**SOURCE CODE**

# Importing necessary libraries

import pandas as pd  
from sklearn.metrics import mean\_squared\_error,confusion\_matrix, precision\_score, recall\_score, auc,roc\_curve  
from sklearn.model\_selection import train\_test\_split  
import numpy as np  
from collections import Counter  
from sklearn import metrics  
import matplotlib.pyplot as plt  
from sklearn.metrics import classification\_report,confusion\_matrix,accuracy\_score

# Importing dataset (2 columns - urls, type)

df=pd.read\_csv('D:\SASTRA UNIVERSITY\Mini\_Project\dataset.csv')  
print(df.shape)  
df.head()

# IP Address

import re  
def having\_ip\_address(urls):  
 match = re.search(  
 '(([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\.([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\.([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\.'  
 '([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\/)|' # IPv4  
 '((0x[0-9a-fA-F]{1,2})\\.(0x[0-9a-fA-F]{1,2})\\.(0x[0-9a-fA-F]{1,2})\\.(0x[0-9a-fA-F]{1,2})\\/)' # IPv4 in hexadecimal  
 '(?:[a-fA-F0-9]{1,4}:){7}[a-fA-F0-9]{1,4}', urls) # Ipv6  
 if match:  
 return 1  
 else:  
 return 0  
df['ip\_in\_URL'] = df['urls'].apply(lambda i: having\_ip\_address(i))

# Random words

def count\_random\_words(urls):  
 li1 = re.findall("[A-Za-z]+",urls)  
 return len(li1)  
df['random\_words'] = df['urls'].apply(lambda i: count\_random\_words(i))  
  
def avg\_len\_random(urls):  
 sum = 0  
 li2 = re.findall("[A-Za-z]+",urls)  
 for u in li2:  
 sum = sum + len(u)  
 return (sum/len(li2))  
  
df['avg\_len\_random\_words'] = df['urls'].apply(lambda i: avg\_len\_random(i))

# English Words

from enchant import Dict  
import swifter  
d = Dict("en\_US")  
pattern = re.compile("[A-Za-z]+")   
  
def count\_english\_words(urls):   
 li1 = pattern.findall(urls)   
 return sum([1 for word in li1 if d.check(word)])   
  
def avg\_len\_english(urls):   
 li2 = pattern.findall(urls)   
 english\_words = [u for u in li2 if d.check(u)]   
 return sum(map(len, english\_words)) / len(english\_words) if len(english\_words)!=0 else 0  
  
df['english\_words'] = df['urls'].swifter.apply(count\_english\_words)  
df['avg\_len\_english\_words'] = df['urls'].swifter.apply(avg\_len\_english)

# Lexical features (;, &, http, https, \_, #, @, url\_length)

df['http\_in\_URL'] = df['urls'].apply(lambda i: i.count('http'))  
df['semicolon\_in\_URL'] = df['urls'].apply(lambda i: i.count(';'))  
df['AND\_in\_URL'] = df['urls'].apply(lambda i: i.count('&'))  
df['URL\_length'] = df['urls'].apply(lambda i: len(str(i)))  
  
df['underscore'] = df['urls'].apply(lambda i: i.count('\_'))  
df['//'] = df['urls'].apply(lambda i: i.count('//'))  
df['@'] = df['urls'].apply(lambda i: i.count('@'))  
df['hash'] = df['urls'].apply(lambda i: i.count('#'))  
df['https'] = df['urls'].apply(lambda i: i.count('https'))

# Letters

def letter\_count(url):  
 letters = 0  
 for i in url:  
 if i.isalpha():  
 letters = letters + 1  
 return letters  
df['alphabets\_in\_URL']= df['urls'].apply(lambda i: letter\_count(i))  
  
def uppercase\_count(url):  
 letters1 = 0  
 for i in url:  
 if i>='A' and i<='Z':  
 letters1 = letters1 + 1  
 return letters1  
df['uppercase\_letters\_inURL']= df['urls'].apply(lambda i: uppercase\_count(i))  
  
def lowercase\_count(url):  
 letters2 = 0  
 for i in url:  
 if i>='a' and i<='z':  
 letters2 = letters2 + 1  
 return letters2  
df['lowercase\_letters\_inURL']= df['urls'].apply(lambda i: lowercase\_count(i))  
  
def uppercase\_ratio(url):  
 url\_len = len(str(url))  
 letters1 = 0  
 for i in url:  
 if i>='A' and i<='Z':  
 letters1 = letters1 + 1  
 return (letters1/url\_len)  
df['uppercase\_letters\_ratio\_inURL']= df['urls'].apply(lambda i: uppercase\_ratio(i))  
  
def lowercase\_ratio(url):  
 url\_len = len(str(url))  
 letters2 = 0  
 for i in url:  
 if i>='a' and i<='z':  
 letters2 = letters2 + 1  
 return (letters2/url\_len)  
df['lowercase\_letters\_ratio\_inURL']= df['urls'].apply(lambda i: lowercase\_ratio(i))

# special characters

def spl\_char\_count(url):  
 url\_len = len(str(url))  
 chars = 0  
 for i in url:  
 if i=='!' or i=='@' or i=='#' or i=='%' or i=='^' or i=='&' or i=='\*' or i=='(' or i==')':  
 chars = chars + 1  
 return chars  
df['Special\_char\_in\_URL']= df['urls'].apply(lambda i: spl\_char\_count(i))  
  
def spl\_char\_ratio(url):  
 url\_len = len(str(url))  
 chars = 0  
 for i in url:  
 if i=='!' or i=='@' or i=='#' or i=='%' or i=='^' or i=='&' or i=='\*' or i=='(' or i==')':  
 chars = chars + 1  
 return chars/url\_len  
df['Special\_char\_ratio']= df['urls'].apply(lambda i: spl\_char\_ratio(i))

# Numbers

def number\_ratio(url):  
 url\_len = len(str(url))  
 nums = 0  
 for i in url:  
 if i>='0' and i<='9':  
 nums = nums + 1  
 return (nums/url\_len)  
df['numbers\_ratio\_inURL']= df['urls'].apply(lambda i: number\_ratio(i))  
  
def number\_URL(url):  
 nums = 0  
 for i in url:  
 if i>='0' and i<='9':  
 nums = nums + 1  
 return (nums)  
df['numbers\_inURL']= df['urls'].apply(lambda i: number\_URL(i))

# Alphabets

def alphabet\_ratio(url):  
 url\_len = len(str(url))  
 letters = 0  
 for i in url:  
 if i.isalpha():  
 letters = letters + 1  
 return (letters/url\_len)  
df['alphabet\_ratio\_inURL']= df['urls'].apply(lambda i: alphabet\_ratio(i))

from urllib.parse import urlparse  
from math import log  
from string import ascii\_lowercase  
  
from tld import get\_tld  
  
def fd\_length(url):  
 urlpath= urlparse(url).path  
 try:  
 return len(urlpath.split('/')[1])  
 except:  
 return 0  
  
df['fd\_length'] = df['urls'].apply(lambda i: fd\_length(i))  
df['fd\_length2'] = df['urls'].apply(lambda i: fd\_length(i))  
  
df['tld'] = df['urls'].apply(lambda i: get\_tld(i,fail\_silently=True))  
def tld\_length(tld):  
 try:  
 return len(tld)  
 except:  
 return -1  
  
df['tld\_length'] = df['tld'].apply(lambda i: tld\_length(i))  
df['tld\_length2'] = df['tld'].apply(lambda i: tld\_length(i))  
  
  
del df['tld']  
  
def has\_server\_in\_string(url):  
 if 'server' in url.lower():  
 return 1  
 else:  
 return 0  
df['has\_server'] = df['urls'].apply(lambda i: has\_server\_in\_string(i))  
  
def has\_login\_in\_string(url):  
 if 'login' in url.lower():  
 return 1  
 else:  
 return 0  
df['has\_login'] = df['urls'].apply(lambda i: has\_login\_in\_string(i))  
  
def has\_client\_in\_string(url):  
 if 'client' in url.lower():  
 return 1  
 else:  
 return 0  
df['has\_client'] = df['urls'].apply(lambda i: has\_client\_in\_string(i))  
  
def has\_admin\_in\_string(url):  
 if 'admin' in url.lower():  
 return 1  
 else:  
 return 0  
df['has\_admin'] = df['urls'].apply(lambda i: has\_admin\_in\_string(i))  
  
def no\_of\_dir(url):  
 urldir = urlparse(url).path  
 return urldir.count('/')  
df['count\_dir'] = df['urls'].apply(lambda i: no\_of\_dir(i))  
df['count\_dir2'] = df['urls'].apply(lambda i: no\_of\_dir(i))

def shortening\_service(url):  
 match = re.search('bit\.ly|goo\.gl|shorte\.st|go2l\.ink|x\.co|ow\.ly|t\.co|tinyurl|tr\.im|is\.gd|cli\.gs|'  
 'yfrog\.com|migre\.me|ff\.im|tiny\.cc|url4\.eu|twit\.ac|su\.pr|twurl\.nl|snipurl\.com|'  
 'short\.to|BudURL\.com|ping\.fm|post\.ly|Just\.as|bkite\.com|snipr\.com|fic\.kr|loopt\.us|'  
 'doiop\.com|short\.ie|kl\.am|wp\.me|rubyurl\.com|om\.ly|to\.ly|bit\.do|t\.co|lnkd\.in|'  
 'db\.tt|qr\.ae|adf\.ly|goo\.gl|bitly\.com|cur\.lv|tinyurl\.com|ow\.ly|bit\.ly|ity\.im|'  
 'q\.gs|is\.gd|po\.st|bc\.vc|twitthis\.com|u\.to|j\.mp|buzurl\.com|cutt\.us|u\.bb|yourls\.org|'  
 'x\.co|prettylinkpro\.com|scrnch\.me|filoops\.info|vzturl\.com|qr\.net|1url\.com|tweez\.me|v\.gd|'  
 'tr\.im|link\.zip\.net',  
 url)  
 if match:  
 return -1  
 else:  
 return 1  
df['short\_url'] = df['urls'].apply(lambda i: shortening\_service(i))  
df['short\_url2'] = df['urls'].apply(lambda i: shortening\_service(i))  
df['short\_url3'] = df['urls'].apply(lambda i: shortening\_service(i))  
  
  
def number\_subdomain(url):  
 count = 0  
 domain = urlparse(url).netloc  
 dom = '.'.join(domain.split('.')[:-2])  
 for i in dom:  
 if i=='.':  
 count = count+1  
 return (count-1)  
  
df['number\_subdom'] = df['urls'].apply(lambda i: number\_subdomain(i))

# importing libraries for domain features

from urllib.parse import urlparse  
from math import log  
from string import ascii\_lowercase

# domain features (lexical)

def domain\_len(url):  
 domain = urlparse(str(url)).netloc  
 return len(domain)  
df['domain\_length']= df['urls'].apply(lambda i: domain\_len(i))  
  
def dom(url):  
 return (urlparse(str(url)).netloc)  
df['domain\_name'] = df['urls'].apply(lambda i: dom(i))  
df['dots\_in\_domain'] = df['domain\_name'].apply(lambda i: i.count('.'))  
df['hyphens\_in\_domain'] = df['domain\_name'].apply(lambda i: i.count('-'))  
  
del df['domain\_name']

df['url\_type'] = df['type'].apply(lambda i: i)

del df['type']

# Saving the dataset as csv file

df.to\_csv(r'D:\SASTRA UNIVERSITY\Mini\_Project\unclean.csv')  
df\_clean = df.loc[:,['ip\_in\_URL','random\_words','avg\_len\_random\_words','english\_words','avg\_len\_english\_words','http\_in\_URL','semicolon\_in\_URL','AND\_in\_URL','URL\_length','alphabets\_in\_URL','uppercase\_letters\_inURL','lowercase\_letters\_inURL','uppercase\_letters\_ratio\_inURL','lowercase\_letters\_ratio\_inURL','Special\_char\_in\_URL','numbers\_ratio\_inURL','alphabet\_ratio\_inURL','fd\_length','fd\_length2','tld\_length','tld\_length2','has\_server','has\_login','has\_client','has\_admin','count\_dir','count\_dir2','domain\_length','dots\_in\_domain','short\_url','short\_url2','short\_url3','number\_subdom','hyphens\_in\_domain','url\_type']]  
df\_clean.to\_csv(r'D:\SASTRA UNIVERSITY\Mini\_Project\clean.csv')

df.head()

# SelectK best algorithm for feature reduction

from sklearn.feature\_selection import SelectKBest, chi2  
data = pd.read\_csv(r'D:\SASTRA\_UNIVERSITY\Mini\_Project\unclean.csv')  
X = data.iloc[:, 2:-1]  
y = data.iloc[:, -1]  
  
  
selector = SelectKBest(chi2,k=20)  
X\_new = selector.fit\_transform(X, y)  
  
selected\_cols = X.columns[selector.get\_support()]  
  
print(selected\_cols)

Index(['ip\_in\_URL', 'random\_words', 'avg\_len\_random\_words', 'english\_words',  
 'avg\_len\_english\_words', 'semicolon\_in\_URL', 'AND\_in\_URL', 'URL\_length',  
 'http\_in\_URL', 'alphabets\_in\_URL', 'uppercase\_letters\_inURL',  
 'lowercase\_letters\_inURL', 'uppercase\_letters\_ratio\_inURL',  
 'lowercase\_letters\_ratio\_inURL', 'Special\_char\_in\_URL',  
 'numbers\_ratio\_inURL', 'alphabet\_ratio\_inURL', 'domain\_length',  
 'dots\_in\_domain', 'hyphens\_in\_domain'],  
 dtype='object')

# Correlation matrix

cols\_of\_interest = ['underscore','//','@','https','hash','numbers\_inURL','Special\_char\_ratio']  
corr\_matrix = df.corr()  
co\_mat = corr\_matrix.loc[:,:]  
co\_mat\_clean = corr\_matrix.loc[:,cols\_of\_interest]  
  
co\_mat.to\_csv('D:\SASTRA UNIVERSITY\Mini\_Project\correlation\_unclean.csv')  
co\_mat\_clean.to\_csv('D:\SASTRA UNIVERSITY\Mini\_Project\correlation\_clean.csv')

# Plotting heat maps

import seaborn as sns  
import matplotlib.pyplot as plt  
  
sns.heatmap(co\_mat, cmap='YlGnBu')  
  
plt.title('28x28')  
plt.xlabel('X-axis label')  
plt.ylabel('Y-axis label')  
  
plt.show()  
  
sns.heatmap(co\_mat\_clean, cmap='YlGnBu')  
  
plt.title('28x7')  
plt.xlabel('X-axis label')  
plt.ylabel('Y-axis label')  
  
plt.show()

# Importing necessary libraries

import pandas as pd  
from sklearn.metrics import mean\_squared\_error, auc,roc\_curve  
from sklearn.model\_selection import train\_test\_split  
import numpy as np  
from sklearn import metrics  
import matplotlib.pyplot as plt  
from sklearn.metrics import classification\_report,confusion\_matrix,accuracy\_score, precision\_score, recall\_score, f1\_score  
import seaborn as sns  
from mpl\_toolkits.mplot3d import Axes3D  
from sklearn.metrics import precision\_recall\_fscore\_support

# Importing dataset

df=pd.read\_csv('D:\SASTRA UNIVERSITY\Mini\_Project\clean.csv')  
print(df.shape)  
df.head()

# Splitting the inputs and output

X = df.iloc[:,1:-1].values  
y = df.iloc[:,-1].values  
print(X)  
print(y)

# Splitting the dataset into train and test set

from sklearn.model\_selection import train\_test\_split  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.25, random\_state = None)

# Feature scaling using z-score method

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  
lr = LogisticRegression(random\_state = None)  
lr.fit(X\_train, y\_train)  
  
probabilities = lr.predict\_proba(X\_test)  
threshold = 0.6  
  
y\_predLR = (probabilities[:, 1] > threshold).astype(int)  
  
  
accuracy = accuracy\_score(y\_test, y\_predLR)  
precision = precision\_score(y\_test, y\_predLR)  
recall = recall\_score(y\_test, y\_predLR)  
f1 = f1\_score(y\_test, y\_predLR)  
  
cm = confusion\_matrix(y\_test, y\_predLR)  
  
print("Accuracy: {:.2f}%".format(accuracy \* 100))  
print("Precision: {:.2f}%".format(precision \* 100))  
print("Recall: {:.2f}%".format(recall \* 100))  
print("F1 Score: {:.2f}%".format(f1 \* 100))  
print("Confusion Matrix:\n", cm)  
  
sns.heatmap(cm, annot=True, cmap='Blues', fmt='g')  
plt.xlabel('Predicted')  
plt.ylabel('True')  
plt.title('Confusion matrix')  
  
metric\_labels = ['Accuracy', 'Precision', 'Recall', 'F1 Score']  
metric\_scores = [accuracy, precision, recall, f1]  
plt.figure()  
plt.bar(np.arange(len(metric\_labels)), metric\_scores)  
plt.xticks(np.arange(len(metric\_labels)), metric\_labels)  
plt.ylim([0, 1])  
plt.title('Evaluation metrics')  
plt.show()  
  
  
precision, recall, f1\_scor, \_ = precision\_recall\_fscore\_support(y\_test, y\_predLR, average=None)  
  
benign = [precision[0], recall[0], f1\_scor[0]]  
malicious = [precision[1], recall[1], f1\_scor[1]]  
index = [0.5, 1.5, 2.5]  
  
fig = plt.figure()  
ax = fig.add\_subplot(111, projection='3d')  
ax.bar(index, malicious, zs=1, zdir='y', color='r', alpha=0.8)  
  
ax.bar(index, benign, zs=0, zdir='y', color='g', alpha=0.8)  
  
ax.set\_xlabel('Metric')  
ax.set\_ylabel('Class')  
ax.set\_zlabel('Value')  
ax.set\_xticks(index)  
ax.set\_xticklabels(['Precision', 'Recall', 'F1-Score'])  
  
ax.set\_yticks([0, 1])  
ax.set\_yticklabels(['Benign', 'Malicious'])  
  
ax.set\_xlim(0, 3)  
ax.set\_ylim(-1, 2)  
ax.set\_zlim(0, 1)  
plt.show()

# Random forest

from sklearn.ensemble import RandomForestClassifier  
  
rf = RandomForestClassifier(n\_estimators = 100)   
rf.fit(X\_train, y\_train)  
   
y\_predRF = rf.predict(X\_test)   
   
accuracy = accuracy\_score(y\_test, y\_predRF)  
precision = precision\_score(y\_test, y\_predRF)  
recall = recall\_score(y\_test, y\_predRF)  
f1 = f1\_score(y\_test, y\_predRF)  
  
cm = confusion\_matrix(y\_test, y\_predRF)  
  
print("Accuracy: {:.2f}%".format(accuracy \* 100))  
print("Precision: {:.2f}%".format(precision \* 100))  
print("Recall: {:.2f}%".format(recall \* 100))  
print("F1 Score: {:.2f}%".format(f1 \* 100))  
print("Confusion Matrix:\n", cm)  
  
sns.heatmap(cm, annot=True, cmap='Blues', fmt='g')  
plt.xlabel('Predicted')  
plt.ylabel('True')  
plt.title('Confusion matrix')  
  
metric\_labels = ['Accuracy', 'Precision', 'Recall', 'F1 Score']  
metric\_scores = [accuracy, precision, recall, f1]  
plt.figure()  
plt.bar(np.arange(len(metric\_labels)), metric\_scores)  
plt.xticks(np.arange(len(metric\_labels)), metric\_labels)  
plt.ylim([0, 1])  
plt.title('Evaluation metrics')  
plt.show()  
  
precision, recall, f1\_scor, \_ = precision\_recall\_fscore\_support(y\_test, y\_predRF, average=None)  
  
benign = [precision[0], recall[0], f1\_scor[0]]  
malicious = [precision[1], recall[1], f1\_scor[1]]  
index = [0.5, 1.5, 2.5]  
  
fig = plt.figure()  
ax = fig.add\_subplot(111, projection='3d')  
ax.bar(index, malicious, zs=1, zdir='y', color='r', alpha=0.8)  
  
ax.bar(index, benign, zs=0, zdir='y', color='g', alpha=0.8)  
  
ax.set\_xlabel('Metric')  
ax.set\_ylabel('Class')  
ax.set\_zlabel('Value')  
ax.set\_xticks(index)  
ax.set\_xticklabels(['Precision', 'Recall', 'F1-Score'])  
  
ax.set\_yticks([0, 1])  
ax.set\_yticklabels(['Benign', 'Malicious'])  
  
ax.set\_xlim(0, 3)  
ax.set\_ylim(-1, 2)  
ax.set\_zlim(0, 1)  
plt.show()

# K-nearest neighbours

from sklearn.neighbors import KNeighborsClassifier  
  
knn = KNeighborsClassifier(n\_neighbors = 5)  
   
knn.fit(X\_train, y\_train)  
y\_predKNN = knn.predict(X\_test)  
   
from sklearn.metrics import classification\_report, confusion\_matrix, accuracy\_score  
  
accuracy = accuracy\_score(y\_test, y\_predKNN)  
precision = precision\_score(y\_test, y\_predKNN)  
recall = recall\_score(y\_test, y\_predKNN)  
f1 = f1\_score(y\_test, y\_predKNN)  
  
cm = confusion\_matrix(y\_test, y\_predKNN)  
  
print("Accuracy: {:.2f}%".format(accuracy \* 100))  
print("Precision: {:.2f}%".format(precision \* 100))  
print("Recall: {:.2f}%".format(recall \* 100))  
print("F1 Score: {:.2f}%".format(f1 \* 100))  
print("Confusion Matrix:\n", cm)  
  
sns.heatmap(cm, annot=True, cmap='Blues', fmt='g')  
plt.xlabel('Predicted')  
plt.ylabel('True')  
plt.title('Confusion matrix')  
  
metric\_labels = ['Accuracy', 'Precision', 'Recall', 'F1 Score']  
metric\_scores = [accuracy, precision, recall, f1]  
plt.figure()  
plt.bar(np.arange(len(metric\_labels)), metric\_scores)  
plt.xticks(np.arange(len(metric\_labels)), metric\_labels)  
plt.ylim([0, 1])  
plt.title('Evaluation metrics')  
plt.show()  
  
  
precision, recall, f1\_scor, \_ = precision\_recall\_fscore\_support(y\_test, y\_predRF, average=None)  
  
benign = [precision[0], recall[0], f1\_scor[0]]  
malicious = [precision[1], recall[1], f1\_scor[1]]  
index = [0.5, 1.5, 2.5]  
  
fig = plt.figure()  
ax = fig.add\_subplot(111, projection='3d')  
ax.bar(index, malicious, zs=1, zdir='y', color='r', alpha=0.8)  
  
ax.bar(index, benign, zs=0, zdir='y', color='g', alpha=0.8)  
  
ax.set\_xlabel('Metric')  
ax.set\_ylabel('Class')  
ax.set\_zlabel('Value')  
ax.set\_xticks(index)  
ax.set\_xticklabels(['Precision', 'Recall', 'F1-Score'])  
  
ax.set\_yticks([0, 1])  
ax.set\_yticklabels(['Benign', 'Malicious'])  
  
ax.set\_xlim(0, 3)  
ax.set\_ylim(-1, 2)  
ax.set\_zlim(0, 1)  
plt.show()

# Support Vector Machine

from sklearn.svm import SVC   
svc = SVC(kernel='rbf')   
  
svc.fit(X\_train, y\_train)   
y\_predSVC = svc.predict(X\_test)  
  
accuracy = accuracy\_score(y\_test, y\_predSVC)  
precision = precision\_score(y\_test, y\_predSVC)  
recall = recall\_score(y\_test, y\_predSVC)  
f1 = f1\_score(y\_test, y\_predSVC)  
  
cm = confusion\_matrix(y\_test, y\_predSVC)  
  
print("Accuracy: {:.2f}%".format(accuracy \* 100))  
print("Precision: {:.2f}%".format(precision \* 100))  
print("Recall: {:.2f}%".format(recall \* 100))  
print("F1 Score: {:.2f}%".format(f1 \* 100))  
print("Confusion Matrix:\n", cm)  
  
sns.heatmap(cm, annot=True, cmap='Blues', fmt='g')  
plt.xlabel('Predicted')  
plt.ylabel('True')  
plt.title('Confusion matrix')  
  
metric\_labels = ['Accuracy', 'Precision', 'Recall', 'F1 Score']  
metric\_scores = [accuracy, precision, recall, f1]  
plt.figure()  
plt.bar(np.arange(len(metric\_labels)), metric\_scores)  
plt.xticks(np.arange(len(metric\_labels)), metric\_labels)  
plt.ylim([0, 1])  
plt.title('Evaluation metrics')  
plt.show()  
  
  
precision, recall, f1\_scor, \_ = precision\_recall\_fscore\_support(y\_test, y\_predSVC, average=None)  
  
benign = [precision[0], recall[0], f1\_scor[0]]  
malicious = [precision[1], recall[1], f1\_scor[1]]  
index = [0.5, 1.5, 2.5]  
  
fig = plt.figure()  
ax = fig.add\_subplot(111, projection='3d')  
ax.bar(index, malicious, zs=1, zdir='y', color='r', alpha=0.8)  
  
ax.bar(index, benign, zs=0, zdir='y', color='g', alpha=0.8)  
  
ax.set\_xlabel('Metric')  
ax.set\_ylabel('Class')  
ax.set\_zlabel('Value')  
ax.set\_xticks(index)  
ax.set\_xticklabels(['Precision', 'Recall', 'F1-Score'])  
  
ax.set\_yticks([0, 1])  
ax.set\_yticklabels(['Benign', 'Malicious'])  
  
ax.set\_xlim(0, 3)  
ax.set\_ylim(-1, 2)  
ax.set\_zlim(0, 1)  
plt.show()

# Gaussian Naive Bayes

X\_train1, X\_test1, y\_train1, y\_test1 = train\_test\_split(X, y, test\_size = 0.15, random\_state = 0)  
  
  
import numpy as np  
  
class GaussianNaiveBayes:  
 def fit(self, X, y):  
 self.classes = np.unique(y)  
 self.num\_classes = len(self.classes)  
 self.means = []  
 self.variances = []  
 self.class\_prior = []  
   
 for i in range(self.num\_classes):  
 X\_class = X[y == self.classes[i]]  
 self.means.append(np.mean(X\_class, axis=0))  
 self.variances.append(np.var(X\_class, axis=0) + 0.0102)  
 self.class\_prior.append(len(X\_class) / len(X))  
   
 def predict(self, X):  
 log\_posterior = np.zeros((X.shape[0], self.num\_classes))  
   
 for i in range(self.num\_classes):  
 mean = self.means[i]  
 variance = self.variances[i]  
 prior = np.log(self.class\_prior[i])  
   
 log\_likelihood = np.sum(-0.5 \* np.log(2 \* np.pi \* variance)  
 - 0.5 \* ((X - mean) \*\* 2 / variance), axis=1)  
   
 log\_posterior[:, i] = log\_likelihood + prior  
   
 return self.classes[np.argmax(log\_posterior, axis=1)]

gnb = GaussianNaiveBayes()  
gnb.fit(X\_train1, y\_train1)  
  
y\_predNB = gnb.predict(X\_test1)  
  
accuracy = accuracy\_score(y\_test1, y\_predNB)  
precision = precision\_score(y\_test1, y\_predNB)  
recall = recall\_score(y\_test1, y\_predNB)  
f1 = f1\_score(y\_test1, y\_predNB)  
  
cm = confusion\_matrix(y\_test1, y\_predNB)  
  
print("Accuracy: {:.2f}%".format(accuracy \* 100))  
print("Precision: {:.2f}%".format(precision \* 100))  
print("Recall: {:.2f}%".format(recall \* 100))  
print("F1 Score: {:.2f}%".format(f1 \* 100))  
print("Confusion Matrix:\n", cm)  
  
sns.heatmap(cm, annot=True, cmap='Blues', fmt='g')  
plt.xlabel('Predicted')  
plt.ylabel('True')  
plt.title('Confusion matrix')  
  
metric\_labels = ['Accuracy', 'Precision', 'Recall', 'F1 Score']  
metric\_scores = [accuracy, precision, recall, f1]  
plt.figure()  
plt.bar(np.arange(len(metric\_labels)), metric\_scores)  
plt.xticks(np.arange(len(metric\_labels)), metric\_labels)  
plt.ylim([0, 1])  
plt.title('Evaluation metrics')  
plt.show()  
  
  
precision, recall, f1\_scor, \_ = precision\_recall\_fscore\_support(y\_test1, y\_predNB, average=None)  
  
benign = [precision[0], recall[0], f1\_scor[0]]  
malicious = [precision[1], recall[1], f1\_scor[1]]  
index = [0.5, 1.5, 2.5]  
  
fig = plt.figure()  
ax = fig.add\_subplot(111, projection='3d')  
ax.bar(index, malicious, zs=1, zdir='y', color='r', alpha=0.8)  
  
ax.bar(index, benign, zs=0, zdir='y', color='g', alpha=0.8)  
  
ax.set\_xlabel('Metric')  
ax.set\_ylabel('Class')  
ax.set\_zlabel('Value')  
ax.set\_xticks(index)  
ax.set\_xticklabels(['Precision', 'Recall', 'F1-Score'])  
  
ax.set\_yticks([0, 1])  
ax.set\_yticklabels(['Benign', 'Malicious'])  
  
ax.set\_xlim(0, 3)  
ax.set\_ylim(-1, 2)  
ax.set\_zlim(0, 1)  
plt.show()

**SOURCE CODE (GUI)**

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title>Machine Learning Results</title>

<style>

body {

font-family: sans-serif;

margin: 0;

padding: 0;

}

header {

background-color: #333;

color: white;

padding: 20px;

text-align: center;

}

h1 {

margin: 0;

}

.container {

display: flex;

flex-wrap: wrap;

justify-content: center;

}

.column {

flex: 1;

padding: 20px;

}

#results {

display: flex;

justify-content: center;

align-items: center;

flex-wrap: wrap;

margin-top: 50px;

}

#results select {

font-size: 16px;

padding: 10px;

margin-right: 20px;

border-radius: 5px;

}

#results table {

border-collapse: collapse;

width: 100%;

max-width: 800px;

}

#results th,

#results td {

text-align: left;

padding: 8px;

border-bottom: 1px solid #ddd;

}

#results th {

background-color: #333;

color: white;

}

#plots-container {

display: flex;

justify-content: center;

align-items: center;

flex-wrap: wrap;

margin-top: 50px;

}

.plot {

margin: 20px;

text-align: center;

}

.plot h2 {

margin-top: 0;

}

</style>

</head>

<body>

<header>

<h1>Machine Learning Results</h1>

</header>

<div class="container">

<div class="column">

<div id="results">

<label for="algorithm">Select an algorithm:</label>

<select id="algorithm" onchange="displayResults()">

<option value="">--Select Algorithm--</option>

<option value="random-forest">Random Forest</option>

<option value="svm">Support Vector Machine</option>

<option value="knn">K-Nearest Neighbors</option>

<option value="naive-bayes">Naive Bayes</option>

<option value="logistic-regression">logistic regression</option>

</select>

<table id="results-table">

<tr>

<th>Metric</th>

<th>Value</th>

</tr>

</table>

</div>

</div>

</div>

<div id="plots-container"></div>

<script>

const results = {

'random-forest': {

'Accuracy': 99.76,

'Precision': 99.96,

'Recall': 99.54,

'F1 Score': 99.75,

'Confusion Matrix': '[[8804 3]\n [ 39 8382]]'

},

'svm': {

'Accuracy': 98.40,

'Precision': 99.14,

'Recall': 97.57,

'F1 Score': 98.35,

'Confusion Matrix': '[[8736 71]\n [ 205 8216]]'

},

'knn': {

'Accuracy': 99.34,

'Precision': 99.55,

'Recall': 99.10,

'F1 Score': 99.32,

'Confusion Matrix': '[[8769 38]\n [ 76 8345]]'

},

'naive-bayes': {

'Accuracy': 76.07,

'Precision': 92.95,

'Recall': 54.63,

'F1 Score': 68.82,

'Confusion Matrix': '[[5133 207]\n [2267 2730]]'

},

'logistic-regression': {

'Accuracy': 93.19,

'Precision': 96.93,

'Recall': 88.88,

'F1 Score': 92.73,

'Confusion Matrix': '[[8570 237]\n [ 936 7485]]'

}

};

function displayResults() {

const algorithm = document.getElementById('algorithm').value;

const resultsTable = document.getElementById('results-table');

const plotsContainer = document.getElementById('plots-container');

resultsTable.innerHTML = '<tr><th>Metric</th><th>Value</th></tr>';

plotsContainer.innerHTML = '';

if (algorithm) {

for (const metric in results[algorithm]) {

if (metric === 'Confusion Matrix') {

resultsTable.innerHTML += `<tr><td>${metric}</td><td><pre>${results[algorithm][metric]}</pre></td></tr>`;

} else {

resultsTable.innerHTML += `<tr><td>${metric}</td><td>${results[algorithm][metric]}</td></tr>`;

}

}

const plot3 = document.createElement('div');

plot3.classList.add('plot');

plot3.innerHTML = `

<h2>Confusion matrix</h2>

<img src="${algorithm}\_plot3.png" alt="${algorithm} Plot 3">

`;

plotsContainer.appendChild(plot3);

const plot1 = document.createElement('div');

plot1.classList.add('plot');

plot1.innerHTML = `

<h2>evaluation metrics</h2>

<img src="${algorithm}\_plot1.png" alt="${algorithm} Plot 1">

`;

plotsContainer.appendChild(plot1);

const plot2 = document.createElement('div');

plot2.classList.add('plot');

plot2.innerHTML = `

<h2>3D plot</h2>

<img src="${algorithm}\_plot2.png" alt="${algorithm} Plot 2">

`;

plotsContainer.appendChild(plot2);

}

}

</script>

</body>

</html>