'state_number', 'ltime', 'seq', 'dur', 'mean', 'stddev', 'sum',
'min',
```

```
       'max', 'spkts', 'dpkts', 'sbytes', 'dbytes', 'rate', 'srate',
'drate',
       'TnBPSrcIP', 'TnBPDstIP', 'TnP_PSrcIP', 'TnP_PDstIP',
'TnP_PerProto',
       'TnP_Per_Dport', 'AR_P_Proto_P_SrcIP', 'AR_P_Proto_P_DstIP',
       'N_IN_Conn_P_DstIP', 'N_IN_Conn_P_SrcIP', 'AR_P_Proto_P_Sport',
       'AR_P_Proto_P_Dport', 'Pkts_P_State_P_Protocol_P_DestIP',
       'Pkts_P_State_P_Protocol_P_SrcIP']])#iv



y=np.asarray(df['proto_number'])#dv

from sklearn import preprocessing
x=preprocessing.StandardScaler().fit(x).transform(x)

from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=.2,random
_state=200)
print("Train set:",x_train.shape,y_train.shape)
print("Test set:",x_test.shape,y_test.shape)
```

```
Train set: (1541680, 34) (1541680,)
Test set: (385421, 34) (385421,)
```

```
from sklearn.linear_model import LogisticRegression
#from sklearn.metrics import classification_report, confusion_matrix
model = LogisticRegression(solver='liblinear', random_state=0)
LR=LogisticRegression(solver='saga')
LR.fit(x_train,y_train)
LR

yhat=LR.predict(x_test)
yhat[:5]

yhat_proba=LR.predict_proba(x_test)
yhat_proba[:5]
```

```
C:\Users\Renuka\anaconda3\lib\site-packages\sklearn\linear_model\
_sag.py:352: ConvergenceWarning: The max_iter was reached which means
the coef_ did not converge
  warnings.warn(

array([[9.94690759e-01, 1.56402221e-03, 6.44145602e-04, 1.55216789e-
03,
        1.54890519e-03],
       [9.62284314e-01, 8.03815523e-03, 1.34244565e-02, 8.13118377e-
03,
        8.12189029e-03],
       [3.73931688e-03, 4.26931992e-03, 9.83366124e-01, 4.31314788e-
03,
```

```
        4.31209134e-03],
       [9.34722855e-01, 1.17894876e-02, 2.98071953e-02, 1.18339528e-
02,
        1.18465088e-02],
       [6.56367233e-03, 4.51969811e-03, 9.79814375e-01, 4.55542509e-
03,
        4.54682996e-03]])
```

```
from sklearn.metrics import f1_score
f1_score(y_test,yhat, average='micro')
```

```
0.9990529836205084
```

```
from sklearn.tree import DecisionTreeClassifier
tree=DecisionTreeClassifier(criterion='entropy', random_state=0)
tree
```

```
DecisionTreeClassifier(criterion='entropy', random_state=0)
```

```
tree.fit(x_train,y_train)
```

```
DecisionTreeClassifier(criterion='entropy', random_state=0)
```

```
y_pred=tree.predict(x_test)
```

```
from sklearn import metrics
print("DecisionTrees's Accuracy: ",metrics.r2_score(y_pred,y_test))
```

```
DecisionTrees's Accuracy:  0.9999688574333424
```

```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(random_state=42)
rfc.fit(x_train, y_train)
rfc_pred = rfc.predict(x_test)
```

```
from sklearn.metrics import classification_report,confusion_matrix
confusion_matrix(y_test, rfc_pred)
```

```
array([[195766,      1,      0,      0],
       [     0,     14,      0,      0],
       [     0,      0, 189635,      0],
       [     0,      0,      0,      5]], dtype=int64)
```

```
classification_report(y_test, rfc_pred)
from sklearn import metrics
metrics.accuracy_score(y_test, y_pred)
```

```
0.9999922163037303
```