**INTRODUCTION**

* 1. **PROPOSED WORK**

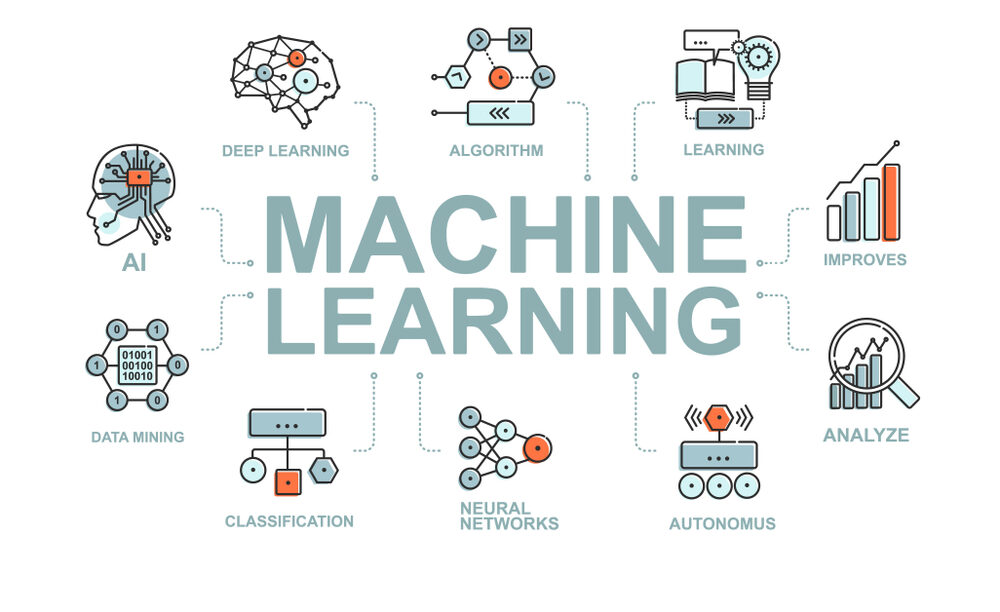
This project aims to develop a machine learning-based system for detecting and classifying malicious URLs. The system will use various models, including Logistic Regression, K-Nearest Neighbors, Naive Bayes, Random Forest, and Support Vector Classification, to identify and categorize URLs as phishing, spam, or malware. The system will be designed to detect disguised and hard-to-spot malicious URLs, which pose a significant threat to the digital world. The results of this project can contribute to improving online security and protecting users' personal and professional information from cybercrime.

* 1. **WHAT IS MACHINE LEARNING?**

Machine learning is a branch of artificial intelligence that focuses on the development of algorithms and statistical models that allow computer systems to learn and improve from experience, without being explicitly programmed. It is a data-driven approach where the computer system learns patterns and relationships from data, and uses that knowledge to make predictions or decisions on new, unseen data.

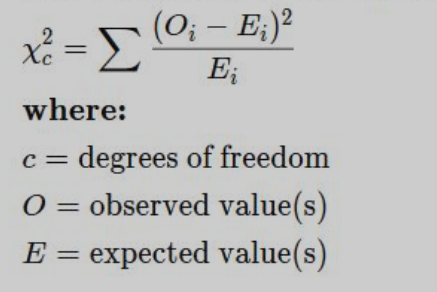
Machine learning algorithms can be classified into three categories: supervised learning, unsupervised learning, and reinforcement learning. In supervised learning, the algorithm is trained using labeled data, where the desired output is already known. In unsupervised learning, the algorithm is trained on unlabeled data, and it tries to find patterns and relationships on its own. In reinforcement learning, the algorithm learns by trial and error, and it receives feedback in the form of rewards or penalties based on its actions.

Machine learning has many applications in various industries, including healthcare, finance, retail, and marketing. It is used for tasks such as image and speech recognition, natural language processing, recommendation systems, fraud detection, and predictive analytics. Machine learning has the potential to revolutionize many fields and make our lives easier and more efficient.



* 1. **DATA ANALYSIS**
     1. **Chi square analysis:**

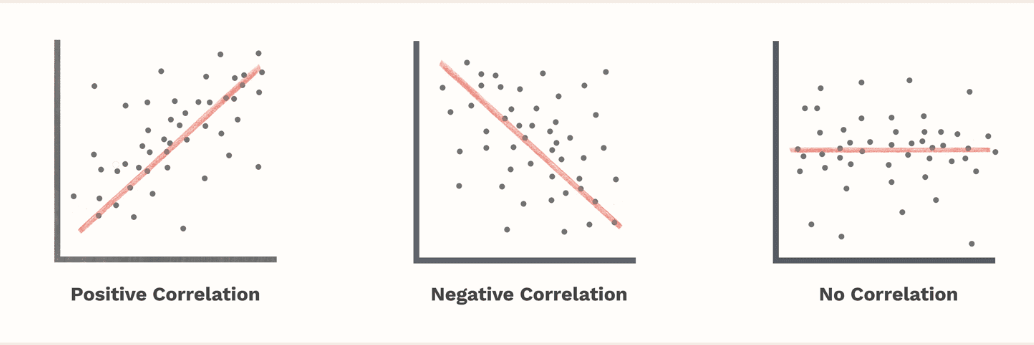
Chi-square analysis is a statistical method that determines whether there is a significant association or dependency between two categorical variables. It calculates a chi-square statistic based on the observed and expected frequency distribution of the variables. The result is a p-value, which represents the probability of obtaining the observed result by chance. If the p-value is less than the significance level, it indicates a significant association or dependency between the variables. This method is commonly used in social sciences, psychology, biology, and epidemiology to analyze categorical data and identify patterns and relationships.



* + 1. **Correlation Analysis:**

Correlation analysis is a statistical method used to measure the strength and direction of the relationship between two continuous variables. It determines how closely related the two variables are and whether they are positively or negatively related.

The correlation coefficient is a numerical value that ranges from -1 to 1 and represents the strength and direction of the relationship between the variables. A coefficient of 1 indicates a perfect positive correlation, where the two variables move in the same direction. A coefficient of -1 indicates a perfect negative correlation, where the two variables move in opposite directions. A coefficient of 0 indicates no correlation between the variables.



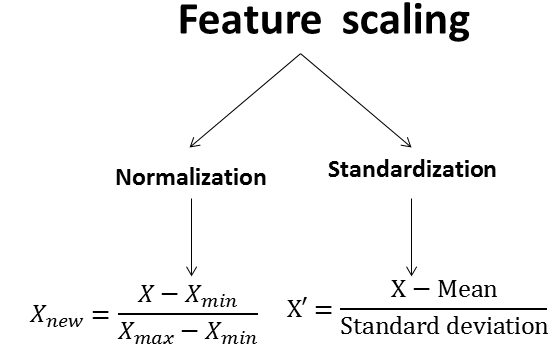
* 1. **FEATURE SCALING**

Feature scaling is a technique used in machine learning to normalize the range of values of independent variables or features of a dataset. It is done to ensure that no variable dominates the others based on their scale, and to improve the performance and accuracy of certain machine learning algorithms.

For example, if a dataset contains features with different scales, such as age and income, then the algorithm may give more weight to income than age, simply because the income values are larger. This can negatively affect the accuracy of the model.

There are several methods of feature scaling, including normalization, standardization, and scaling to unit length. Normalization scales the data so that it ranges between 0 and 1. Standardization scales the data so that it has a mean of 0 and a standard deviation of 1. Scaling to unit length scales the data so that the length of each sample vector is 1.

Overall, feature scaling is an important pre-processing step in many machine learning applications, particularly in cases where the features have different scales and units.



* 1. **MACHINE LEARNING MODELS**
     1. **Logistic Regression**

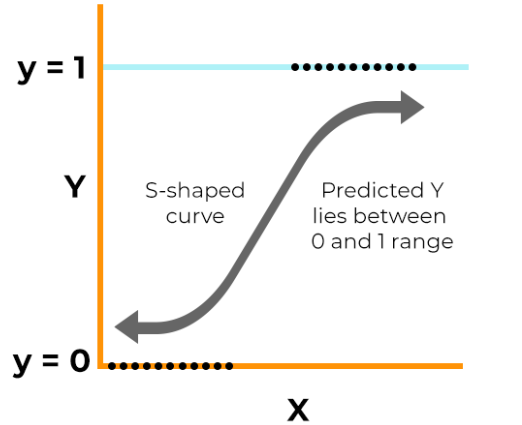
Logistic regression is a type of supervised machine learning algorithm that is used for binary classification problems, where the outcome variable is categorical and has only two possible outcomes, such as yes/no or true/false.

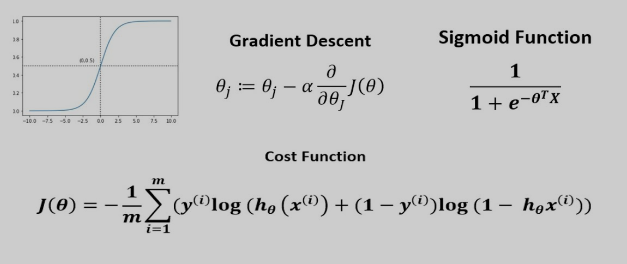
The goal of logistic regression is to predict the probability of an outcome based on one or more input variables or features. It uses a logistic function (also known as the sigmoid function) to transform the input variables into a probability value between 0

and 1, which represents the likelihood of the outcome being positive.

Logistic regression works by fitting a logistic regression model to the training data, which involves estimating the coefficients of the input variables that best fit the data. The model can then be used to make predictions on new data by applying the learned coefficients to the input variables and using the logistic function to calculate the probability of the outcome.

In summary, logistic regression is a simple yet powerful algorithm for binary classification problems that predicts the probability of an outcome based on input variables or features.





* + 1. **K Nearest Neighbors**

K-Nearest Neighbors (KNN) is a type of supervised machine learning algorithm used for classification and regression tasks. The algorithm is based on the idea that data points with similar attributes tend to be clustered together in space.

In KNN, the k nearest neighbors to a new data point are determined from the training data, based on their distance to the new data point. The distance can be calculated using various metrics such as Euclidean distance or Manhattan distance.

For classification tasks, the class label of the new data point is then determined by majority vote of the class labels of its k nearest neighbors. For regression tasks, the output value is the average of the values of the k nearest neighbors.

The choice of k is an important parameter in KNN, as it can significantly affect the performance of the algorithm. A smaller value of k can lead to overfitting and a larger value of k can lead to underfitting.

KNN is a simple yet effective algorithm that can be used for a wide range of classification and regression tasks. However, it can be computationally expensive and may not work well with high-dimensional data.



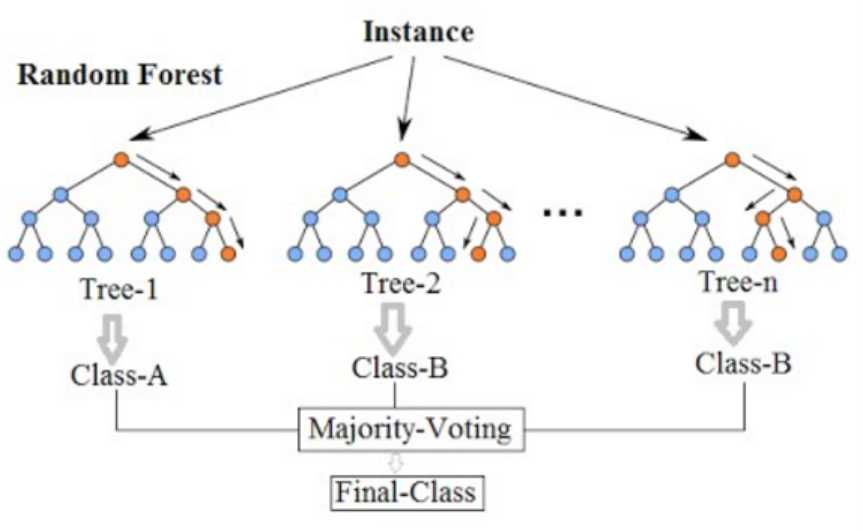
* + 1. **Random Forest**

Random forest is a type of supervised machine learning algorithm used for classification and regression tasks. It is an ensemble learning method that combines multiple decision trees to make more accurate predictions.

In random forest, multiple decision trees are constructed using a random subset of the training data and a random subset of the input variables. The decision trees are then combined to make predictions, with each tree contributing a vote to the final prediction.

The randomization of the data and variables used in each decision tree helps to reduce overfitting and increase the accuracy of the model. Additionally, random forest can handle both numerical and categorical data and can also provide information on the importance of the input variables.

Random forest is a powerful algorithm that can be used for a wide range of classification and regression tasks. However, it can be computationally expensive and may not work well with high-dimensional data.



* + 1. **Support Vector Machine**

Support Vector Machines is a type of supervised machine learning algorithm used for classification and regression tasks. The goal of SVM is to find a hyperplane that separates the data into different classes with the largest margin possible.

In SVM, the hyperplane is determined by finding the support vectors, which are the data points closest to the hyperplane. The distance between the support vectors and the hyperplane is known as the margin, and the goal of SVM is to maximize the margin.

If the data is not linearly separable, SVM can still be used by transforming the data into a higher-dimensional space using a kernel function. This allows the data to be separated by a hyperplane in the higher-dimensional space, which corresponds to a nonlinear separation in the original feature space.

SVM is a powerful algorithm that can handle both linear and nonlinear classification tasks. However, it can be computationally expensive and sensitive to the choice of hyperparameters. Additionally, SVM can be affected by the presence of outliers in the data.



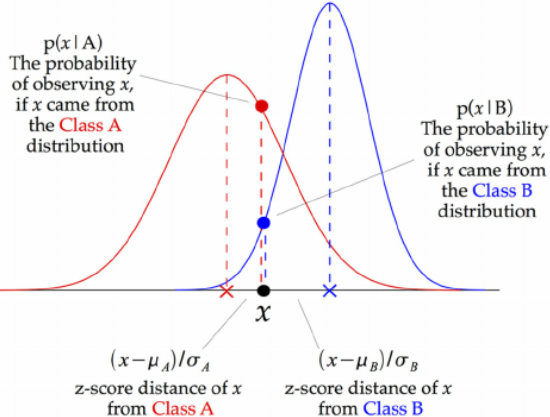
* + 1. **Gaussian Naïve Bayes**

Gaussian Naive Bayes is a type of supervised machine learning algorithm used for classification tasks. It is based on Bayes' theorem and assumes that the input variables are independent and normally distributed.

In Gaussian Naive Bayes, the algorithm first calculates the prior probability of each class based on the training data. Then, for a new data point, the algorithm calculates the conditional probability of each class given the input variables using the Gaussian probability density function.

The algorithm selects the class with the highest probability as the predicted class for the new data point.

Gaussian Naive Bayes is a simple and fast algorithm that can handle high-dimensional data and is relatively insensitive to irrelevant input variables. However, it assumes that the input variables are independent, which may not be true in practice.



* 1. **PROPOSED METHODOLOGY**

The proposed method consists of following phases 1. Feature Extraction 2. Feature reduction 3. Train the Model 4. Testing.

Feature extraction is the primary phase for machine learning techniques. UNB dataset 2016 is used for feature extraction and experiments with different machine learning algorithms.

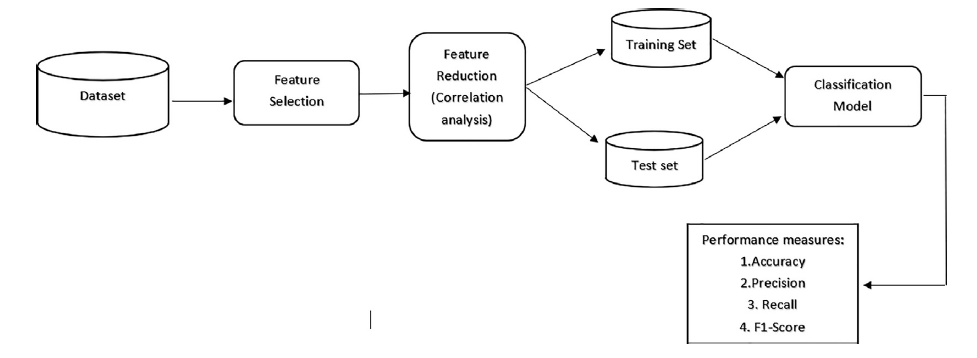
The UNB dataset includes benign, phishing, malware, spam url dataset.

Minimum set of optimal features which reduces the execution time and storage consumption. Initially identified 18 most common features and 9 newly identified features (\*) in the dataset.

All extract features are not always suitable for classification. So it requires feature reduction method to identify the fitness of the features. Correlation analysis help to find the relationship between target feature and other features which in turn help to reduce the feature. Features are removed if there is no significant correction is present.

Nearly 20 features are selected after correction analysis and 7 features are removed from features set (Underscores \_ in\_Domain, Double-slashes \_in\_URL, At(@)\_in\_URL, Hash(#) \_in\_URL, Https\_in\_URL, Numbers\_\_in\_URL,

Special\_char\_ratio\_in\_URL).

****

**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**

**import numpy as np**

**class LogisticRegression:**

**def \_\_init\_\_(self, lr=0.01, num\_iters=100000, fit\_intercept=True):**

**self.lr = lr**

**self.num\_iters = num\_iters**

**self.fit\_intercept = fit\_intercept**

**def sigmoid(self, z):**

**return 1 / (1 + np.exp(-z))**

**def add\_intercept(self, X):**

**intercept = np.ones((X.shape[0], 1))**

**return np.concatenate((intercept, X), axis=1)**

**def fit(self, X, y):**

**if self.fit\_intercept:**

**X = self.add\_intercept(X)**

**self.theta = np.zeros(X.shape[1])**

**for i in range(self.num\_iters):**

**z = np.dot(X, self.theta)**

**h = self.sigmoid(z)**

**gradient = np.dot(X.T, (h - y)) / y.size**

**self.theta -= self.lr \* gradient**

**def predict\_prob(self, X):**

**if self.fit\_intercept:**

**X = self.add\_intercept(X)**

**return self.sigmoid(np.dot(X, self.theta))**

**def predict(self, X, threshold=0.6):**

**return self.predict\_prob(X) >= threshold**

**lr = LogisticRegression()**

**lr.fit(X\_train, y\_train)**

**y\_predLR = lr.predict(X\_test)**

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**

**import numpy as np**

**import heapq**

**class KNNClassifier:**

**def \_init\_(self):**

**self.k = 5**

**def fit(self, X, y):**

**self.X\_train = X**

**self.y\_train = y**

**def predict(self, X\_test):**

**y\_pred = []**

**for x in X\_test:**

**distances = np.sqrt(np.sum(np.square(self.X\_train - x), axis=1))**

**k\_nearest\_heap = []**

**for i in range(len(distances)):**

**if len(k\_nearest\_heap) < 5:**

**heapq.heappush(k\_nearest\_heap, (-distances[i], self.y\_train[i]))**

**else:**

**if -distances[i] > k\_nearest\_heap[0][0]:**

**heapq.heappop(k\_nearest\_heap)**

**heapq.heappush(k\_nearest\_heap, (-distances[i], self.y\_train[i]))**

**labels = [label for \_, label in k\_nearest\_heap]**

**y\_pred.append(max(set(labels), key=labels.count))**

**return y\_pred**

**knn = KNNClassifier()**

**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()

Plots For Comparison

x\_values = [2, 4, 6, 8, 10]

y\_values = [l1, r1, k1, s1, n1]

x\_labels = ['LR', 'Random Forest', 'KNN', 'SVM','Naive Bayes']

plt.xticks(x\_values, x\_labels)

plt.plot(x\_values, y\_values, '-o')

plt.title('Accuracy')

plt.show()

x\_values = [2, 4, 6, 8, 10]

y\_values = [l2, r2, k2, s2, n2]

x\_labels = ['LR', 'Random Forest', 'KNN', 'SVM','Naive Bayes']

plt.xticks(x\_values, x\_labels)

plt.plot(x\_values, y\_values, '-o')

plt.title('Precision')

plt.show()

x\_values = [2, 4, 6, 8, 10]

y\_values = [l3, r3, k3, s3, n3]

x\_labels = ['LR', 'Random Forest', 'KNN', 'SVM','Naive Bayes']

plt.xticks(x\_values, x\_labels)

plt.plot(x\_values, y\_values, '-o')

plt.title('Recall')

plt.show()

x\_values = [2, 4, 6, 8, 10]

y\_values = [l4, r4, k4, s4, n4]

x\_labels = ['LR', 'Random Forest', 'KNN', 'SVM','Naive Bayes']

plt.xticks(x\_values, x\_labels)

plt.plot(x\_values, y\_values, '-o')

plt.title('F1-Score')

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);

}

}

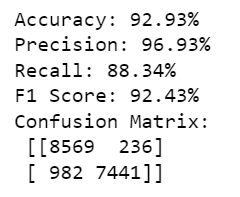
</script>

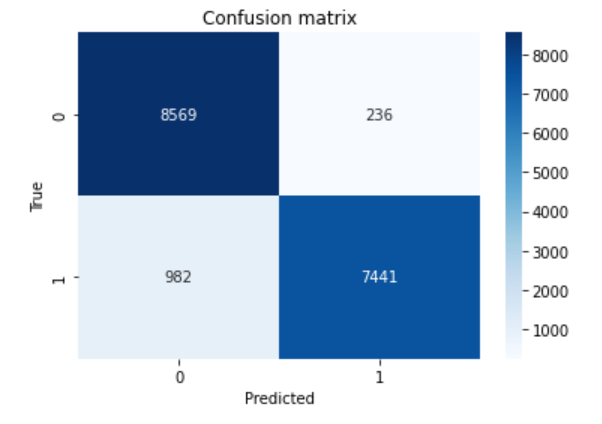
</body>

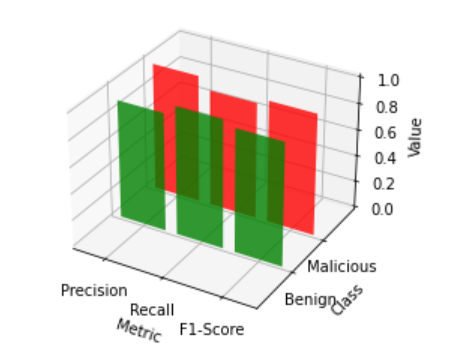
</html>

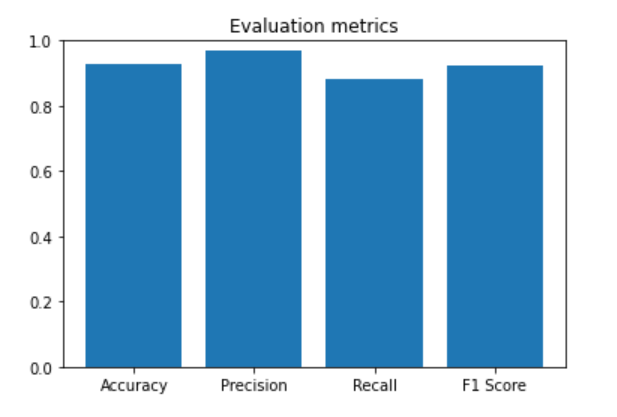
**RESULTS :**

**i) Logistic Regression :**

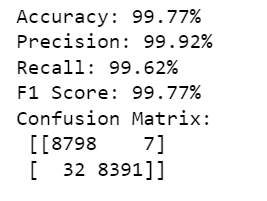


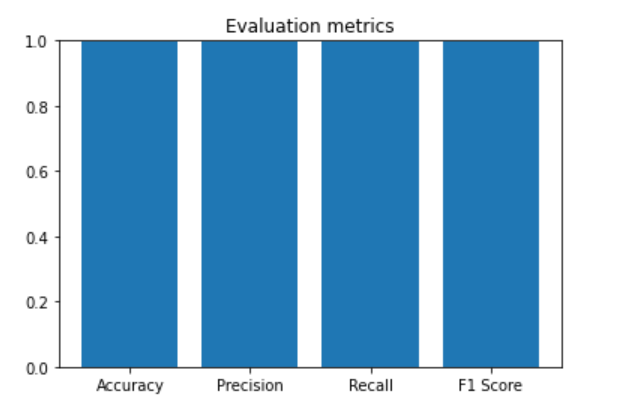
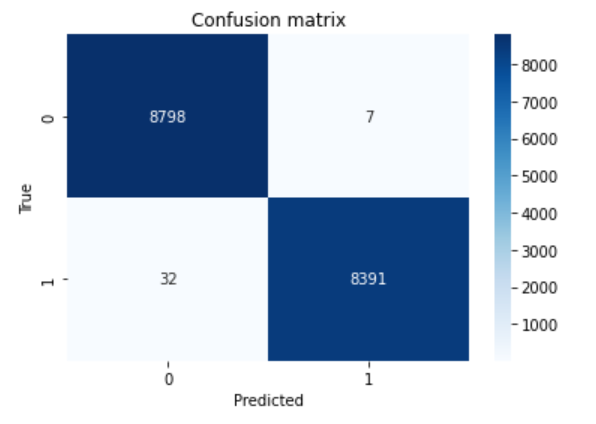
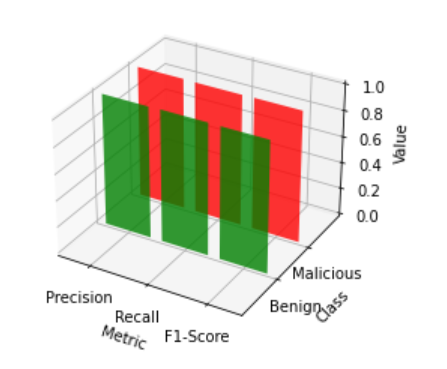




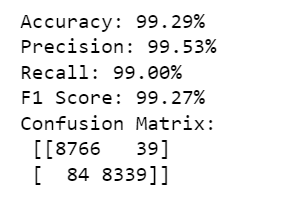


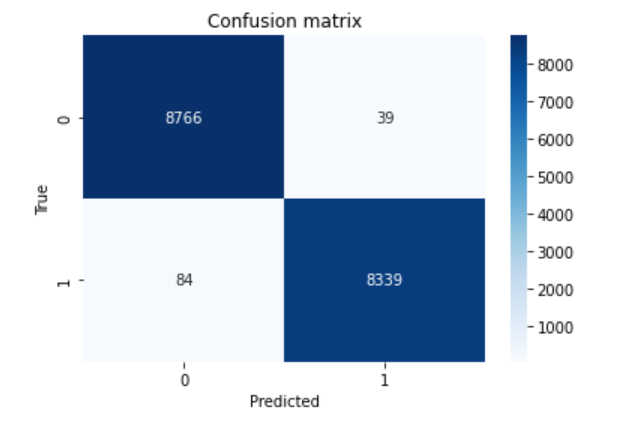
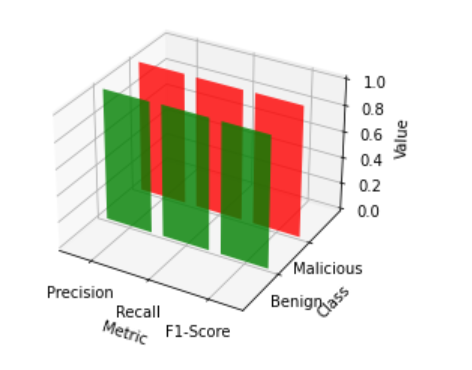
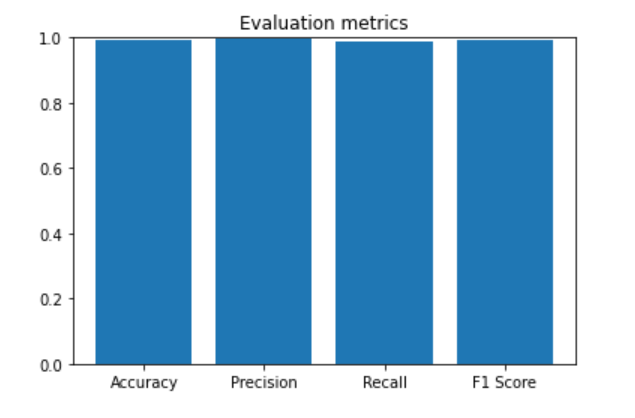
**ii) Random Forest :**



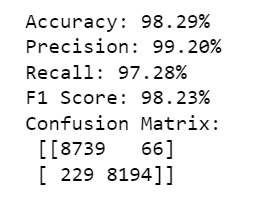


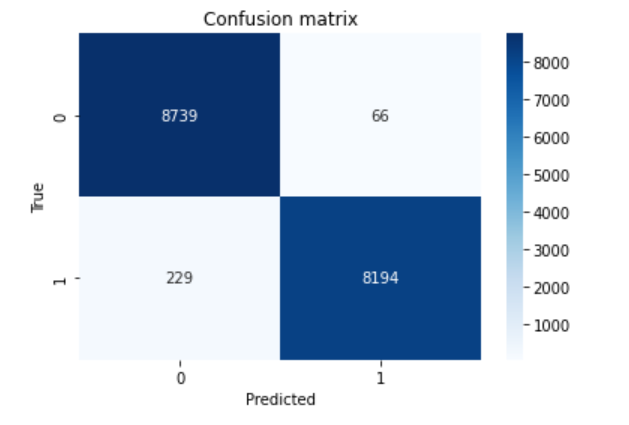
**iii) K - nearest neighbors :**

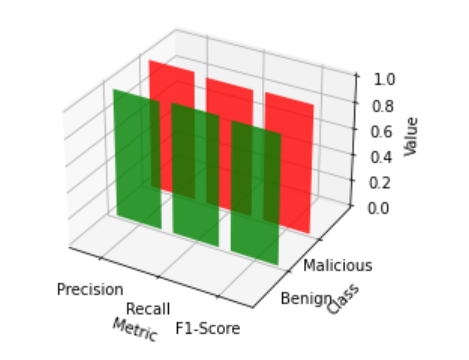
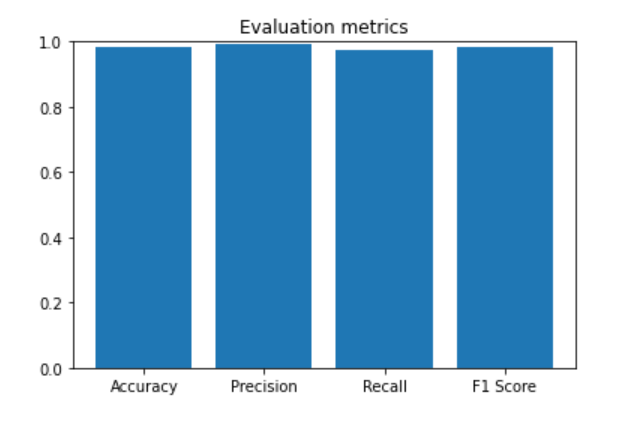




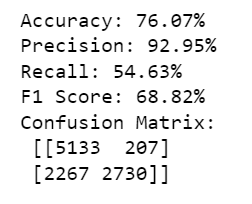
**iv) Support Vector Machine :**

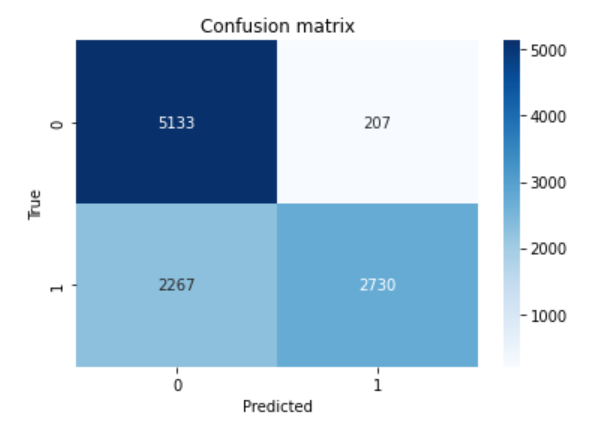
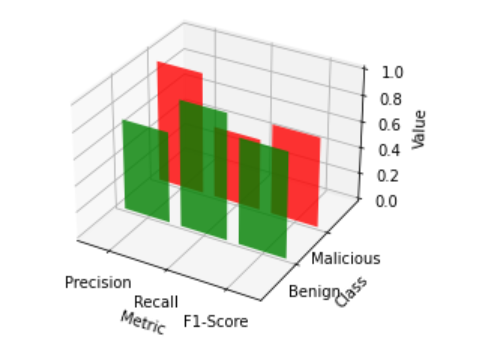
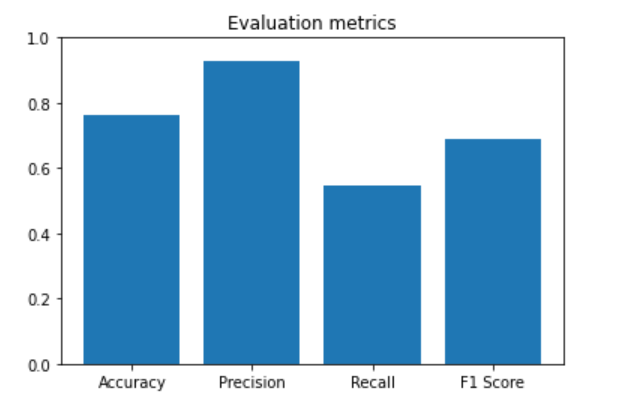






**v) Gaussian Naïve Bayes :**





**vi) Comparison Plots :**

