### **Importing** **all** **the** **required** **libraries**

**import** pandas **as** pd **import** numpy **as** np **import** itertools

**import** matplotlib.pyplot **as** plt

**import** vpython **as** vs

**from** sklearn.preprocessing **import** MinMaxScaler

**from** IPython.display **import** display

**from** sklearn.model\_selection **import** train\_test\_split **from** sklearn.linear\_model **import** LogisticRegression **from** sklearn.svm **import** SVC

**from** sklearn.ensemble **import** RandomForestClassifier **from** sklearn.neighbors **import** KNeighborsClassifier **from** sklearn.neural\_network **import** MLPClassifier

**from** sklearn.metrics **import** accuracy\_score,roc\_curve,auc,classification\_report,confusion\_matrix

**Loading dataset**

pureData **=** pd.read\_csv("C:\Users\sabba\OneDrive\Desktop\Python\benign\_traffic.csv")

print("Non-malicious Datasets") display(pureData.head())

maliciousDataset **=** pd.read\_csv("C:\Users\sabba\OneDrive\Desktop\Python\junk.csv")

print("Malicious Dataset") display(maliciousDataset.head())

### **Description** **of** **dataset**

print("There are %d records with %d features in non-malicious dataset"**%**(pureData.shape[0],pure Data.shape[1]))

display(pureData.describe())

print("There are %d records with %d features in malicious dataset"**%**(maliciousDataset.shape[0], maliciousDataset.shape[1]))

display(maliciousDataset.describe())

**Adding label to the dataset**

print("Adding output column in the datasets with all 0 in pureData dataset and all 1 in maliciousData set dataset")

pureData["output"] **=** 0

maliciousDataset["output"] **=** 1

### **Combining** **Non-malicious and Malicious datasets**

dataset **=** pd.concat([pureData, maliciousDataset], axis**=**0)

print("There are %d records with %d features in combined dataset"**%**(dataset.shape[0],dataset.sha pe[1]))

**Splitting input and output**

Output **=** dataset.output

Input **=** dataset.loc[:,"MI\_dir\_L5\_weight":"HpHp\_L0.01\_pcc"] print("Output shape :-",Output.shape)

print("Intput shape :-",Input.shape)

### **Preprocessing**

Output**=**np.array(Output).flatten()

**Normalization**

print("Calculation Z-score normalization which converts all indicators to a common scale with an av erage of zero and standard deviation of one. The average of zero means that it avoids introducing ag gregation distortions stemming from differences in indicators' means.") datasetNormalised**=**(dataset**-**dataset.mean())**/**(dataset.std()) datasetNormalised\_array**=**np.array(datasetNormalised)

print("Data after normalisation") display(datasetNormalised.head())

### **Split dataset (80-20)**

X\_train, X\_test, y\_train, y\_test **=** train\_test\_split(datasetNormalised\_array, Output, test\_size **=** 0.2, rand om\_state **=** 3)

print ("Training set has {} samples.".format(X\_train.shape[0])) print ("Testing set has {} samples.".format(X\_test.shape[0]))

**Model definition**

SVM **=** SVC(probability**=True**)

RF **=** RandomForestClassifier(n\_estimators**=**1500)

### **Model training and prediction**

**def** train\_predict(model, X\_train, y\_train, X\_test, y\_test): model.fit(X\_train, y\_train)

y\_pred **=** model.predict(X\_test)

acc\_test **=** accuracy\_score(y\_test,y\_pred) report **=** classification\_report(y\_test,y\_pred)

**return** acc\_test,report

**Prediction and accuracy score**

SVM\_acc,SVM\_report **=** train\_predict(SVM,X\_train,y\_train,X\_test,y\_test) print("The accuracy score of SVM is %f"**%**SVM\_acc) print("Classification report :-")

print(SVM\_report)

RF\_acc,RF\_report **=** train\_predict(RF,X\_train,y\_train,X\_test,y\_test) print("The accuracy score of Random Forest Classifier is %f"**%**RF\_acc) print("Classification report :-")

print(RF\_report)

### **Confusion matrix**

**def** plot\_confusion\_matrix(cm, classes,normalize**=False**,title**=**'Confusion matrix',cmap**=**plt.cm.Blues)

:

plt.imshow(cm, interpolation='nearest', cmap=cmap)

plt.title(title)

plt.colorbar()

tick\_marks = np.arange(len(classes))

plt.xticks(tick\_marks, classes)

plt.yticks(tick\_marks, classes)

thresh = cm.max() / 2.

for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):

plt.text(j, i, cm[i, j],

horizontalalignment="center",

color="white" if cm[i, j] > thresh else "black")

plt.tight\_layout()

plt.ylabel('True label')

plt.xlabel('Predicted label')

def plot\_confusion\_matrix(cm, classes, normalize=False, title='Confusion matrix', cmap=plt.cm.Blues):

plt.imshow(cm, interpolation='nearest', cmap=cmap)

plt.title(title)

plt.colorbar()

tick\_marks = np.arange(len(classes))

plt.xticks(tick\_marks, classes)

plt.yticks(tick\_marks, classes)

thresh = cm.max() / 2.

for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):

plt.text(j, i, cm[i, j],

horizontalalignment="center",

color="white" if cm[i, j] > thresh else "black")

plt.tight\_layout()

plt.ylabel('True label')

plt.xlabel('Predicted label')

# Assuming y\_test and X\_test are defined

cnf\_matrix = confusion\_matrix(y\_test, RF.predict(X\_test))

np.set\_printoptions(precision=2)

plt.figure()

plot\_confusion\_matrix(cnf\_matrix, classes=[1, 0], title='RF Confusion matrix')

plt.show()