LONDON METROPOLITAN UNIVERSITY

Islington college
(इस्लिङ्टन कलेज)

Module Code & Module Title

CU6051NI - Artificial Intelligence

Assessment Weightage & Type

75% Individual Coursework

Year and Semester

2024 Spring

Student Name: Sanskar Kafle

London Met ID: 22085609

College ID: NP01CP4S230092

Assignment Due Date:

Assignment Submission Date:

Word Count: 5085

I confirm that I understand my coursework needs to be submitted online via Google Classroom under the relevant module page before the deadline in order for my assignment to be accepted and marked. I am fully aware that late submissions will be treated as non-submission and a marks of zero will be awarded.

Abstract

This project responds to the increasing risk of cyber threats due to malicious URLs including phishing, scams and other unsafe websites. To reduce these threats, a strong detection system was established through employing the different algorithms such as SVM, KNN and Random Forest. The essential ideas used in the development of the system involve an array of more than 14,000 URLs from which features are analysed with the objective of isolating the URLs as either genuine or malicious. With the help of an advanced analytical toolset, the capacity to identify and avoid cyber threats is increased to a proactive level. Built to function as a continuation of users' internal frameworks, the proposed solution enhances protection against data leaks and financial scams as well as identity theft. Thus, by detecting threats in real-mode and guaranteeing strong protection it participates in the formation of the secure environment in the World Wide Web for users. This project also highlights the need to use machine learning considering the dynamics that face the cybersecurity field.

Table of Contents

List of Figures

List of Tables

1. Introduction

1.1 Explanation of the AI concepts

1.1.1 Artificial Intelligence

Imagine a world where your closest confidant isn't a person, but an intelligent machine that understands you better than anyone else. Artificial Intelligence research, which began in the 1950s and 1960s, was driven by the ambition to replicate human strongest abilities. The concept of AI companionship is not new, but its realization in practical applications is a relatively recent development but systems during that era was distinguished by the concentration on imitating human thinking, where the brain can derive deductions from given information. (Sahota, 2024)

Manufactured insights is the process of making machines think like people, but machine learning, which is a part of AI, is largely about getting computers to carry out tasks by recognizing patterns and differences. While AI is programmed to follow a set of predetermined algorithms, machine learning needs only the clear programming to locate those big data, extract patterns and provide suggestions. This resulting machine learning models improve in accuracy as they are trained on more and higher-quality data, reflecting the knowledge acquired from the training process. (Google Cloud, 2024)

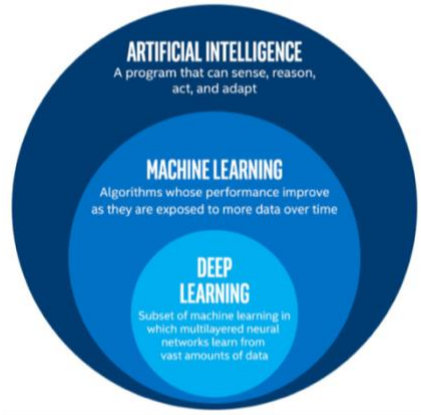


Figure 1: AI, ML and DL (i2tutorials, 2019)

1.1.2 Machine Learning

Machine Learning is a subfield of Artificial Intelligence (AI) that focuses on the development of computer algorithms that improve automatically through experience and using data. In simpler terms, machine learning enables computers to learn from data and make decisions or predictions without being explicitly programmed to do so. (Crabtree, 2024)

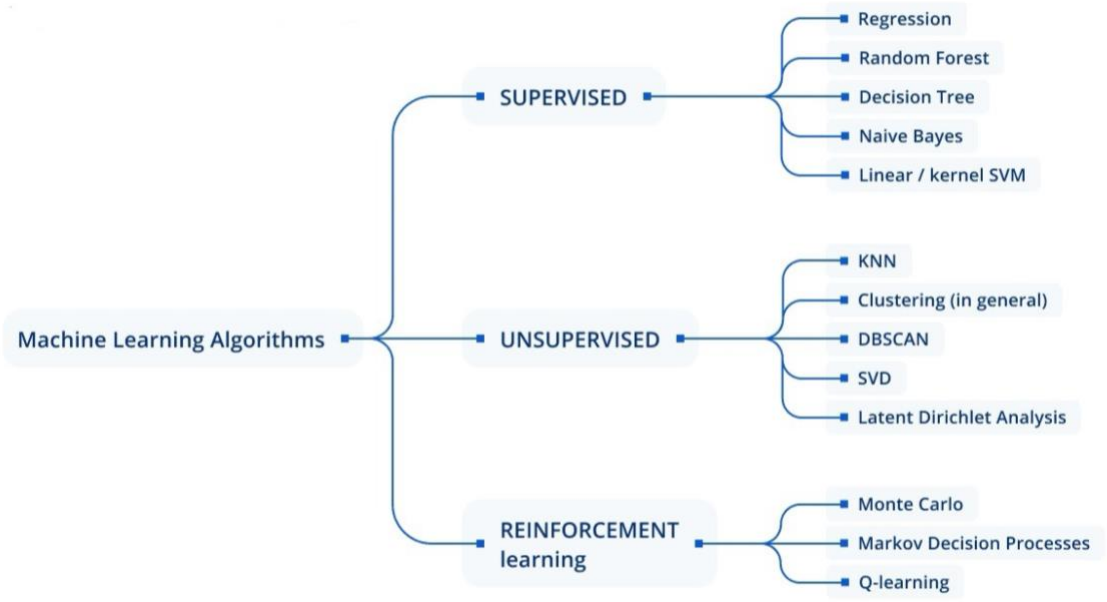


Figure 2: Types of ML algorithms (mobidev, 2024)

(A more descriptive detail about machine learning can be found in the appendix: Machine Learning)

1.1.3 Classification

Classification is a fundamental task in machine learning modelling process in which an algorithm is trained on a dataset consisting of labelled samples, each of which belongs to a given predefined groups called classes. The goal is to enable the algorithm to recognize patterns and relationships within the data, allowing it to accurately classify new, unseen instances into predefined categories (Belcic, 2024). For instance, an algorithm can learn to predict whether a given email is spam or ham (no spam), as illustrated below.

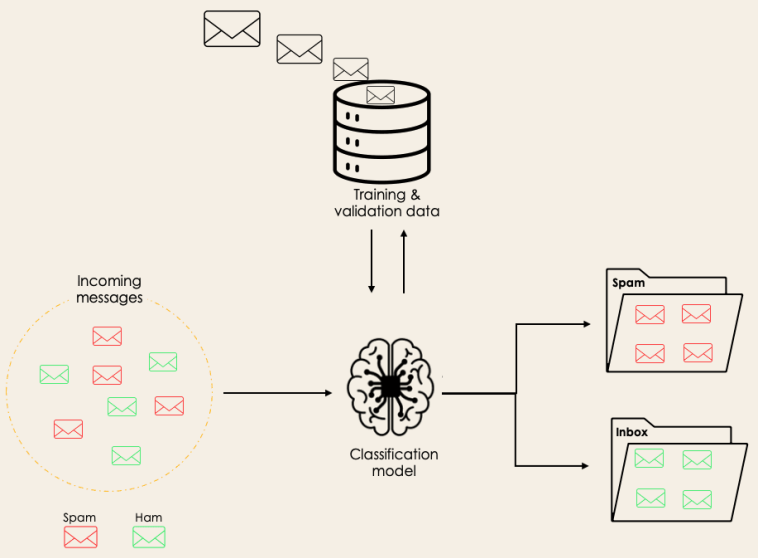


Figure 3: Example of Classification model (datacamp, 2024)

Listed below are some of the classification algorithms:

- Logistic regression
- Decision trees
- Random Forest
- Support Vector Machines (SVMs)
- K-nearest Neighbours (KNN)
- Naive Bayes

1.1.4 Binary Classification

When using machine learning, binary classification is the main supervised learning technology that classifies new observations into one of two classes. It forecasts a binary outcome, which can be either positive or negative. For example, medical diagnosis identifying if a patient has a specific disease (e.g., cancer) or not. The following are a few binary classification applications where the 0 and 1 columns are two possible classes for each observation. (Karabiber, 2024)

Application	Observation	0	1
Medical Diagnosis	Patient	Healthy	Diseased
Email Analysis	Email	Not Spam	Spam
Financial Data Analysis	Transaction	Not Fraud	Fraud
Marketing	Website visitor	Won't Buy	Will Buy
Image Classification	Image	Hotdog	Not Hotdog

Figure 4: Binary classification applications (Karabiber, 2024)

1.2 Problem Domain

The rapid development of digital platforms in this era has made it more and more difficult to separate quiet from potentially dangerous ones. More and more websites are made every day, and it has become popular to collect user data. Nonetheless, not being able to recognize and classify harmful URLs) in a fast manner, this growth represents a critical security weakness (Blome, 2024). Along with the risks of using phishing emails to lure victims to open the links, spammers may distribute spam emails, which will take over the user's computer and use it for sending more outwards, means of spreading such malware have increased significantly in recent years. The security requirement that phishing and attachment are disabled or encrypted should become a mandatory rule for enterprises, so if users engage in these behaviours, violations are automatically detected.

As of the second quarter of 2024, average daily time spent using the internet by online users worldwide is six hours and 36 minutes daily and it continues to grow (Petrosyan, 2024). Although the Internet is widely used, many users lack the necessary skills and knowledge to protect themselves from cyber security threats because malicious URLs are so hidden and easy to sneak into normal browsing activity.

In the third quarter of 2024, the number of unique phishing sites worldwide reached over 932,000, slightly more than in the previous quarter. The figure for unique phishing sites was particularly high between the second and third quarters of 2020, rising from nearly 147,000 to about 572,000. This representation is based on the unique base URLs of the phishing sites. (statista, 2024)

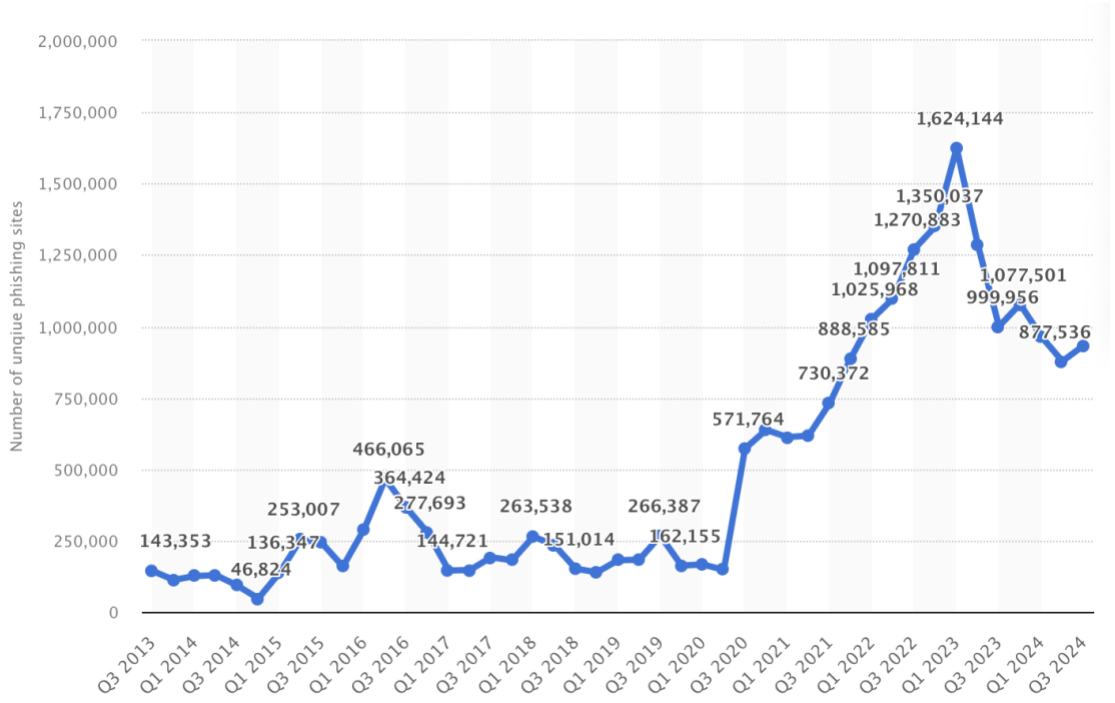


Figure 5 : Unique phishing sites detected (statista, 2024)

Visiting certain websites can pose serious risks, including malware attacks and personal data breaches, highlighting the urgent need for robust cybersecurity strategies. This project focuses on creating a sophisticated system to detect and classify malicious URLs through the application of advanced machine learning techniques. By strengthening defences against dynamic cyber threats, the initiative aims to reduce vulnerabilities and ensure a safer online experience for users, effectively addressing the dangers associated with harmful URLs.

2. Background

2.1 Research done on the chosen topic/problem domain

2.1.1 URL Phishing Detection

URL phishing is a common scam where attackers send fake emails to trick people into visiting fake websites. These emails often look like they come from trusted organizations, such as banks or online services, to make them believable. When someone clicks on the link, they are taken to a website that looks almost identical to the real one. Once they have this data, they can access accounts like sensitive information, email, social media, passwords, card details or online banking credentials. Every day, about 15 billion spam emails are sent, and more than 80% of businesses have faced phishing attacks at some point. Despite increased awareness, at least one-third of all phishing emails are opened, and in about 90% of data breaches, phishing is the root cause. (Fortinet, 2024)

To detect whether a URL is a fake, the users should have the knowledge about URL format. Each URL consists of the components illustrated in the diagram below.

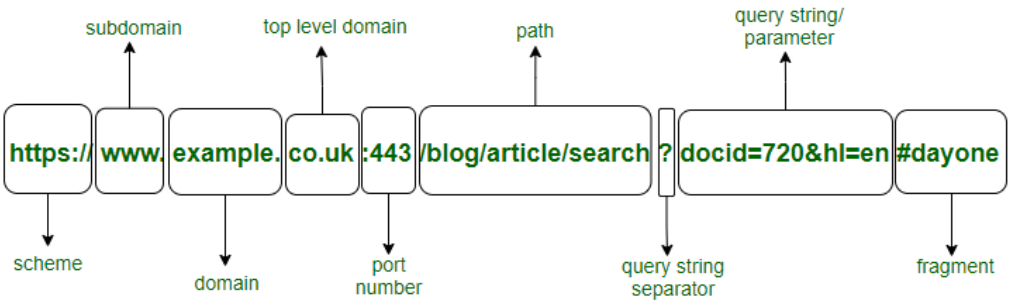


Figure 6: Parts of a URL (geeksforgeeks, 2021)

(This section of the report is continued in the appendix: URL Phishing Detection)

Ability to detect fake web URLs is an essential aspect of the fight against phishing as well as other cyber-attacks. To stay competent in identifying and navigating to those URLs that can possibly be the source of danger, the users should concentrate on details such as domain legality, spotting what may look like a small mistake, as well as affirming the security protocols.

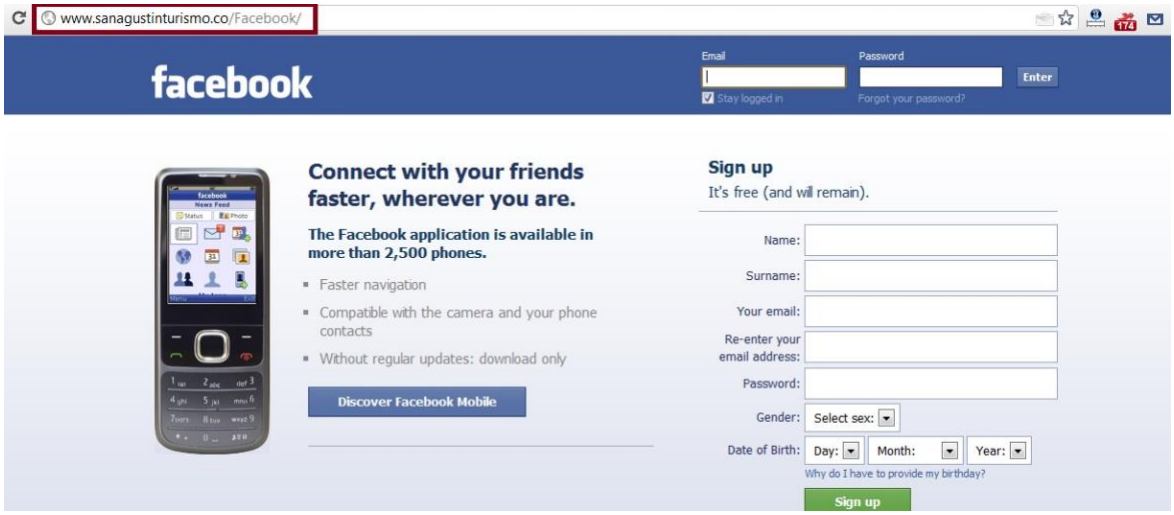


Figure 7: Example of phishing websites 1 (Büber, 2025)

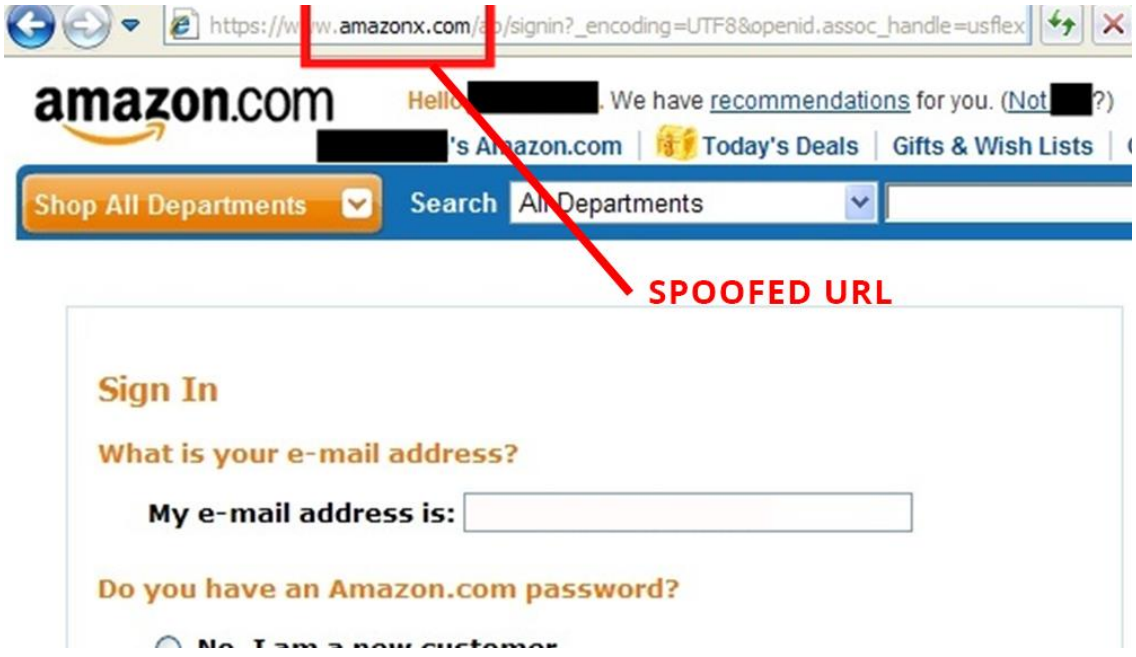


Figure 8: Example of phishing websites 2 (purecloudsolutions, 2024)

2.1.2 Problems of URL Detection in current scenario

- **Phishing Attacks on the Rise**

According to data from the Cybercrime Information Center, there was an 85% increase in unique domains reported for phishing, and 77% of these domains were specifically registered by phishers for phishing purposes. These reports linked to over 500,000 phishing attacks a significant 37% rise over the prior quarter. (Interisle Consulting Group, 2024)

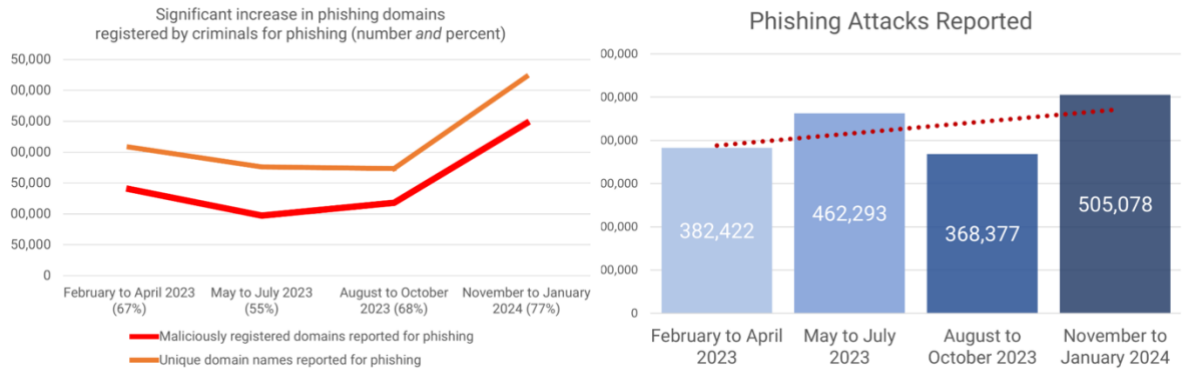


Figure 9: Unique domains reported for phishing (Interisle Consulting Group, 2024)

(This section is continued in the appendix: Problems of URL Detection in current scenario)

2.1.3 Machine Learning / AI in URL Detection

Fake URLs pose serious threats by hosting harmful content such as spam, phishing schemes, and malware. They trick users into falling victim to scams, leading to financial losses, identity fraud, and system compromises are collectively causing billions of dollars in damages annually. Rapid detection and reduction of such threats are crucial. Traditionally, blacklist-based methods have been the primary approach for detecting malicious URLs. However, these methods have limitations and often struggle to accurately detect newly created or rapidly evolving threats.

In recent days, machine learning / AI has gained momentum as a solution of enhancing malicious URL detection. This approach involves training machine learning models on datasets containing URLs, enabling the models to tell the difference between malicious and genuine URLs based on statistical features. The process begins with extracting valid attributes from URLs, which are then transformed into numerical representations suitable for model training. The effectiveness of this method depends on the quality and relevance of the extracted features, which directly influence the accuracy of predictions. (Abad, 2023)

Several websites mentioned below have implemented some sort of ML/AI to detect malicious URLs.

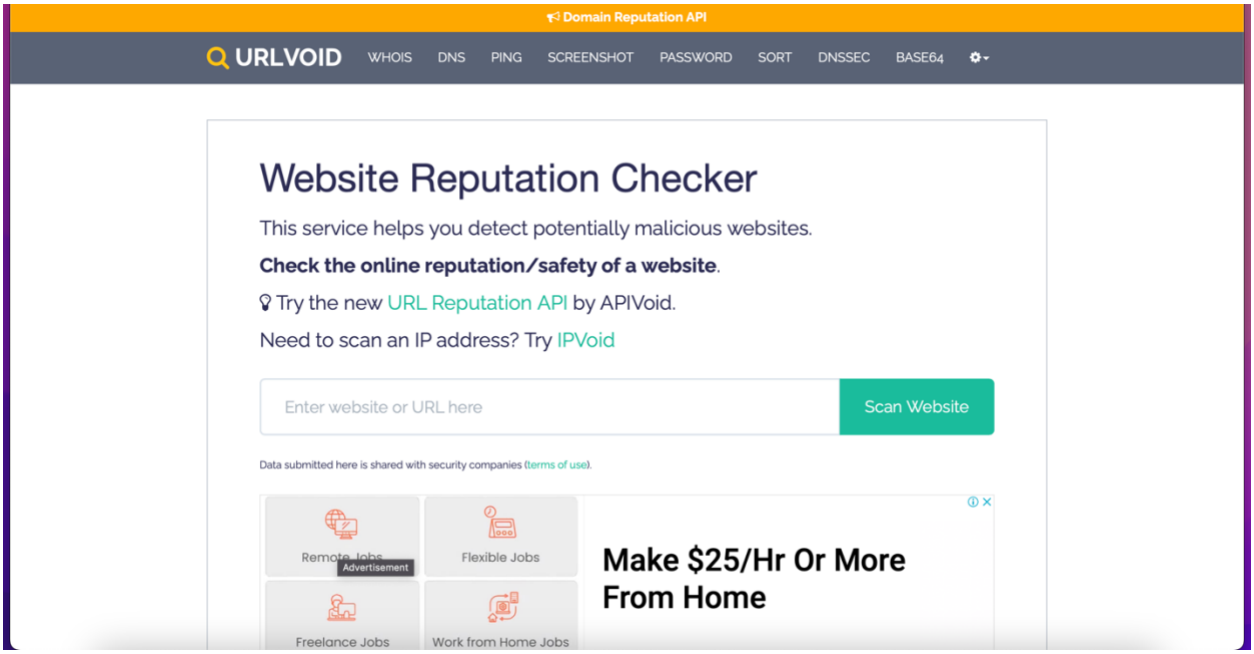


Figure 10: Screenshot of urlvoid

Figure 11: Screenshot of virustotal

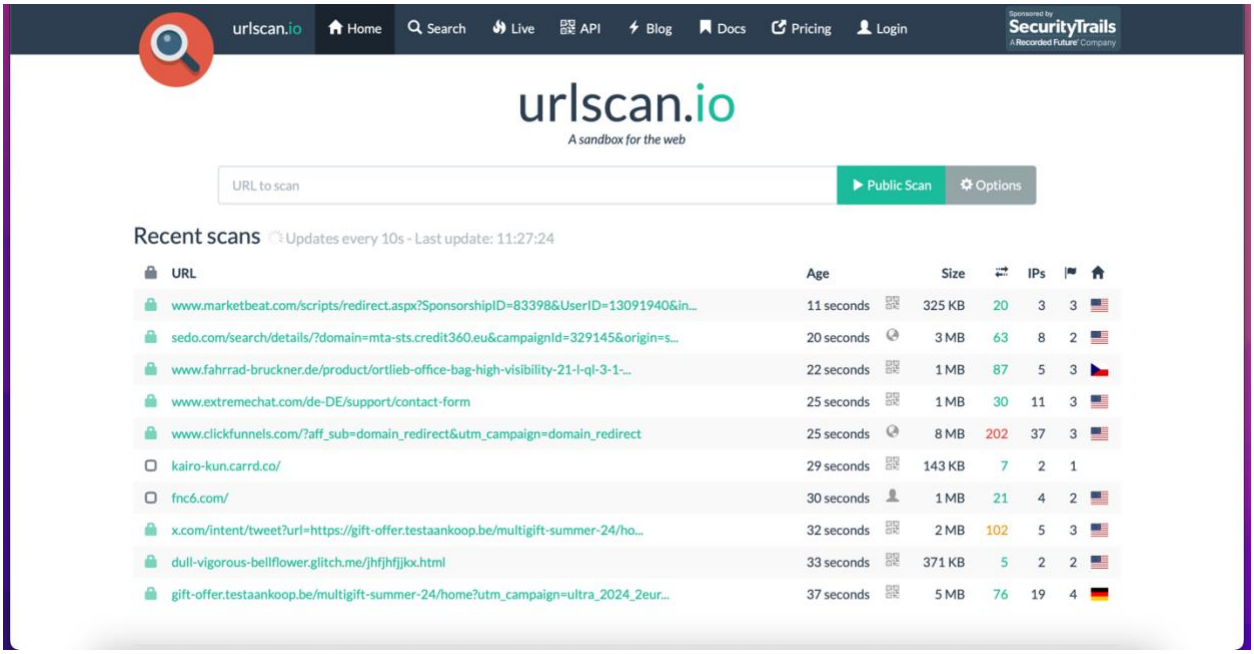


Figure 12: Screenshot of urlscan.io

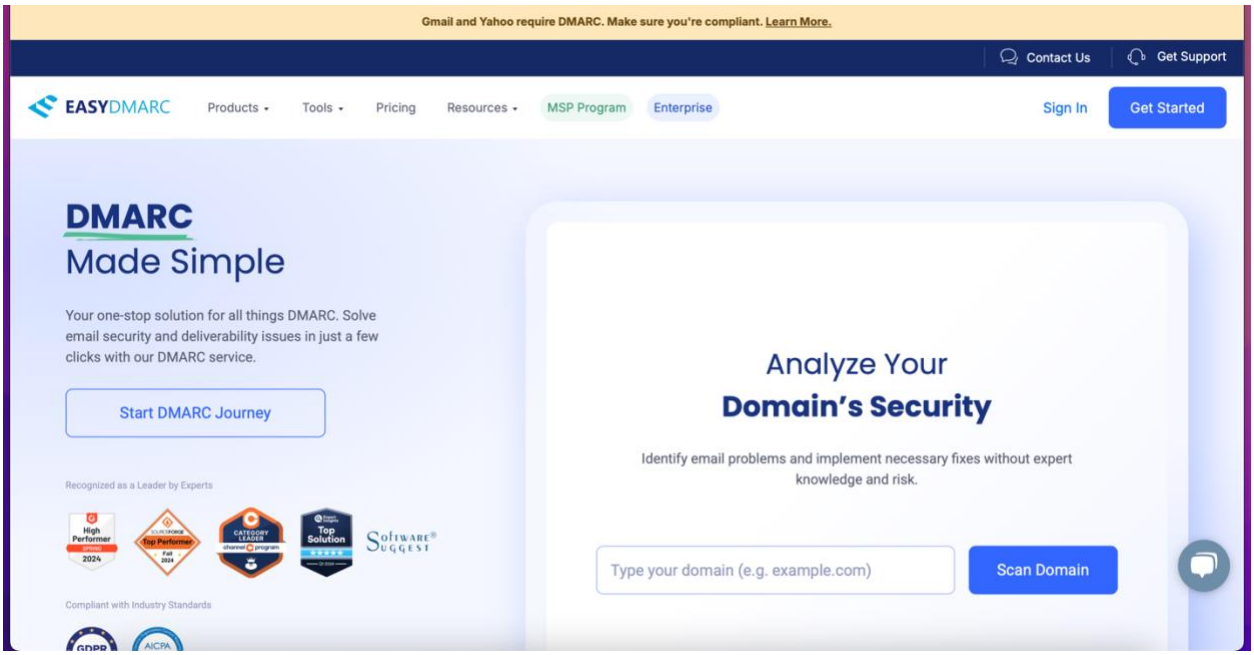


Figure 13: Screenshot of EasyDMARC

2.1.4 Datasets

Machine learning (ML) datasets are collections of data used to train, test, and evaluate models, enabling programmers to understand algorithms and implement practical predictions. While some datasets are freely available online for public use, others are tailored to specific project requirements. In the monarchy of machine learning, data is the fuel that powers innovation. The quality and quantity of data directly influence the performance and capabilities of machine learning models. (geeksforgeeks, 2024)

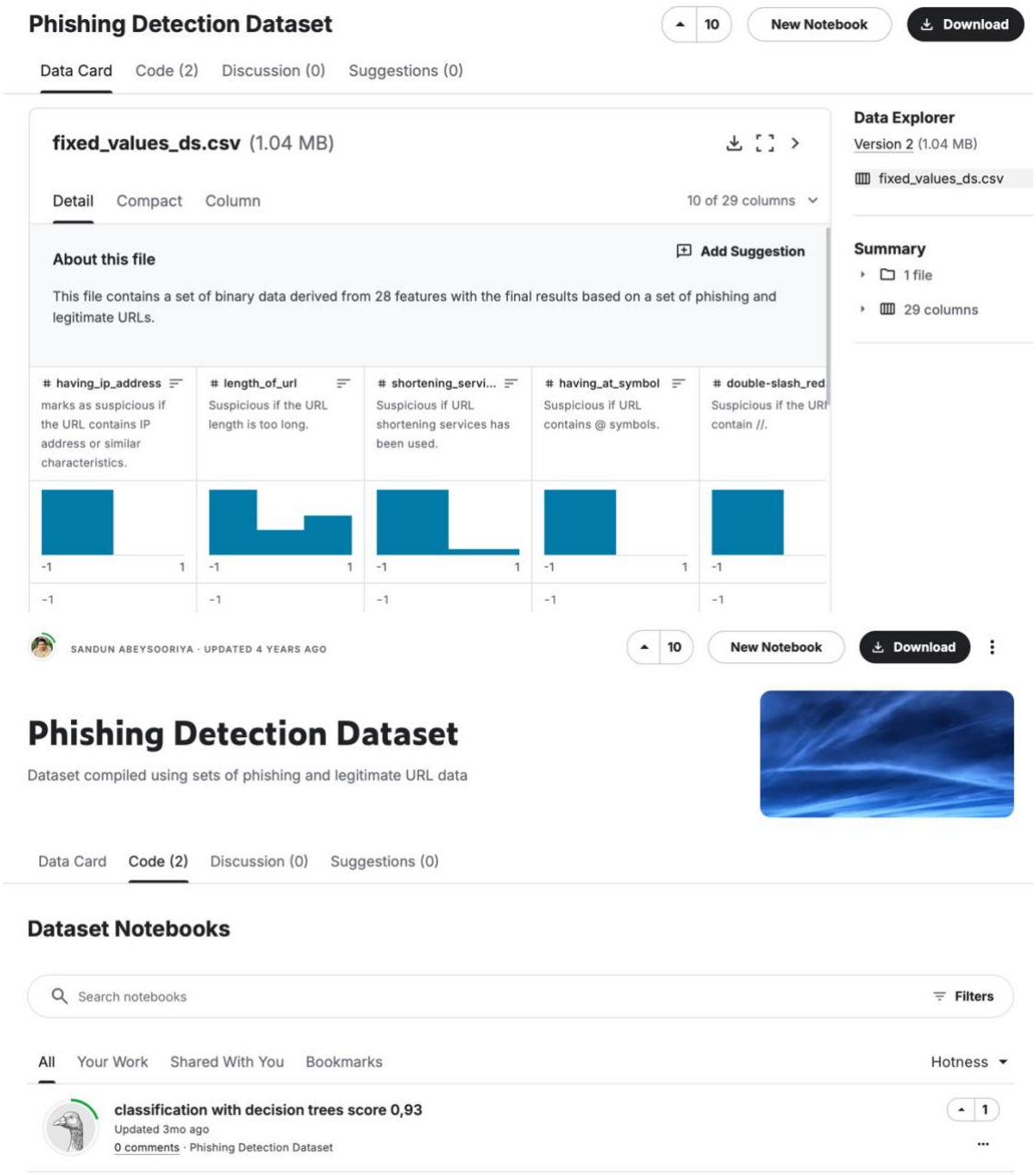


Figure 14: Dataset to be used for Phishing URL Detection (Anuganti, 2020)

Table 1: Dataset attributes description

Column Name	Column Description
having_ip_address	Checks if the URL contains IP address or similar characteristics.
length_of_url	Checks if the URL length is too long.
shortening_services	Checks if the URL shortening service has been used.
having_at_symbol	Checks if URL contains symbols.
double-slash_redirection	Checks if the URL contains double slash.
prefix and suffix	Checks if the URL contains prefixes and suffixes.
sub_domains	Checks if the URL contains many subdomains.
ssl_state	Checks if the URL is related to SSL state.
domain_registered	Checks if the URL is related to domain registration.
favicons	Checks if the URL is related to favicons.
ports	Checks if the URL is related to ports.
https	Checks if the URL contains HTTPS.
external_objects	Checks if the URL is related to website elements.
anchor_tags	Checks if the URL contains anchor tags.
links_in_tags	Checks if the URL is related to links in html tags.

sfh-domain	Checks if the URL is related to SFH domain.
auto_email	Checks if the URL auto sends emails.
abnormal_url	Checks if the URL contains abnormal characteristics.
iframe_redirection	Checks if the URL is redirected to IFrame.
on_mouse_over	Checks if the URL is related to onMouseOver event.
right_click	Checks if the URL is related to rightClick event.
popup_windows	Checks if the URL executes to pop up windows.
domain_age	Checks the URL domain age.
dns_record	Checks the URL DNS record.
web_traffic	Checks the URL website traffic.
links_pointing	Checks if the URL links pointing to the web page.
statistical_report	Checks if the URL statistical report.
image_text_keyword	Checks if the URL website images contain phishing keywords.
result	Phishing (1), Legitimate (-1).

(The reason for choosing this dataset is described in the appendix section: Dataset)

2.2 Review and Analysis of existing work in the problem domain

2.2.1 Model of detection of phishing URLs based on machine learning

Author: Vishesh Bharuka, Allan Almeida, Sharvari Patil

Journal: International Journal of Scientific Research in Computer Science, Engineering and Information Technology, Volume 10, Issue 2

Issued Date: March 2024

This research paper focuses on the use of ML techniques for the detection of phishing websites which is a major security threat. It uses datasets with URLs that are basically phishing URLs from site like Phishtank and legitimate URL from reliable sources. The study also underlines the issue that poses the identification of phishing sites as a difficult problem, as the sites constantly transform, mimic other genuine ones. Thus, the research proposes to increase the accuracy and effectiveness of the differentiated phishing classification by training different machine learning models on these datasets. Such findings show that machine learning can properly classify URLs between legitimate and phishing, thus helping to advance the way in designing better means to prevent cyber threats. (Vishesh Bharuka, 2024)

Algorithms used in the study:

Decision Tree**,** Random Forest**,** Multilayer Perceptron**,** XGBoost**,** Autoencoder Neural Network**,** Support Vector Machines

Evaluation metrics used:

Accuracy, Precision, Recall, F1 Score

Accuracy Result

The Random Forest and XGBoost algorithms performed the best, achieving an accuracy of 98%, making them the most reliable models for phishing detection in this study.

2.2.2 Phishing URL detection with neural networks: an empirical study

Author: Hayk Ghalechyan, Elina Israyelyan, Avag Arakelyan, Gerasim Hovhannisyan & Arman Davtyan

Article Number: 25134

Issued Date: October 2024

The article proposes a study whose aim is the application of machine learning techniques to predict the existence of outcomes in the general case of a complex system. The authors train a variety of models on the data trying to find patterns based on which they can determine the accuracy. The results show that the system can significantly improve predictive power over traditional methods using machine learning methods based upon a large dataset. This work highlights the real-world application potential of these models, as well as the robustness of the chosen methods and the increased prediction performance compared to existing methods. Moreover, the research emphases the necessity of modelling fine-tuned and suitable features to maximize prediction correctness.(Hayk Ghalechyan, 2023)

Algorithms used in the study:

Decision Tree**,** SVM, Random Forest, GBM

Evaluation metrics used:

Accuracy, Precision and Recall, F1 Score, MSE, AUC, Cross-entropy

Accuracy Result

The study reported that the best-performing machine learning model achieved an accuracy of 92.5% on the test dataset, significantly outperforming traditional prediction models.

2.2.3 Life-long phishing attack detection using continual learning

Author: Asif Ejaz, Adnan Noor Mian & Sanaullah Manzoor
Article Number: 11488
Issued Date: 2023

This study focuses on the use of machine learning techniques in predicting disease outcomes within the healthcare realm. The creators aim to correct the existing situation of early diagnosis and patient management by studying a dataset consisting of electronic patient medical records, laboratory results, and imaging. By bringing into play different machine learning models, they point out their ability to get close accuracy by predicting patient's future health and the course of the disease. The survey concluded that machine learning is valuable in the healthcare pathway since it can effectively connect with physicians and clinicians, making the decision-making process more informed. The investigation further looks at the significance of feature selection as well as model interpretability, so the results are both accurate and explainable for the clinical practice.

Algorithms used in the study:

Logistic Regression, SVM, Random Forest, KNN, Gradient Boosting, XGBoost, DNN

Evaluation metrics used:

Accuracy, Precision and Recall, F1 Score, AUC, Mean Squared Error (MSE), Cross-entropy

Accuracy Result

The study reported that the best-performing machine learning model, using Random Forest, achieved an accuracy of 89.7% on the validation dataset. When deep learning methods, such as Deep Neural Networks, were applied, the accuracy increased to 92.3%.

2.2.4 Summarized Review and Analysis of Researched Journals

Based on the research from the journal & articles, it can be concluded that the three papers tackle the important issue of malicious Phishing URL detection and explore different machine learning methods to improve cybersecurity and enhance the security of their users.

The first paper explores how deep learning with attention mechanisms can improve phishing URL detection using a Long Short-Term Memory (LSTM) model with an attention layer. This method focuses on the most relevant parts of the URL, enhancing the accuracy of identifying phishing websites in real-time.

The second paper compares machine learning algorithms, including Logistic Regression, Decision Trees, and Support Vector Machines (SVM), to classify phishing URLs. It evaluates various URL features and determines which algorithms are most effective for detecting phishing sites.

The third paper uses Graph Neural Networks (GNNs) to detect phishing URLs in real-time. GNNs model URLs and their relationships as a graph, improving the accuracy and speed of phishing detection by capturing contextual information.

In summary, these studies offer different methods to improve phishing URL detection and enhance cybersecurity using machine learning. Each approach provides valuable solutions for better, faster detection of phishing websites.

3. Solution

3.1 Proposed Approach to solving the problem

The solution for this problem is by using a machine learning algorithm that has highest accuracy, such as Support Vector Machine (SVM) Algorithm, k-nearest neighbors (KNN) algorithm and Random Forest algorithm. For this project, libraries like NumPy and Pandas for data analysis, Matplotlib and Seaborn for visualizing the data, and scikit-learn for performing the machine learning tasks. The dataset has been selected form Kaggle and contains URLs from over 14,000 websites, each with 28 features and label that indicates whether a website is phishing (1) or legitimate (-1).

3.1.1 K-Nearest Neighbours (KNN) Algorithm

The k-nearest neighbors (KNN) approach is an old and famous classification and prediction machine learning method that has gained widespread use since its invention. It is a simple and powerful nonparametric method that uses the k-nearest neighbors (the true data points) to define the class of data points that do not exist on the data plane. The KNN technique is flexible because it does not require the data to have a structure, so it is relatively easy for the user to understand. (IBM, 2024)

$$d(p,q) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2}$$

were,

p, q = two points in Euclidean n-space

q_i, p_i= Euclidean vectors, starting from the origin of the space

n = n-space

3.1.2 Support Vector Machine (SVM) Algorithm

SVM is a powerful algorithm to be used in the case of small data on which it is easy to get the top score but for more difficult data, support vector machines are usually built as an SVM model. (Anshul, 2024). There are two types of SVM Algorithms Linear SVM and Non-Linear SVM. This can be used for Face detection, image classification, text categorization, etc. The below diagram in which there are two different categories that are classified using a decision boundary or hyperplane.

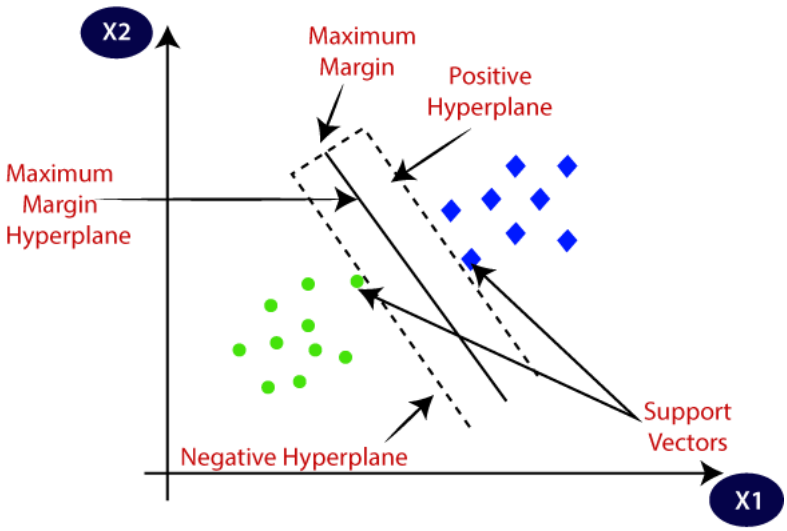


Figure 15: Classified using a decision boundary (Better Future, 2024)

3.1.3 Random Forest Algorithm

The Random Forest approach for classification consists of many decision trees that are generated from distinct subsets of the given dataset. It gathers the predictions made by each tree to improve predictive accuracy. The random forest uses majority votes from forecasts rather than just one decision tree to decide the outcome.

3.2 Evaluation Metrics

3.2.1 Confusion Matrix

A tabular overview of the number of accurate and inaccurate predictions a classifier generated is called a confusion matrix. It serves to assess a classification model's effectiveness. It may be used to calculate performance measures such as accuracy, precision, recall, and F1-score to assess how well a classification model is doing. (Anuganti, 2020)

3.2.2 Classification Metrics

Classification metrics are utilized to appraise the effectiveness of classification algorithms. These metrics measure the ability of the model to forecast correct labels of classes, they consist of accuracy, precision, recall, F1-score, and so on. They provide the exact description of the model working, especially in the case of effective datasets. (scikitlearn, 2024)

3.2.3 Accuracy Metrics

Accuracy metrics are used to calculate the accuracy of a classification model. The most common metric is accuracy, which is the ratio of correct predictions to total predictions. Other metrics such as precision, recall, and F1-score also offer insights into model performance, especially when dealing with class imbalances or different types of errors in prediction. (scikitlearn, 2024)

3.3 Pseudocode

START
IMPORT necessary libraries
LOAD dataset
EXPLORE dataset
IF ANY repeating row
    REMOVE repeating rows
END IF

IF ANY dataset value **EQUAL TO** NaN
    REMOVE NaN values
END IF

VISUALIZE the data
SPLIT dataset into train and test sets
IMPORT KNN classifier model
FIT training data into KNN model
USE test data for prediction
CALCULATE Euclidean distance from the test data to all the train data fit into the model
EXTRACT K number of nearest neighbors in terms of distance
EVALUATE the majority result/instance amongst those neighbors
PREDICT the test data using that majority result
CREATE classification report
CREATE confusion matrix
CALCULATE accuracy
STOP

START

IMPORT necessary libraries

LOAD dataset

EXPLORE dataset

IF ANY repeating row

    REMOVE repeating rows

END IF

IF ANY dataset value **EQUAL TO** NaN

    REMOVE NaN values

END IF

VISUALIZE the data

SPLIT dataset into train and test sets

IMPORT Support Vector Machine Classifier

FIT training data into support vector machine classifier model

END FIT

CALCULATE the best boundary that separates the classes

MAXIMIZE the gap between the boundary and the closest data points

ENSURE data points are on the correct side of the boundary

USE test data for prediction

PREDICT test samples based on their position relative to the boundary

CREATE classification metrics:

    CREATE classification report

    CREATE confusion matrix

CALCULATE accuracy

STOP

START

IMPORT necessary libraries

LOAD dataset

EXPLORE dataset

IF ANY repeating row

    REMOVE repeating rows

END IF

IF ANY dataset value **EQUAL TO** NaN

    REMOVE NaN values

END IF

VISUALIZE the data

SPLIT dataset into train and test sets

IMPORT Random Forest Classifier

FIT training data into random forest classifier model

END FIT

USE test data for prediction

CALCULATE Gini Index for all columns

GENERATE multiple decision trees using leaf and root notes

EVALUATE the majority result/instance amongst those trees

PREDICT the test data using that majority result

CREATE classification metrics:

    CREATE classification report

    CREATE confusion matrix

CALCULATE accuracy

STOP

3.4 Flowchart

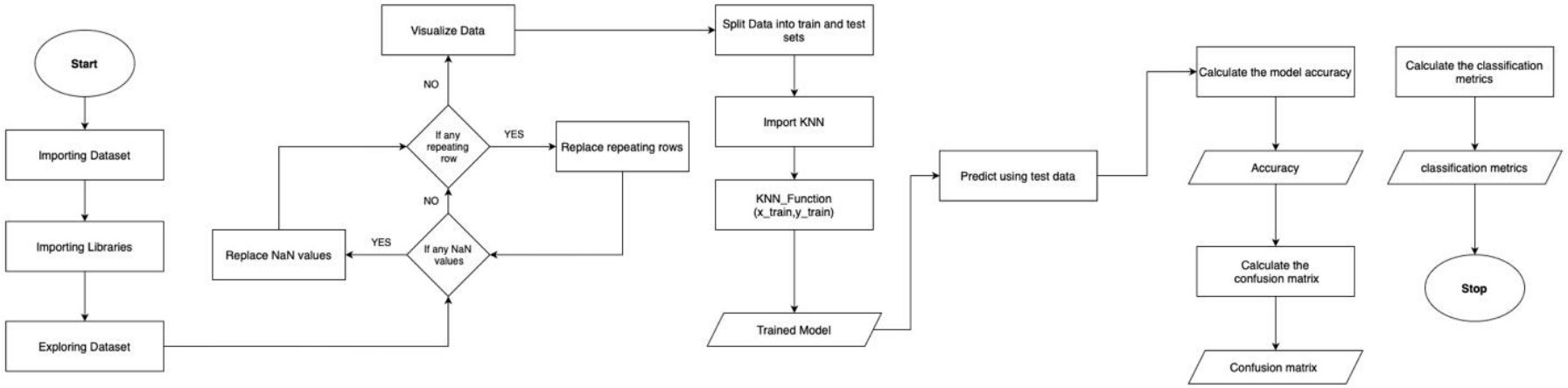


Figure 16: Flowchart of the KNN

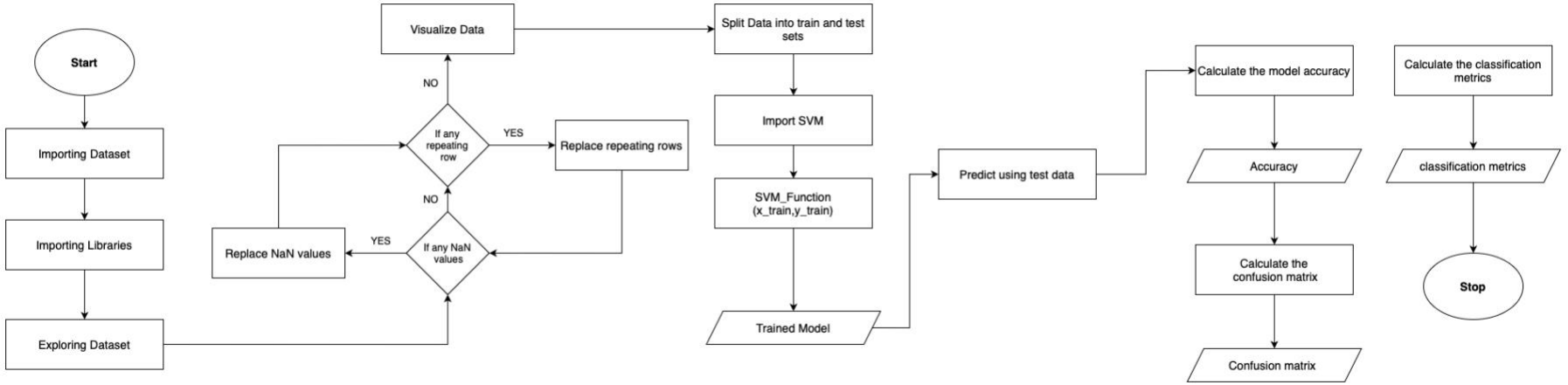


Figure 17: Flowchart of SVM

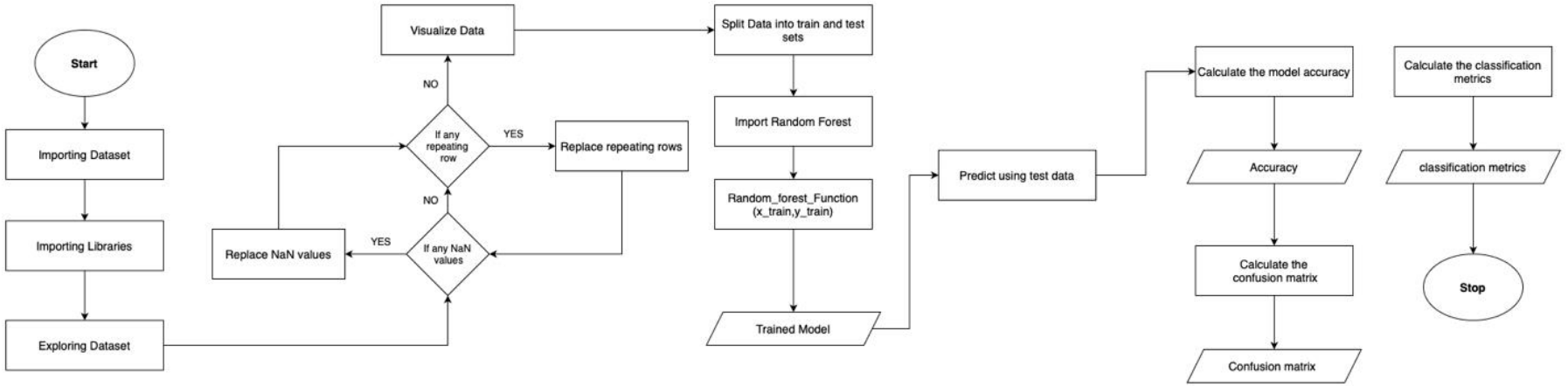


Figure 18: Flowchart of Random Forest

3.5 Development Process

3.5.1 Tools Used

- Jupyter Notebook
- Matplotlib
- NumPy
- Pandas
- Seaborn

3.5.2 Explanation of the development process

3.5.2.1 Importing Libraries

During this phase, libraries for machine learning, data manipulation, and visualisation were imported. The libraries such as matplotlib, seaborn, scikit-learn, pandas, and NumPy to efficiently implement machine learning algorithms and manage data was used and unnecessary warnings that may appear during the project are suppressed.

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV, RepeatedStratifiedKFold
from sklearn.metrics import make_scorer, accuracy_score, precision_score, recall_score, f1_score, mean_squared_error, confusion_matrix
```

```
# Filtering unnecessary warnings
warnings.filterwarnings('ignore')
```

Figure 19: Importing necessary libraries and filtering unnecessary warnings

3.5.2.2 Loading Dataset

The dataset, which is in CSV format, is loaded into the project to train and test the machine learning models and showing the first and last 5 rows of DataFrame.

```
[ ] # Reading the dataset from a CSV file into a DataFrame
    df = pd.read_csv('DataSet.csv')
```

```
[ ] df.head()
```

	having_ip_address	length_of_url	shortening_services	having_at_symbol	double-slash_redirection	prefix and suffix	sub_domains	ssl_state	domain_registered	favicons	...	on_mouse_over	right_click	po
0	-1	-1	-1	-1	-1	-1	1	1	1	1	...	1	1	
1	-1	0	-1	-1	-1	1	0	-1	1	1	...	-1	-1	
2	-1	-1	-1	-1	-1	-1	-1	-1	1	1	...	-1	-1	
3	-1	-1	-1	-1	-1	-1	-1	-1	1	1	...	-1	-1	
4	-1	1	-1	-1	-1	1	1	-1	1	1	...	-1	-1	

5 rows × 29 columns

```
[ ] df.tail()
```

	having_ip_address	length_of_url	shortening_services	having_at_symbol	double-slash_redirection	prefix and suffix	sub_domains	ssl_state	domain_registered	favicons	...	on_mouse_over	right_click
14088	-1	-1	-1	-1	-1	-1	0	-1	1	-1	...	-1	-1
14089	-1	-1	-1	-1	-1	-1	0	-1	1	-1	...	-1	-1
14090	-1	-1	-1	-1	-1	-1	0	-1	1	-1	...	-1	-1
14091	-1	-1	-1	-1	-1	-1	0	-1	1	-1	...	-1	-1
14092	-1	-1	-1	-1	-1	-1	0	-1	1	-1	...	-1	-1

5 rows × 29 columns

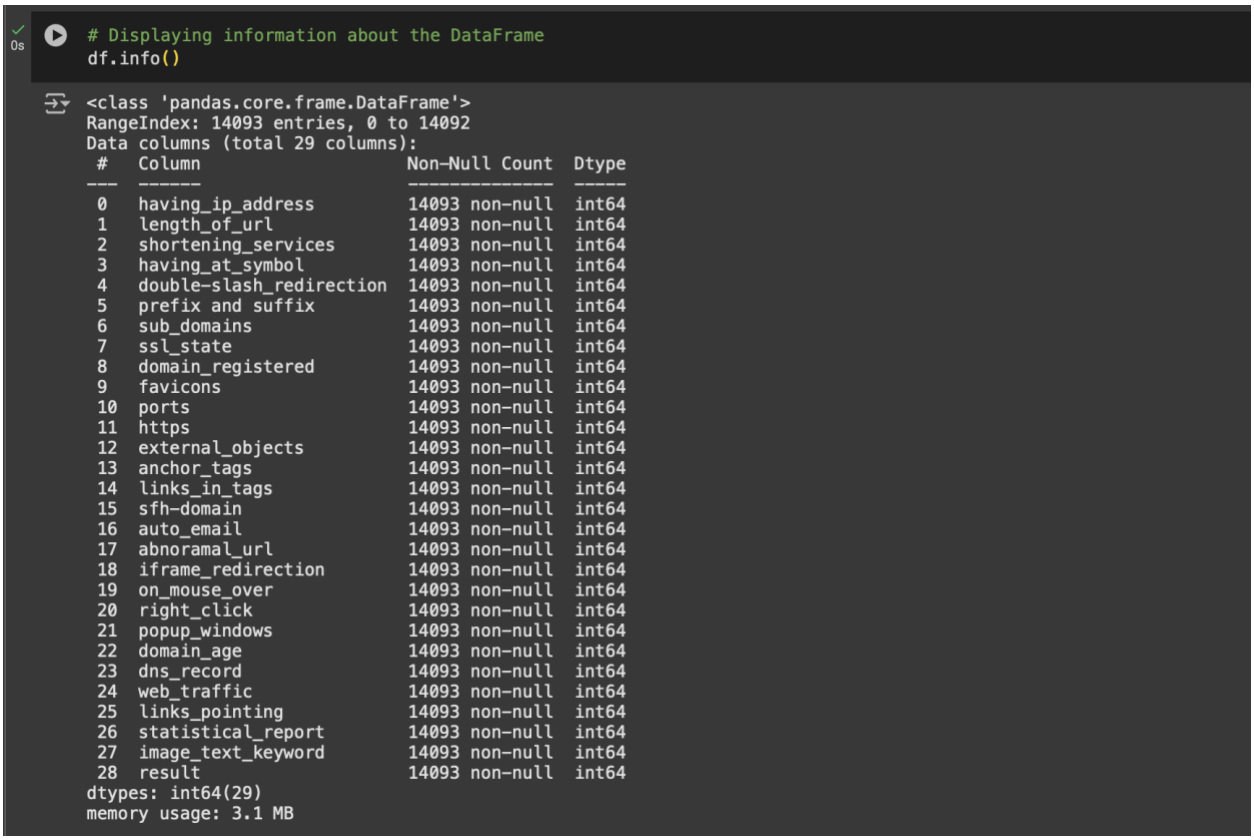
Figure 20: Loading CSV file and displaying first and last five rows

3.5.2.3 Exploring Dataset

The code is used to display the number of rows and columns of the dataset. The result displays that there is a total of 14093 rows and 29 columns in the dataset.

Figure 21 : Displaying the dimensions of the dataset

The dataset's attributes are displayed, including non-null counts and data types. The results show that all columns are of integer data type and have no null values.

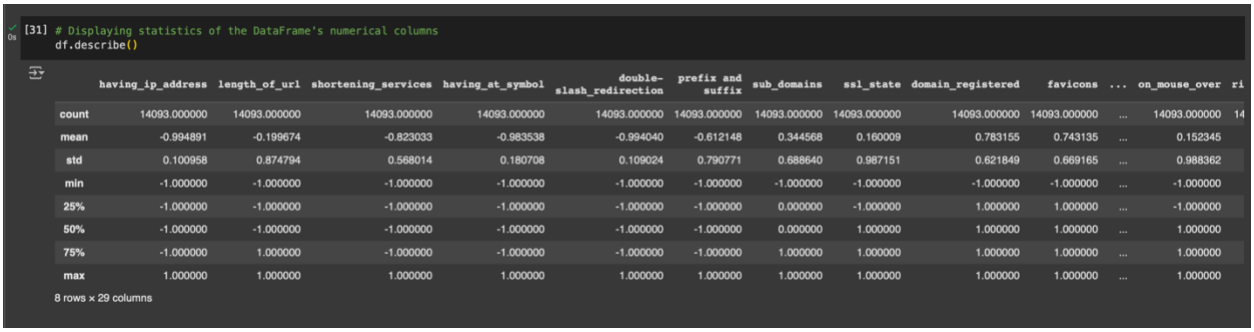


```
# Displaying information about the DataFrame
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14093 entries, 0 to 14092
Data columns (total 29 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   having_ip_address         14093 non-null  int64
 1   length_of_url             14093 non-null  int64
 2   shortening_services       14093 non-null  int64
 3   having_at_symbol          14093 non-null  int64
 4   double-slash_redirection  14093 non-null  int64
 5   prefix and suffix         14093 non-null  int64
 6   sub_domains               14093 non-null  int64
 7   ssl_state                 14093 non-null  int64
 8   domain_registered         14093 non-null  int64
 9   favicons                  14093 non-null  int64
 10  ports                     14093 non-null  int64
 11  https                     14093 non-null  int64
 12  external_objects          14093 non-null  int64
 13  anchor_tags               14093 non-null  int64
 14  links_in_tags             14093 non-null  int64
 15  sfh-domain                14093 non-null  int64
 16  auto_email                14093 non-null  int64
 17  abnoramal_url             14093 non-null  int64
 18  iframe_redirection        14093 non-null  int64
 19  on_mouse_over             14093 non-null  int64
 20  right_click               14093 non-null  int64
 21  popup_windows             14093 non-null  int64
 22  domain_age                14093 non-null  int64
 23  dns_record                14093 non-null  int64
 24  web_traffic               14093 non-null  int64
 25  links_pointing            14093 non-null  int64
 26  statistical_report        14093 non-null  int64
 27  image_text_keyword        14093 non-null  int64
 28  result                    14093 non-null  int64
dtypes: int64(29)
memory usage: 3.1 MB
```

Figure 22 : Displaying information of dataset

The DataFrame displays statistical information for each attribute, including the total number of values, mean, standard deviation, minimum value, first quartile (median), third quartile, and maximum value.



```
[31] # Displaying statistics of the DataFrame's numerical columns
     df.describe()
```

	having_ip_address	length_of_url	shortening_services	having_at_symbol	double-slash_redirection	prefix and suffix	sub_domains	ssl_state	domain_registered	favicons	...	on_mouse_over	ri
count	14093.000000	14093.000000	14093.000000	14093.000000	14093.000000	14093.000000	14093.000000	14093.000000	14093.000000	14093.000000	...	14093.000000	14
mean	-0.994891	-0.199674	-0.823033	-0.983538	-0.994040	-0.612148	0.344568	0.160009	0.783155	0.743135	...	0.152345	
std	0.100958	0.874794	0.568014	0.180708	0.109024	0.790771	0.688640	0.987151	0.621849	0.669165	...	0.988362	
min	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	...	-1.000000	
25%	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	0.000000	-1.000000	1.000000	1.000000	...	-1.000000	
50%	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	0.000000	1.000000	1.000000	1.000000	...	1.000000	
75%	-1.000000	1.000000	-1.000000	-1.000000	-1.000000	-1.000000	1.000000	1.000000	1.000000	1.000000	...	1.000000	
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	...	1.000000	

8 rows × 29 columns

Figure 23: Displaying statistics of the dataset's numerical columns

The figure displays the number of unique values in each column.

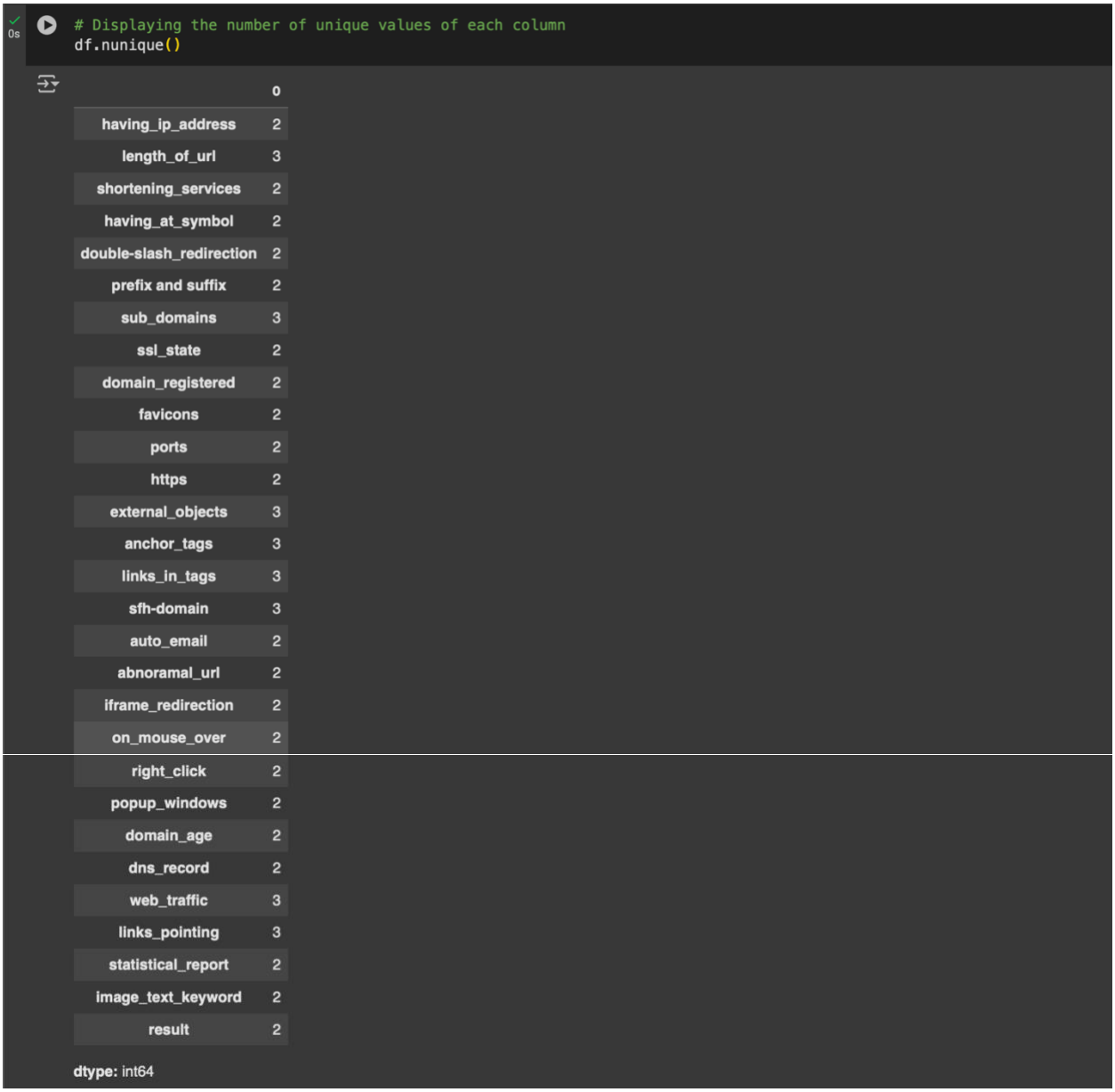


Figure 24: Displaying the number of unique values in each column

.

3.5.2.4 Cleaning Dataset

Data cleaning involves removing outliers, filling in missing values, and maintaining the dataset's integrity. Prepare a clean and reliable dataset for model training. Since the dataset is already cleaned by its owner, there’s no cleaning to do.

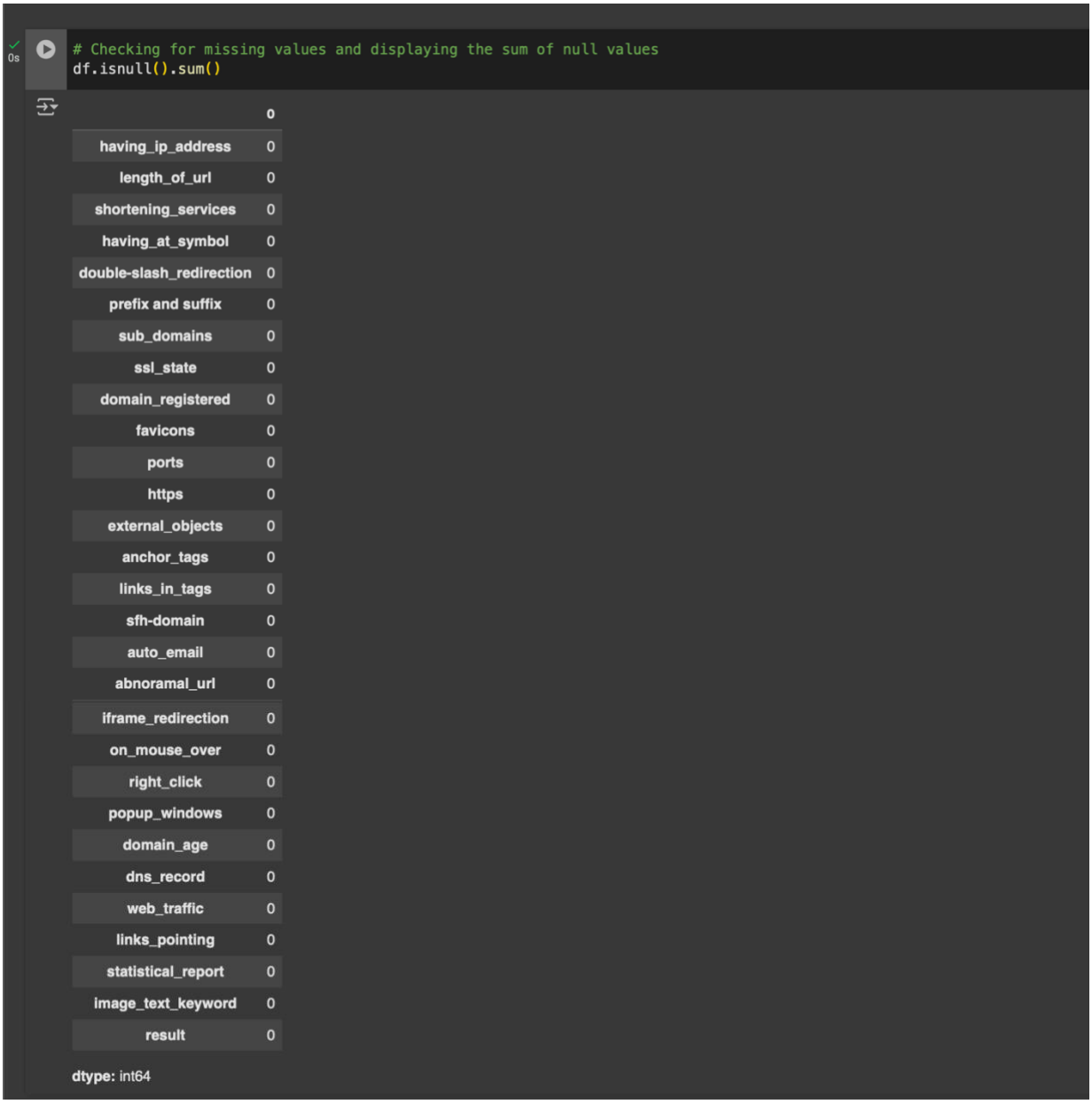
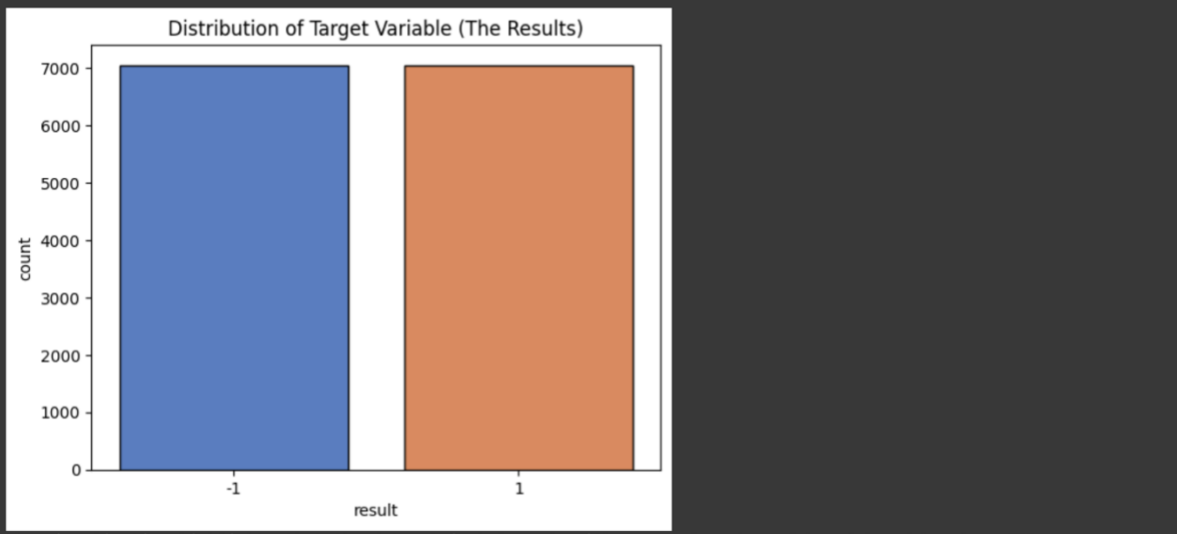


Figure 25 : Displaying the sum of null values

3.5.2.5 Visualizing Data

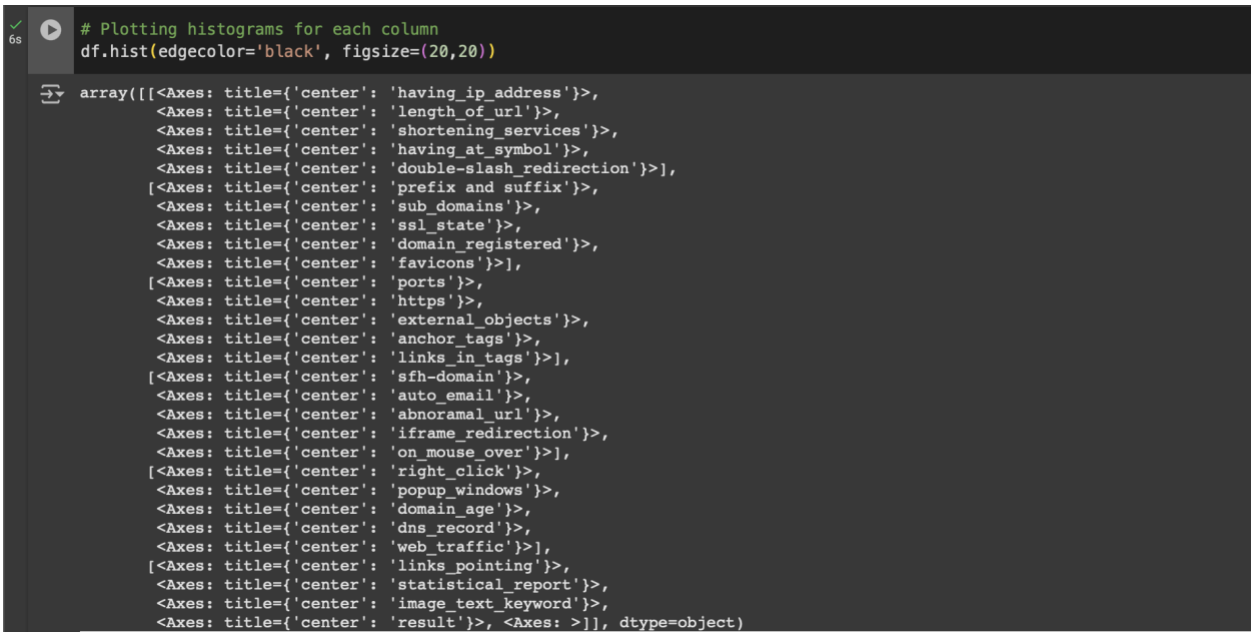
Data visualisation tools like matplotlib and seaborn are used to generate graphical representations of datasets. Visualisations facilitate understanding of data patterns, trends, and relationships. The code selects a colour scheme from the Seaborn library and generates a count plot, where the x-axis shows the values of the "result" column, and the y-axis shows the number of each value. The graph demonstrates how the data is split equally to distinguish between safe and dangerous URLs.

```
# Visualizing the distribution of the target variable 'result'

color = sns.color_palette("muted", 2)  # Changed to 'muted' color palette
sns.countplot(x='result', data=df, palette=color, edgecolor='black')  # Updated edge color to black
plt.title('Distribution of Target Variable (The Results)')
plt.show()

print('The visualization highlights that the dataset is balanced, with an equal number of harmful and harmless URLs.')
```




```
The visualization highlights that the dataset is balanced, with an equal number of harmful and harmless URLs.
```

Figure 26: Displaying the distribution of Target Variable (The Results)



```
# Plotting histograms for each column
df.hist(edgecolor='black', figsize=(20,20))
```

```
array([[<Axes: title={'center': 'having_ip_address'}>,
        <Axes: title={'center': 'length_of_url'}>,
        <Axes: title={'center': 'shortening_services'}>,
        <Axes: title={'center': 'having_at_symbol'}>,
        <Axes: title={'center': 'double-slash_redirection'}>],
       [<Axes: title={'center': 'prefix and suffix'}>,
        <Axes: title={'center': 'sub_domains'}>,
        <Axes: title={'center': 'ssl_state'}>,
        <Axes: title={'center': 'domain_registered'}>,
        <Axes: title={'center': 'favicons'}>],
       [<Axes: title={'center': 'ports'}>,
        <Axes: title={'center': 'https'}>,
        <Axes: title={'center': 'external_objects'}>,
        <Axes: title={'center': 'anchor_tags'}>,
        <Axes: title={'center': 'links_in_tags'}>],
       [<Axes: title={'center': 'sfh-domain'}>,
        <Axes: title={'center': 'auto_email'}>,
        <Axes: title={'center': 'abnoramal_url'}>,
        <Axes: title={'center': 'iframe_redirection'}>,
        <Axes: title={'center': 'on_mouse_over'}>],
       [<Axes: title={'center': 'right_click'}>,
        <Axes: title={'center': 'popup_windows'}>,
        <Axes: title={'center': 'domain_age'}>,
        <Axes: title={'center': 'dns_record'}>,
        <Axes: title={'center': 'web_traffic'}>],
       [<Axes: title={'center': 'links_pointing'}>,
        <Axes: title={'center': 'statistical_report'}>,
        <Axes: title={'center': 'image_text_keyword'}>,
        <Axes: title={'center': 'result'}>, <Axes: >]], dtype=object)
```

Figure 27: Plotting histograms for each columns

Figure 28 : Subplot of each columns

Plotting a pie chart for the dataframe's "abnormal_url" column reveals that 58.53% of the URLs may be fake.

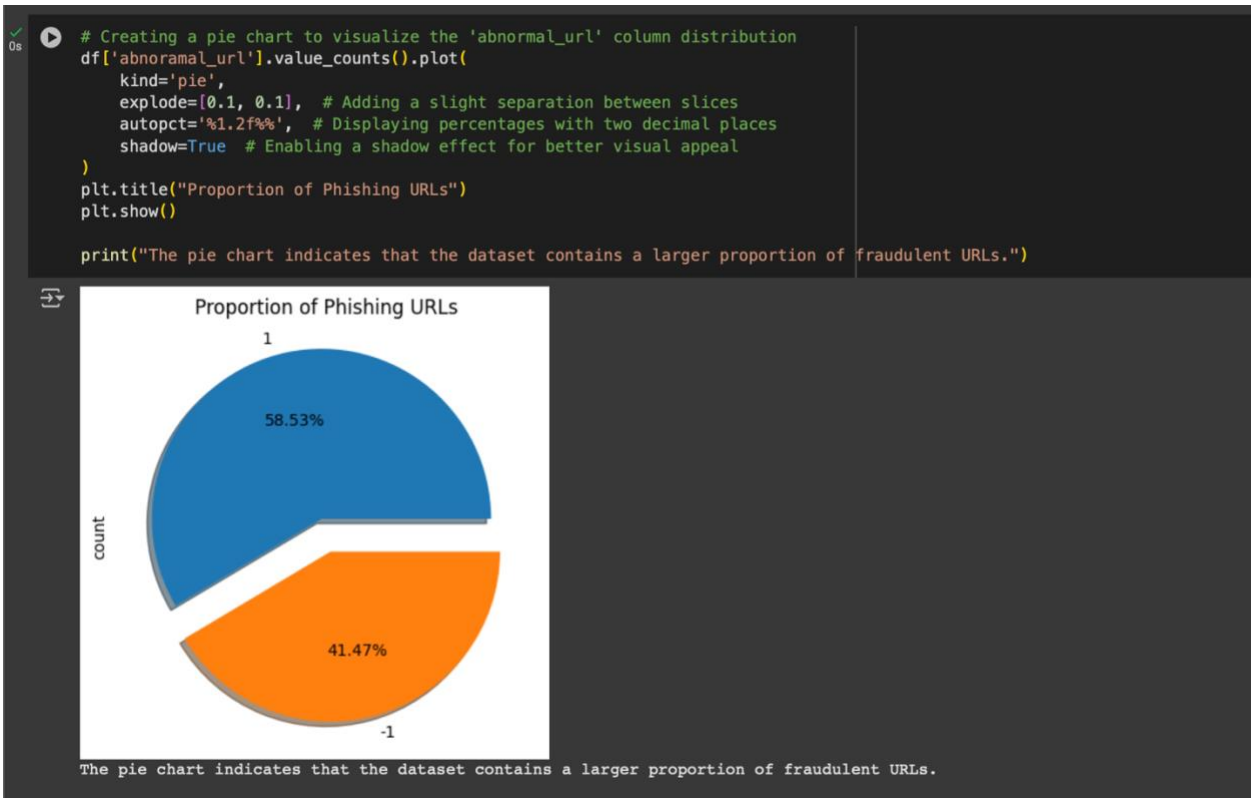
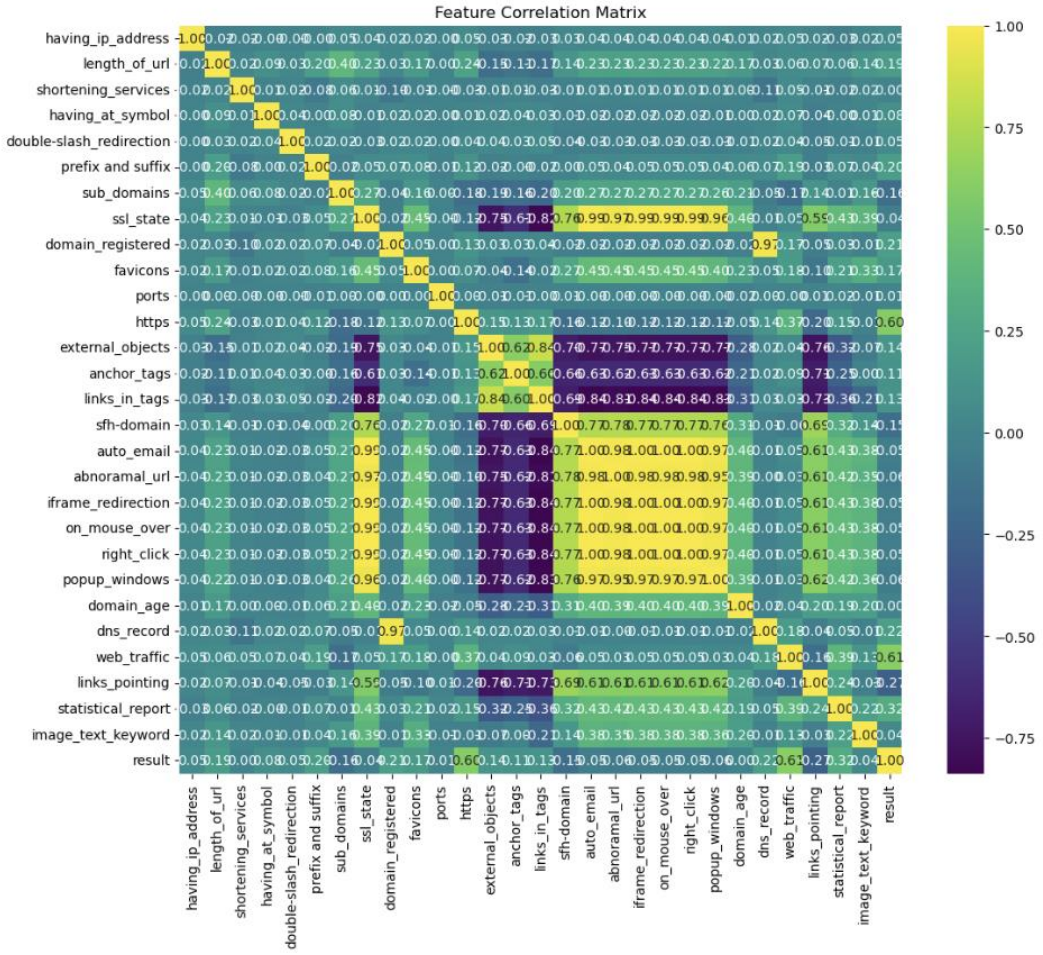


Figure 29 : pie chart for the 'abnormal_url' column

To see how each column in the DataFrame relates to the others, a correlation heatmap is plotted for each column.

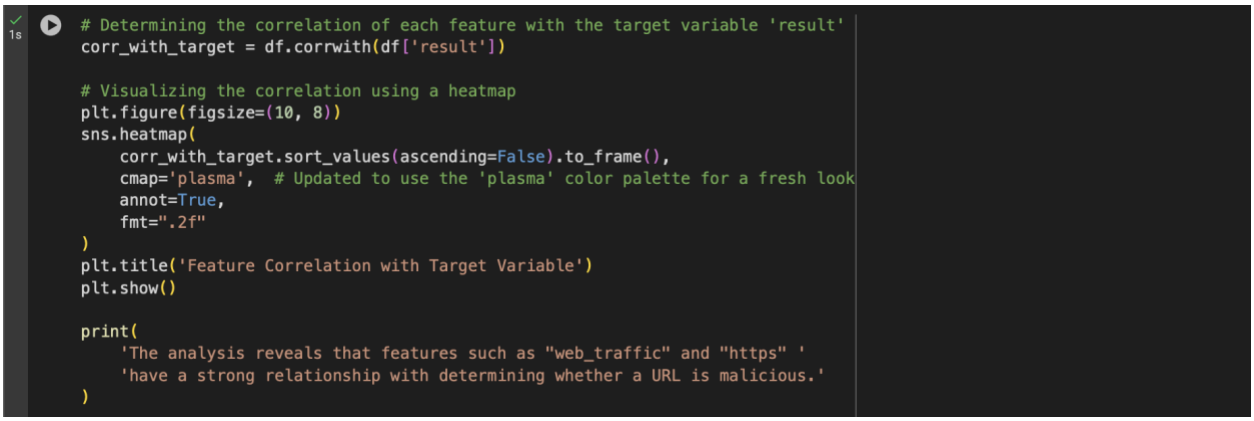
```
# Displaying the correlation matrix to analyze feature relationships
plt.figure(figsize=(12, 10))
sns.heatmap(df.corr(), annot=True, cmap='viridis', fmt=".2f")  # Using 'viridis' for a vibrant color palette
plt.title('Feature Correlation Matrix')
plt.show()

print("The heatmap illustrates the degree of correlation between various features in the dataset.")
```




```
The heatmap illustrates the degree of correlation between various features in the data
set.
```

Figure 30 : Correlation matrix between all the features

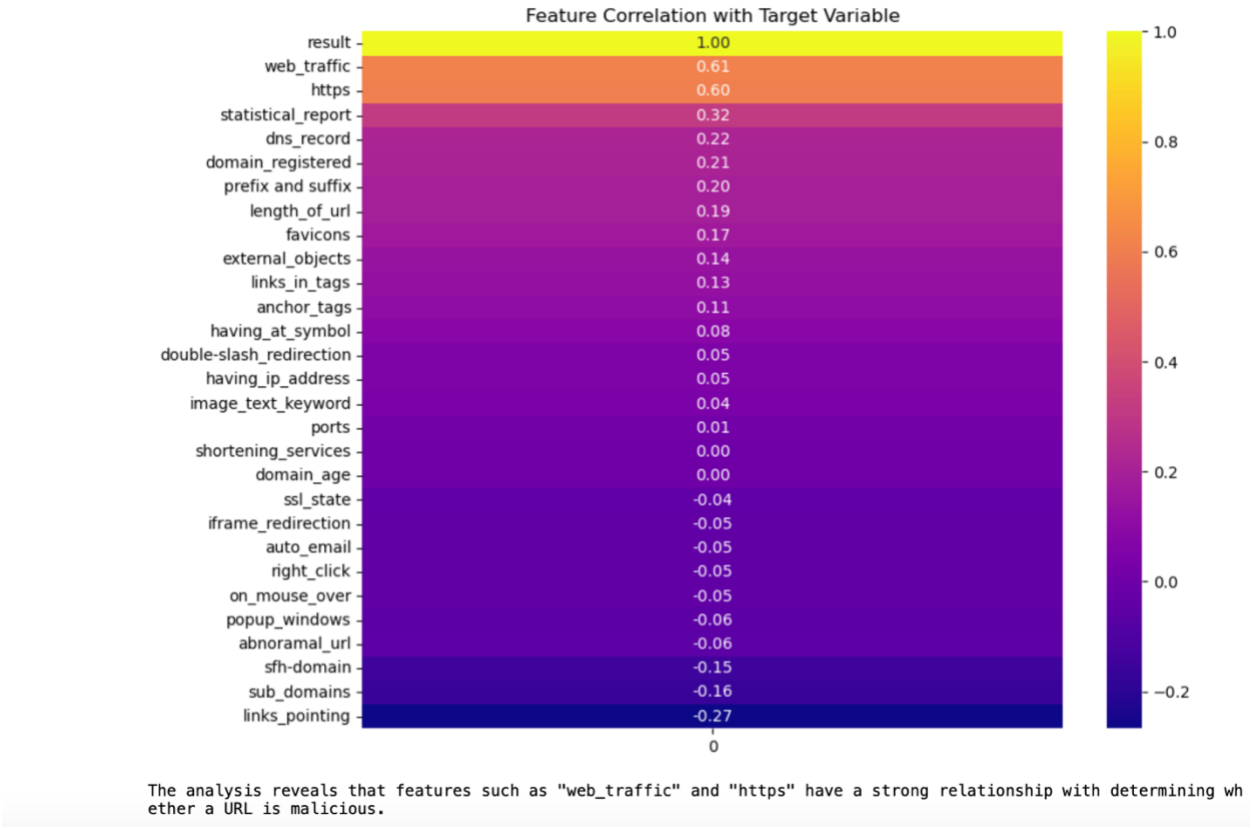
To determine the relationship between "result" and the other attributes, a heatmap is plotted once the correlation between each column and the target variable is determined.



```
# Determining the correlation of each feature with the target variable 'result'
corr_with_target = df.corrwith(df['result'])

# Visualizing the correlation using a heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(
    corr_with_target.sort_values(ascending=False).to_frame(),
    cmap='plasma',  # Updated to use the 'plasma' color palette for a fresh look
    annot=True,
    fmt=".2f"
)
plt.title('Feature Correlation with Target Variable')
plt.show()

print(
    'The analysis reveals that features such as "web_traffic" and "https" '
    'have a strong relationship with determining whether a URL is malicious.'
)
```

Figure 31 : Calculating correlation of each feature

The correlation shows that "web_traffic" and "https" have a strong relationship with the goal variable "result," indicating that it is essential for determining whether the URL is fraudulent.




```
The analysis reveals that features such as "web_traffic" and "https" have a strong relationship with determining wh
ether a URL is malicious.
```

Figure 32 : Visualizing correlation of each feature

A subplot of the "length_of_url" column is shown, indicating that most URLs are deemed suspicious since they are lengthier than typical.

```
# Analyzing the distribution of URL lengths using a histogram
plt.figure(figsize=(14, 10))
plt.subplot(2, 2, 1)
sns.histplot(df['length_of_url'], bins=30, kde=True, color='teal')  # Changed color to 'teal'
plt.title('URL Length Distribution')

print(
    'The histogram demonstrates that the majority of URLs are lengthy, '
    'indicating a higher prevalence of fraudulent URLs in the dataset.'
)
```

```
The histogram demonstrates that the majority of URLs are lengthy, indicating a higher prevalence of fraudulent URLs in the dataset.
```

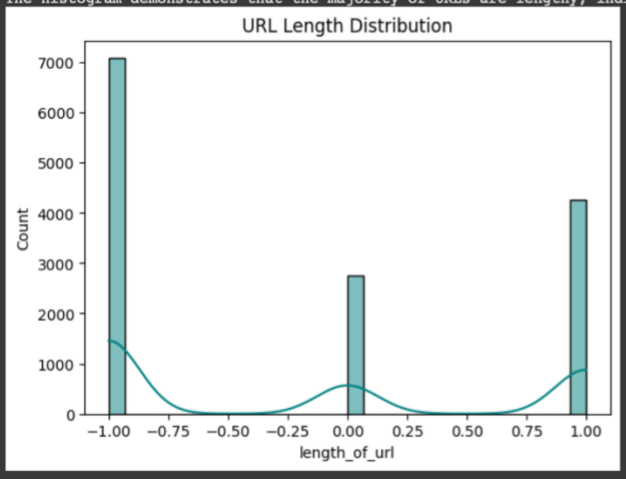
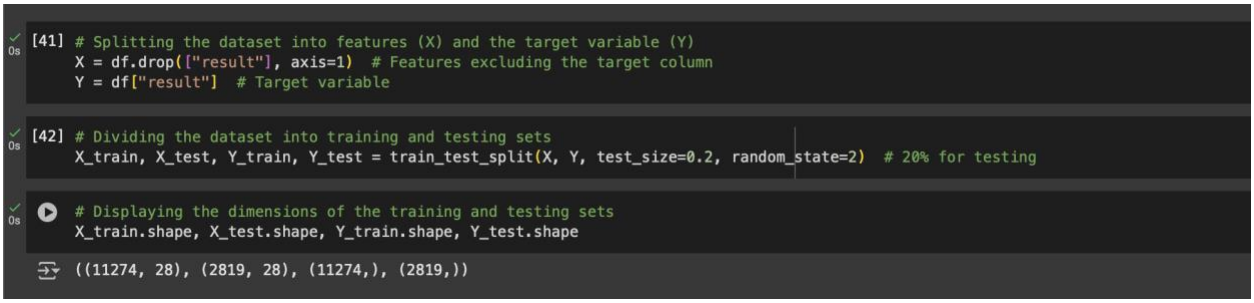


Figure 33 : Distribution of URL length using a histogram

3.5.2.6 Train and Test Split

Two variables are established to separate the dataset, one variable only contains the "result" column, while the other variable has the "result" column deleted. After dividing the dataset into train (80%) and test (20%) sets, the dimensions of each set are shown.



```
[41] # Splitting the dataset into features (X) and the target variable (Y)
     X = df.drop(["result"], axis=1)  # Features excluding the target column
     Y = df["result"]  # Target variable

[42] # Dividing the dataset into training and testing sets
     X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=2)  # 20% for testing

     # Displaying the dimensions of the training and testing sets
     X_train.shape, X_test.shape, Y_train.shape, Y_test.shape

     ((11274, 28), (2819, 28), (11274,), (2819,))
```

Figure 34 : Splitting the dataset into train and test

3.5.2.7 Training Models

3.5.2.7.1 K-Nearest Neighbours Classifier

A grid of hyperparameters is defined and the KNN classifier is initialised. The optimal hyperparameter for KNN is found through hyperparameter adjustment using grid search.

```
# Setting up the K-Nearest Neighbors (KNN) classifier
knn_model = KNeighborsClassifier()

# Specifying the range of hyperparameter values for tuning
knn_params = {'n_neighbors': [3, 5, 7, 9, 11, 13]}

# Applying GridSearchCV to identify the optimal hyperparameters
knn_grid_search = GridSearchCV(
    estimator=knn_model,
    param_grid=knn_params,
    refit=True,
    cv=5,  # 5-fold cross-validation
    scoring='accuracy'
)
```

Figure 35 : Initializing KNN model and defining grid of hyperparameter values

To determine the optimal hyperparameter and its score, the grid search is fitted using the training data.

```
# Training the KNN model using grid search
knn_grid_search.fit(X_train, Y_train)

# Extracting the optimal hyperparameters
knn_best_params = knn_grid_search.best_params_
print("Optimal KNN Hyperparameters: ", knn_best_params)

# Retrieving the highest cross-validation accuracy achieved by the KNN model
knn_best_score = knn_grid_search.best_score_
print("Best Cross-Validated Score for KNN: ", knn_best_score)
```

```
Optimal KNN Hyperparameters:  {'n_neighbors': 11}
Best Cross-Validated Score for KNN:  0.907663144309107
```

Figure 36 : Fitting KNN grid search

The KNN model is initialised with optimal hyperparameters and fitted with training data and then model makes predictions based on the test data.

```
# Creating a new KNN model with the best hyperparameter
knn_best_model = KNeighborsClassifier(n_neighbors=list(knn_best_params.values())[0])

# Fitting the best KNN model with the training data
knn_best_model.fit(X_train, Y_train)

# Predicting using the test dataset with the best model
knn_y_pred = knn_best_model.predict(X_test)
```

Figure 37 : Creating new KNN model with best hyperparameter

The accuracy, precision, recall, f1 score and error is calculated and displayed for the model.

```
# Displaying the accuracy of the optimized KNN model
knn_accuracy = accuracy_score(Y_test, knn_y_pred)
print(f"KNN Accuracy: {knn_accuracy:.3f}")

# Displaying the precision of the optimized KNN model
knn_precision = precision_score(Y_test, knn_y_pred)
print(f"KNN Precision: {knn_precision:.3f}")

# Displaying the recall of the optimized KNN model
knn_recall = recall_score(Y_test, knn_y_pred)
print(f"KNN Recall: {knn_recall:.3f}")

# Displaying the F1 score of the optimized KNN model
knn_f1 = f1_score(Y_test, knn_y_pred)
print(f"KNN F1 Score: {knn_f1:.3f}")

# Displaying the error rate of the optimized KNN model
knn_error = 1 - knn_accuracy
print(f"KNN Error Rate: {knn_error:.3f}")
```

```
KNN Accuracy: 0.909
KNN Precision: 0.902
KNN Recall: 0.918
KNN F1 Score: 0.910
KNN Error Rate: 0.091
```

Figure 38 : Calculating and displaying accuracy, precision, recall, f1 score and error

The confusion matrix is calculated for the KNN model

```
# Generating and displaying the confusion matrix for the KNN model
knn_con_matrix = confusion_matrix(Y_test, knn_y_pred)
print(f"Confusion Matrix for KNN Model:\n{knn_con_matrix}")
```

```
Confusion Matrix for KNN Model:
[[1260  141]
 [ 116 1302]]
```

Figure 39 : Calculating confusion matrix

The confusion matrix of the model being displayed as a heatmap.

```
# Visualizing the confusion matrix using a heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(knn_con_matrix, annot=True, fmt="d", cmap="coolwarm", cbar=False,
            xticklabels=['Predicted Negative', 'Predicted Positive'],
            yticklabels=['Actual Negative', 'Actual Positive'])
plt.title('KNN Confusion Matrix')
plt.xlabel('Predicted Labels')
plt.ylabel('Actual Labels')
plt.show()
```

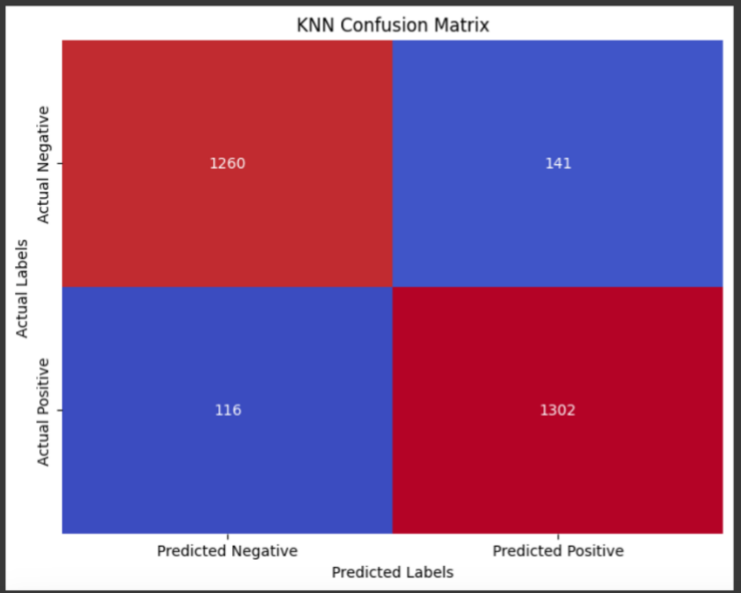


Figure 40: Plotting confusion matrix on heatmap

3.5.2.7.2 Random Forest Classifier

The Random Forest classifier is set up with a grid of hyperparameters. Grid search is used to optimise hyperparameters for Random Forest Classifiers.

```
# Setting up the Random Forest (RF) classifier
rf_model = RandomForestClassifier()

# Specifying the hyperparameter grid for tuning the RF model
rf_param_grid = {
    'n_estimators': [50, 100, 150],          # Number of trees in the forest
    'max_depth': [None, 10, 20, 30],         # Maximum depth of each tree
    'min_samples_split': [2, 5, 10],         # Minimum samples required to split an internal node
    'min_samples_leaf': [1, 2, 4]            # Minimum samples required to be at a leaf node
}

# Setting up GridSearchCV to identify the optimal hyperparameters for the RF model
rf_grid_search = GridSearchCV(
    estimator=rf_model,
    param_grid=rf_param_grid,
    cv=5,
    scoring='accuracy'
)
```

Figure 41 : Initializing RF model and defining grid of hyperparameter

The grid search is fit with the training data to get the best hyperparameter along with its score.

```
# Training the Random Forest model
rf_grid_search.fit(X_train, Y_train)

# Retrieving the optimal hyperparameters
rf_best_params = rf_grid_search.best_params_
print("Optimal Hyperparameters for Random Forest: ", rf_best_params)

# Retrieving the highest cross-validation accuracy score achieved
rf_best_score = rf_grid_search.best_score_
print("Best Cross-Validation Accuracy for Random Forest: ", rf_best_score)
```

```
Optimal Hyperparameters for Random Forest:  {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 100}
Best Cross-Validation Accuracy for Random Forest:  0.9228303857935731
```

Figure 42 : Getting the best hyperparameter

The Random Forest Classifier is defined with the best hyperparameters for the best results.

```
# Random Forest model using the optimal hyperparameters obtained from GridSearchCV
best_rf_model = RandomForestClassifier(
    n_estimators=rf_best_params['n_estimators'],
    max_depth=rf_best_params['max_depth'],
    min_samples_split=rf_best_params['min_samples_split'],
    min_samples_leaf=rf_best_params['min_samples_leaf']
)
```

Figure 43 : Creating the best hyperparameter

The Random Forest Classifier model is fit with the training data and then the model predicts using the test dataset.

```
# Training the optimized Random Forest model on the training dataset
best_rf_model.fit(X_train, Y_train)

# Making predictions on the test dataset using the trained model
rf_y_pred = best_rf_model.predict(X_test)
```

Figure 44 : Fitting the new RF model with training data

The accuracy, precision, recall, f1 score and error is calculated and displayed for the model.

```
# Displaying the accuracy of the Random Forest model
rf_accuracy = accuracy_score(Y_test, rf_y_pred)
print(f"Accuracy of Random Forest Model: {rf_accuracy: .3f}")

# Displaying the precision of the Random Forest model
rf_precision = precision_score(Y_test, rf_y_pred)
print(f"Precision of Random Forest Model: {rf_precision: .3f}")

# Displaying the recall of the Random Forest model
rf_recall = recall_score(Y_test, rf_y_pred)
print(f"Recall of Random Forest Model: {rf_recall: .3f}")

# Displaying the F1 score of the Random Forest model
rf_f1 = f1_score(Y_test, rf_y_pred)
print(f"F1 Score of Random Forest Model: {rf_f1: .3f}")

# Displaying the error rate of the Random Forest model
rf_error = 1 - rf_accuracy
print(f"Error Rate of Random Forest Model: {rf_error: .3f}")
```

```
Accuracy of Random Forest Model:  0.923
Precision of Random Forest Model:  0.913
Recall of Random Forest Model:  0.936
F1 Score of Random Forest Model:  0.924
Error Rate of Random Forest Model:  0.077
```

Figure 45 : Displaying accuracy, precision, recall, f1 score and error

The confusion matrix is calculated for the Random Forest model and displayed in the form of heatmap.

```
# Generating a confusion matrix for the Random Forest model
rf_con_matrix = confusion_matrix(y_test, rf_y_pred)

# Visualizing the confusion matrix with a heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(rf_con_matrix, annot=True, fmt="d", cmap="coolwarm", cbar=False,
            xticklabels=['Predicted: Not Harmful', 'Predicted: Harmful'],
            yticklabels=['Actual: Not Harmful', 'Actual: Harmful'])
plt.title('Confusion Matrix - Random Forest Model')
plt.xlabel('Predicted Labels')
plt.ylabel('Actual Labels')
plt.show()
```

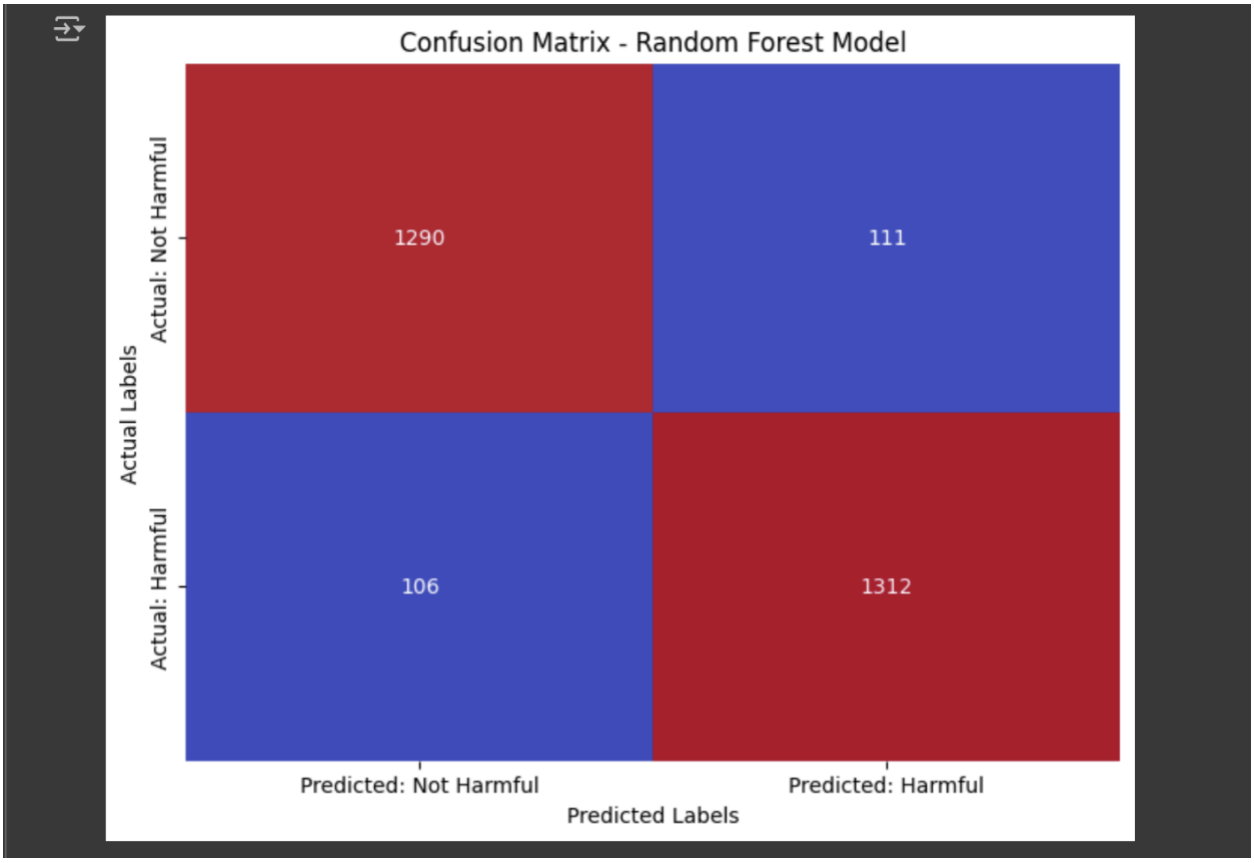


Figure 46 : Plotting confusion matrix of RF model

3.5.2.7.3 Support Vector Machines (SVM)

The SVM model is initialised, and a hyperparameter grid is defined. Grid search is used to optimise hyperparameters for SVMs.

```
[63] # Initializing the Support Vector Machine (SVM) model
     svm_model = SVC()

     # Defining a range of hyperparameters for SVM tuning
     svm_param_grid = {
         'C': [0.1, 1, 10, 100],  # Regularization parameter
         'kernel': ['linear', 'rbf', 'poly'],  # Different types of kernel functions
         'gamma': ['scale', 'auto']
     }

     # Setting up GridSearchCV to perform hyperparameter optimization for the SVM model
     svm_grid_search = GridSearchCV(estimator=svm_model, param_grid=svm_param_grid, cv=5, scoring='accuracy')
```

Figure 47 : Initializing SVM model and defining grid of hyperparameter

The grid search is used with training data to determine the optimal hyperparameter and its score.

```
[64] # Fitting the SVM grid search to the training data
     svm_grid_search.fit(X_train, Y_train)

     # Getting the best hyperparameters after the grid search
     svm_best_params = svm_grid_search.best_params_
     print("SVM Best Hyperparameters: ", svm_best_params)

     # Getting the best cross-validation score from the grid search
     svm_best_score = svm_grid_search.best_score_
     print("SVM Best Score: ", svm_best_score)

SVM Best Hyperparameters:  {'C': 100, 'gamma': 'scale', 'kernel': 'rbf'}
SVM Best Score:  0.9207018220379831
```

Figure 48 : Getting the best hyperparameter for SVM

The SVM is defined with the best hyperparameters for the best results.

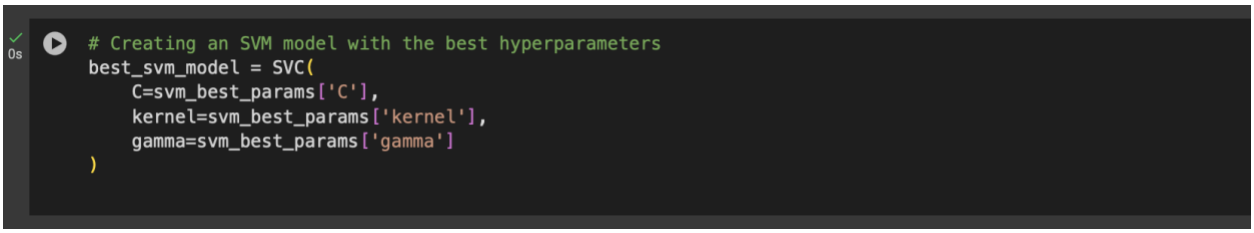


Figure 49 : Creating the best hyperparameter for SVM

The SVM model is fit with the training data and then the model predicts using the test dataset.

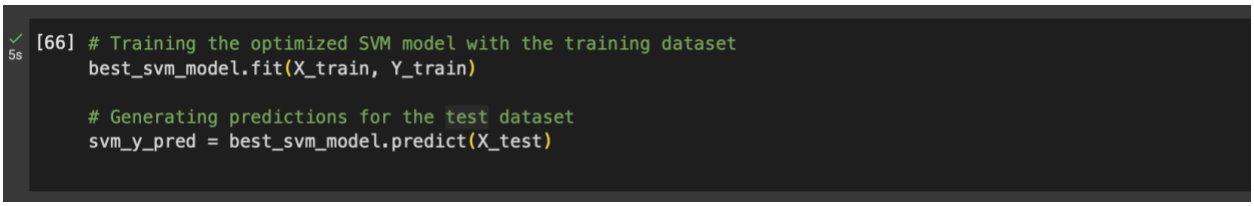


Figure 50 : Fitting the new SVM model with training data

The accuracy, precision, recall, f1 score and error is calculated and displayed for the model.

```
# Evaluating the accuracy of the SVM model
svm_accuracy = accuracy_score(Y_test, svm_y_pred)
print(f"SVM Model Accuracy: {svm_accuracy: .3f}")

# Calculating the precision of the SVM model
svm_precision = precision_score(Y_test, svm_y_pred)
print(f"SVM Precision Score: {svm_precision: .3f}")

# Calculating the recall of the SVM model
svm_recall = recall_score(Y_test, svm_y_pred)
print(f"SVM Recall Score: {svm_recall: .3f}")

# Calculating the F1 score for the SVM model
svm_f1 = f1_score(Y_test, svm_y_pred)
print(f"SVM F1 Score: {svm_f1: .3f}")

# Calculating the error rate of the SVM model
svm_error = 1 - svm_accuracy
print(f"SVM Model Error Rate: {svm_error: .3f}")
```

```
SVM Model Accuracy:  0.920
SVM Precision Score:  0.914
SVM Recall Score:  0.929
SVM F1 Score:  0.921
SVM Model Error Rate:  0.080
```

Figure 51 : Displaying accuracy, precision, recall, f1 score and error for SVM

The confusion matrix is calculated for the SVM and displayed in the form of heatmap.

```
# Generating the confusion matrix for SVM model predictions
svm_con_matrix = confusion_matrix(Y_test, svm_y_pred)

# Plotting the confusion matrix as a heatmap for better visualization
plt.figure(figsize=(8, 6))
sns.heatmap(svm_con_matrix, annot=True, fmt="d", cmap="coolwarm", cbar=False,
            xticklabels=['Predicted 0', 'Predicted 1'],
            yticklabels=['Actual 0', 'Actual 1'])
plt.title('SVM Confusion Matrix')
plt.xlabel('Predicted Values')
plt.ylabel('Actual Values')
plt.show()
```

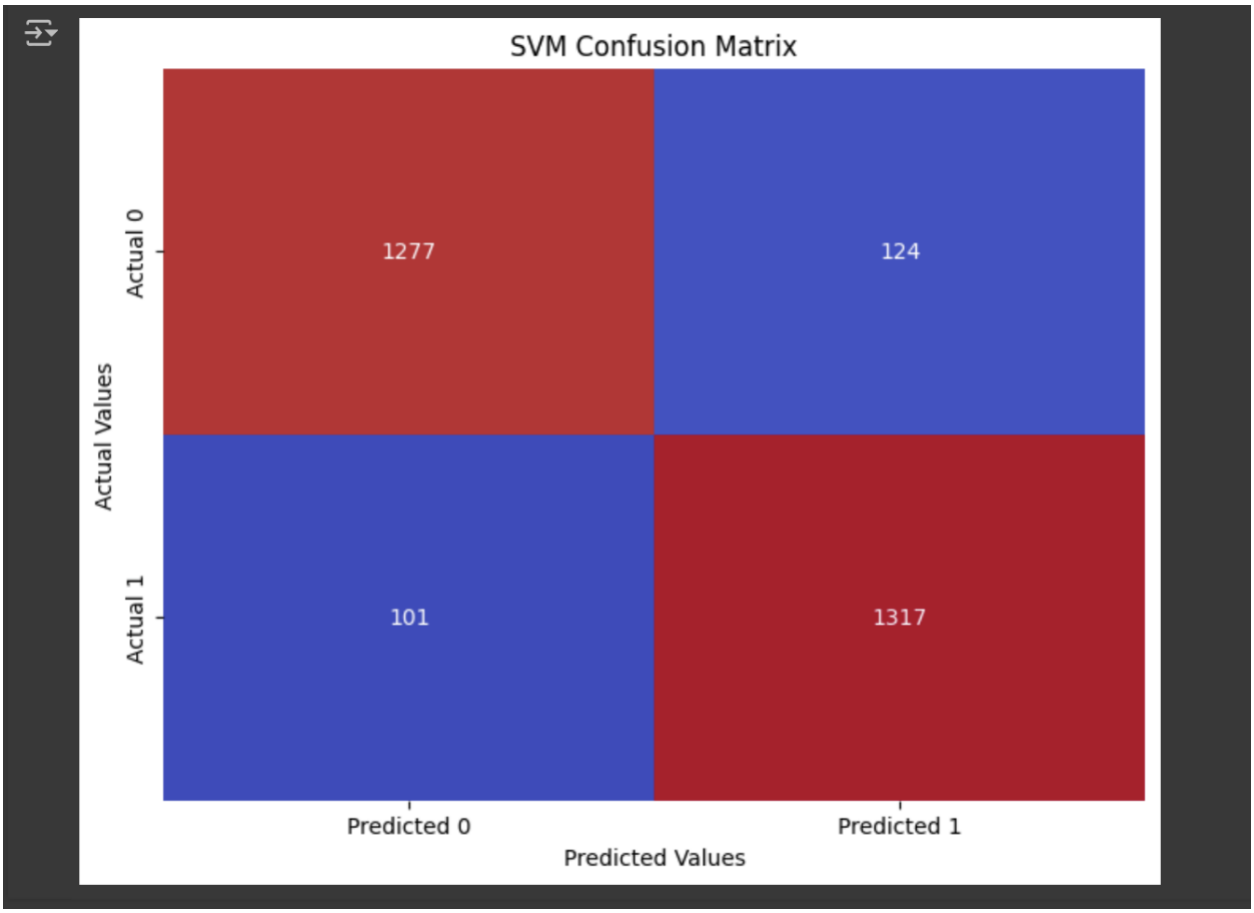


Figure 52 : Plotting confusion matrix for svm

3.5.3 Achieved Result

```
# Creating a dictionary to store performance metrics for each algorithm
metrics_dict = {
    'KNN': [knn_accuracy, knn_precision, knn_recall, knn_f1, knn_error],
    'Random Forest': [rf_accuracy, rf_precision, rf_recall, rf_f1, rf_error],
    'SVM': [svm_accuracy, svm_precision, svm_recall, svm_f1, svm_error]
}

# Converting the dictionary into a DataFrame for easier visualization
metrics_df = pd.DataFrame(metrics_dict, index=['Accuracy', 'Precision', 'Recall', 'F1 Score', 'Error'])

# Displaying the DataFrame
metrics_df
```

	KNN	Random Forest	SVM
Accuracy	0.908833	0.923022	0.920184
Precision	0.902287	0.913283	0.913949
Recall	0.918195	0.935825	0.928773
F1 Score	0.910171	0.924417	0.921301
Error	0.091167	0.076978	0.079816

Figure 53 : Performance metrics of all algorithms

```
# Plotting a horizontal bar chart to visualize the performance metrics of each algorithm
metrics_df.plot(kind='barh', figsize=(10, 6), colormap='viridis', width=0.8)
plt.title('Comparison of Performance Metrics Across Algorithms')
plt.xlabel('Score')
plt.ylabel('Algorithm')
plt.legend(title='Metrics')
plt.show()
```

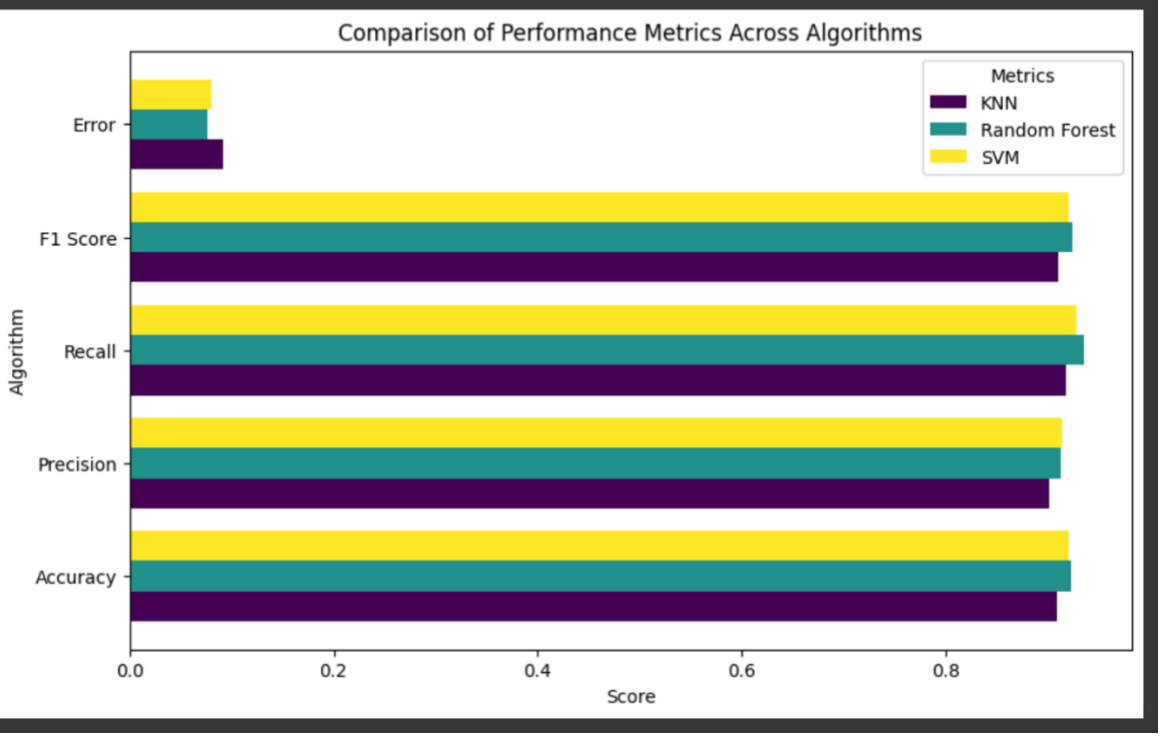


Figure 54: Bar chart displaying overall performance metrics

Among the models, the Random Forest model achieved the highest accuracy, demonstrating its strong performance in identifying URL phishing attacks. Both SVM and KNN performed well, striking a good balance between recall and precision. These results highlight the importance of selecting an algorithm for URL phishing detection that considers the specific requirements and trade-offs involved. The study provides valuable insights into the relative effectiveness of different machine learning models, aiding in the selection of suitable algorithms for real-world cybersecurity challenges.

4. Conclusion

4.1 Summary of Key Findings

This project maiveged to solve one of the biggest problems of identifying URLs which are a security threat as they are used in phishing campaigns, malware distribution, and data theft. The classification of URLs under the categories of phishing and legitimate sites were achieved using the K-Nearest Neighbours (KNN), Support Vector Machines (SVM) and Random Forest techniques. The selected models were tested and compared on a Kaggle dataset with 28 separate features and their performance was evaluated in terms of accuracy, precision, recall and F1 scores catering to the detection of malicious URLs.

4.2 Effectiveness of the Solution

This project's solution provides a practical approach to the problem by using machine learning models that can discern intricate patterns within URL structures. This proactive method not only detects malicious URLs but also helps to mitigate phishing attempts, reducing risks like financial fraud and data theft. The comparative analysis of various algorithms ensured the selection of models best suited for specific use cases, making the system flexible and reliable for real-world applications.

Despite its achievements, the system has certain drawbacks:

- The models depend on pre-existing datasets, which may not fully reflect the constantly changing strategies of cybercriminals.
- Real-time detection can be computationally demanding, posing challenges for systems with limited resources.
- The accuracy of the models hinges on the quality and variety of the dataset, which could limit their ability to detect novel threats.

4.3 Further Work

To overcome the difficulties and enhance the system, mesh-topology-based URL patterns can be analysed using transformers or the more complex recurrent neural networks (RNNs). One more belief, which is behind ensemble methods, is that the results of several algorithms are likely to be more accurate than each of them. Further, integrating real time threat intelligence feeds would assist the system in providing new threats as they are developed. Applying it to other fields of cybersecurity might help to build up the more effective system, for instance, in the case of email filtering or in the case of malware detection.

This project provides a good starting point in enhancing URL detection systems. In the case of the proposed system, the performance and generalizability of the system can be improved if the system is tested with other datasets and scenarios ongoingly. These efforts are crucial to achieve a worthy match with the ever-growing talents in the field of cyber security attacks.

5. Bibliography

Anuganti, S., 2020. *What is a confusion matrix?.* [Online]
Available at: https://medium.com/analytics-vidhya/what-is-a-confusion-matrix-d1c0f8feda5
[Accessed 18 December 2023].

Sahota, N., 2024. *Forbes.* [Online]
Available at: https://www.forbes.com/sites/neilsahota/2024/07/18/how-ai-companions-are-redefining-human-relationships-in-the-digital-age/
[Accessed 10 12 2024].

Google Cloud, 2024. *Google Cloud.* [Online]
Available at: https://cloud.google.com/learn/artificial-intelligence-vs-machine-learning
[Accessed 10 12 2024].

geeksforgeeks, 2024. *geeks for geeks.* [Online]
Available at: https://www.geeksforgeeks.org/machine-learning-algorithms/
[Accessed 11 12 2024].

geeksforgeeks, 2024. *geeks for geeks.* [Online]
Available at: https://www.geeksforgeeks.org/reinforce-algorithm/
[Accessed 11 12 2024].

i2tutorials, 2019. *i2tutorials.* [Online]
Available at: https://www.i2tutorials.com/what-is-the-difference-between-artificial-intelligence-machine-learning-and-deep-learning/
[Accessed 17 12 2024].

mobidev, 2024. [Online]
Available at: https://mobidev.biz/blog/essential-machine-learning-algorithms-for-business-applications
[Accessed 11 12 2024].

Belcic, I., 2024. *IBM.* [Online]
Available at: https://www.ibm.com/think/topics/classification-machine-learning#:~:text=A%20classification%20model%20is%20a,according%20to%20those%

20learned%20characteristics.
[Accessed 15 12 2024].
datacamp, 2024. *datacamp.* [Online]
Available at: https://www.datacamp.com/blog/classification-machine-learning
[Accessed 13 12 2024].
Karabiber, F., 2024. *LearnDataSci.* [Online]
Available at: https://www.learndatasci.com/glossary/binary-classification/
[Accessed 15 12 2024].
Blome, T., 2024. *WorldCrunch.* [Online]
Available at: https://worldcrunch.com/tech-science/the-internet-disappearing-content
[Accessed 15 12 2024].
Petrosyan, A., 2024. *statista.* [Online]
Available at: https://www.statista.com/statistics/1380282/daily-time-spent-online-global/#:~:text=As%20of%20the%20second%20quarter,decrease%20of%20almost%20one%20percent.
[Accessed 15 12 2024].
statista, 2024. *statista.* [Online]
Available at: https://www.statista.com/statistics/266155/number-of-phishing-domain-names-worldwide/
[Accessed 15 12 2024].
Fortinet, 2024. *fortinet.* [Online]
Available at: https://www.fortinet.com/resources/cyberglossary/url-phishing#:~:text=URL%20phishing%20is%20the%20use,login%20credentials%20or%20financial%20information.
[Accessed 15 12 2024].
geeksforgeeks, 2021. *geeksforgeeks.* [Online]
Available at: https://www.geeksforgeeks.org/components-of-a-url/
[Accessed 17 12 2024].
Team, H., 2024. *Hostwinds.* [Online]
Available at: https://www.hostwinds.com/blog/11-parts-of-url-complete-

guide?utm_source=chatgpt.com
[Accessed 17 12 2024].
purecloudsolutions, 2024. *purecloud.* [Online]
Available at: https://www.purecloudsolutions.co.uk/spoofing-attacks-what-they-are-how-they-work/
[Accessed 17 12 2024].
Interisle Consulting Group, 2024. *Cybercrime Information Center.* [Online]
Available at: https://www.cybercrimeinfocenter.org/phishing-trends-november-january-2024
[Accessed 11 12 2024].
Pant, J., 2024. *Bolster.* [Online]
Available at: https://bolster.ai/glossary/brand-impersonation/
[Accessed 17 12 2024].
Abad, S., 2023. *National Center of Biotechnology Information.* [Online]
Available at:
https://pmc.ncbi.nlm.nih.gov/articles/PMC10537824/?utm_source=chatgpt.com
[Accessed 17 12 2024].
geeksforgeeks, 2024. *geeksforgeeks.* [Online]
Available at: https://www.geeksforgeeks.org/top-machine-learning-dataset-find-open-datasets/
[Accessed 17 12 2024].
Abeysooriya, S., 2020. *Kaggle.* [Online]
Available at: https://www.kaggle.com/datasets/sandunabeysooriya/phishing-detection-dataset/data
[Accessed 17 12 2024].
Anshul, 2024. *analyticsvidhya.* [Online]
Available at: https://www.analyticsvidhya.com/blog/2021/10/support-vector-machinessvm-a-complete-guide-for-beginners/
[Accessed 20 12 2024].

Free learning platform for better Future, n.d. *Support Vector Machine Algorithm.* [Online]
Available at: https://www.javatpoint.com/machine-learning-support-vector-machine-algorithm

IBM, 2024. *IBM.* [Online]
Available at: https://www.ibm.com/think/topics/knn#:~:text=The%20k%2Dnearest%20neighbors%20(KNN)%20algorithm%20is%20a%20non,used%20in%20machine%20learning%20today.
[Accessed 17 12 2024].

Vishesh Bharuka, A. A. S. P., 2024. International Journal of Scientific Research in Computer Science, Engineering and Information Technology. *Phishing Detection Using Machine Learning Algorithm,* 10 (2), p. 343.

Hayk Ghalechyan, E. I. A. A. G. H. &. A. D., 2023. Scientific Reports. *Phishing URL detection with neural networks: an empirical study,* 14(2), pp. 200-300.

scikitlearn, 2024. *Metrics and scoring: quantifying the quality of predictions.* [Online]
Available at: https://scikit-learn.org/stable/modules/model_evaluation.html#classification-metrics
[Accessed 20 12 2024].

Crabtree, M., 2024. *What is Machine Learning?.* [Online]
Available at: https://www.datacamp.com/blog/what-is-machine-learning
[Accessed 5 01 2025].

Büber, E., 2025. *Medium.* [Online]
Available at: Phishing URL Detection with ML
[Accessed 03 01 2025].

6. Appendix

6.1 Appendix 1 – Machine Learning

Return to top

The machine learning algorithms and approaches are classified into three types:

i. **Supervised Learning**

It involves training a model using labelled data, where each input comes with a corresponding correct output (geeksforgeeks, 2024). For example: Decision Tree, Random Forest etc.

ii. **Unsupervised Learning**

It entails utilizing unsupervised learning algorithms to train a model that uncovers patterns and relationships within the data, without relying on any prior understanding of its meaning. For example: K-means clustering, Neural networks etc. (geeksforgeeks, 2024)

iii. **Reinforcement Learning**

It involves training a model by combining supervised and unsupervised learning through feedback for actions, exploring alternatives to find the best response. For example: Markov Decision Process. (geeksforgeeks, 2024)

6.2 Appendix 2 – Research work done on the chosen topic/problem domain

6.2.1 URL Phishing Detection

Return to top

The system defines the relationship between the browser and the server, as HTTP and HTTPS are the top and most secure protocols. The domain is the main name of the web site. There are two parts that must be provided. The first one is the subdomain which can be found on the left side of the website address and often indicated by the word "www". The second one is the top-level domain (TLD). Subdirectory stands for the path which goes to a special place on a server. The slug, which usually comes from the page's title, is used to identify a specific page or resource. Parameters, which are sharing the data in a value and name, are useful for customization or retrieval of the specific content in the URL. (Team, 2024).There are three types of URL they are canonical URLs, callback URLs and vanity URLs.

6.2.2 Problems of URL Detection in current scenario

Return to top

- **Unauthorized Access to Accounts**

Individuals may be prompted to reset their password, verify their account, or log in to an apparently legitimate but fraudulent account via a phishing email. The goal of this deceitful approach is for offenders to illegally get legitimate login credentials, allowing them to hack and take control of the targeted account. (Pant, 2024)

- **Direct Financial Loss**

Phishing attacks cause significant financial losses, with hackers using tactics like credential theft and fake invoices to deceive victims. In 2019 alone, the FBI's Internet Crime Complaint Center (IC3) reported $1.7 billion in losses from such attacks—funds organizations could have used for growth, new equipment, or even enhancing employee spaces like chillout corners.

- **Generation of False Information**

Some of the most common ways of fooling people who are the targets of phishing include social engineering strategies. That is, a scammer tells a lie or uses the phishing site to provide them false. In this case, the presence of misguided information on the Internet negatively impacts confidence in online sources and puts the public and individual reputation at risk in some cases.

- **Damage to Brand Reputation**

The phishing URLs associated with well-known companies frequently bring about chaos and making a huge impact stealing the trust that consumers should have in the true enterprise that is masqueraded.

6.3 Appendix 3 – Datasets

Return to top

This dataset was selected for these reasons:

- **Reliable and Trustworthy Source**
  The dataset is mainly from phishtank.com, a well-known platform with a large collection of categorized URLs. Extra features were added using the Kaggle project "Phishing URL Detection with Python and ML". This makes the dataset reliable and useful for real-world phishing detection.

- **Relevant to Phishing Detection**
  The dataset is designed specifically for phishing URL detection, containing features commonly seen in phishing attacks. This ensures the data is closely related to the problem.

- **Large and Varied Dataset**
  It has a big collection of URLs, including both safe and harmful ones. This variety helps in training models to handle different types of phishing attacks.

- **Great for Testing and Comparison**
  The dataset is perfect for testing different machine learning models. It allows you to compare their performance using metrics like accuracy, precision, recall, and F1 score.

- **Detailed Features**
  Each URL comes with important details like SSL status, redirects, URL length, and keywords. These features make the models more effective and accurate.

6.4 Appendix 4 – Development Code

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV, RepeatedStratifiedKFold
from sklearn.metrics import make_scorer, accuracy_score, precision_score, recall_score, f1_score, mean_squared_error, confusion_matrix

# Filtering unnecessary warnings
warnings.filterwarnings('ignore')

# Reading the dataset from a CSV file into a DataFrame
df = pd.read_csv('DataSet.csv')

df.head()

df.tail()

# Displaying the dimensions (number of rows and columns) of the DataFrame
df.shape

# Displaying information about the DataFrame
df.info()
```

```
# Displaying statistics of the DataFrame's numerical columns
df.describe()

# Displaying the number of unique values of each column
df.nunique()

# Checking for missing values and displaying the sum of null values
df.isnull().sum()

# Visualizing the distribution of the target variable 'result'

color = sns.color_palette("muted", 2)  # Changed to 'muted' color palette
sns.countplot(x='result', data=df, palette=color, edgecolor='black')  # Updated edge color to black
plt.title('Distribution of Target Variable (The Results)')
plt.show()

print('The visualization highlights that the dataset is balanced, with an equal number of harmful and harmless URLs.')

# Plotting histograms for each column
df.hist(edgecolor='black', figsize=(20,20))

# Creating a pie chart to visualize the 'abnormal_url' column distribution
df['abnoramal_url'].value_counts().plot(
    kind='pie',
    explode=[0.1, 0.1],  # Adding a slight separation between slices
    autopct='%1.2f%%',  # Displaying percentages with two decimal places
    shadow=True  # Enabling a shadow effect for better visual appeal
)
plt.title("Proportion of Phishing URLs")
```

```
plt.show()

print("The pie chart indicates that the dataset contains a larger proportion of fraudulent URLs.")

# Displaying the correlation matrix to analyze feature relationships
plt.figure(figsize=(12, 10))
sns.heatmap(df.corr(), annot=True, cmap='viridis', fmt=".2f")  # Using 'viridis' for a vibrant color palette
plt.title('Feature Correlation Matrix')
plt.show()

print("The heatmap illustrates the degree of correlation between various features in the dataset.")

# Determining the correlation of each feature with the target variable 'result'
corr_with_target = df.corrwith(df['result'])

# Visualizing the correlation using a heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(
    corr_with_target.sort_values(ascending=False).to_frame(),
    cmap='plasma',  # Updated to use the 'plasma' color palette for a fresh look
    annot=True,
    fmt=".2f"
)
plt.title('Feature Correlation with Target Variable')
plt.show()

print(
    'The analysis reveals that features such as "web_traffic" and "https" '
```

```
    'have a strong relationship with determining whether a URL is malicious.'
)

# Analyzing the distribution of URL lengths using a histogram
plt.figure(figsize=(14, 10))
plt.subplot(2, 2, 1)
sns.histplot(df['length_of_url'], bins=30, kde=True, color='teal')  # Changed color to 'teal'
plt.title('URL Length Distribution')

print(
    'The histogram demonstrates that the majority of URLs are lengthy, '
    'indicating a higher prevalence of fraudulent URLs in the dataset.'
)

# Splitting the dataset into features (X) and the target variable (Y)
X = df.drop(["result"], axis=1)  # Features excluding the target column
Y = df["result"]  # Target variable

# Dividing the dataset into training and testing sets
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=2)  #
20% for testing

# Displaying the dimensions of the training and testing sets
X_train.shape, X_test.shape, Y_train.shape, Y_test.shape

# Setting up the K-Nearest Neighbors (KNN) classifier
knn_model = KNeighborsClassifier()

# Specifying the range of hyperparameter values for tuning
knn_params = {'n_neighbors': [3, 5, 7, 9, 11, 13]}
```

```
# Applying GridSearchCV to identify the optimal hyperparameters
knn_grid_search = GridSearchCV(
    estimator=knn_model,
    param_grid=knn_params,
    refit=True,
    cv=5,  # 5-fold cross-validation
    scoring='accuracy'
)

# Training the KNN model using grid search
knn_grid_search.fit(X_train, Y_train)

# Extracting the optimal hyperparameters
knn_best_params = knn_grid_search.best_params_
print("Optimal KNN Hyperparameters: ", knn_best_params)

# Retrieving the highest cross-validation accuracy achieved by the KNN model
knn_best_score = knn_grid_search.best_score_
print("Best Cross-Validated Score for KNN: ", knn_best_score)

# Creating a new KNN model with the best hyperparameter
knn_best_model                                                    =
KNeighborsClassifier(n_neighbors=list(knn_best_params.values())[0])

# Fitting the best KNN model with the training data
knn_best_model.fit(X_train, Y_train)

# Predicting using the test dataset with the best model
knn_y_pred = knn_best_model.predict(X_test)

# Displaying the accuracy of the optimized KNN model
```

```
knn_accuracy = accuracy_score(Y_test, knn_y_pred)
print(f"KNN Accuracy: {knn_accuracy:.3f}")

# Displaying the precision of the optimized KNN model
knn_precision = precision_score(Y_test, knn_y_pred)
print(f"KNN Precision: {knn_precision:.3f}")

# Displaying the recall of the optimized KNN model
knn_recall = recall_score(Y_test, knn_y_pred)
print(f"KNN Recall: {knn_recall:.3f}")

# Displaying the F1 score of the optimized KNN model
knn_f1 = f1_score(Y_test, knn_y_pred)
print(f"KNN F1 Score: {knn_f1:.3f}")

# Displaying the error rate of the optimized KNN model
knn_error = 1 - knn_accuracy
print(f"KNN Error Rate: {knn_error:.3f}")

# Generating and displaying the confusion matrix for the KNN model
knn_con_matrix = confusion_matrix(Y_test, knn_y_pred)
print(f"Confusion Matrix for KNN Model:\n{knn_con_matrix}")

# Visualizing the confusion matrix using a heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(knn_con_matrix, annot=True, fmt="d", cmap="coolwarm", cbar=False,
        xticklabels=['Predicted Negative', 'Predicted Positive'],
        yticklabels=['Actual Negative', 'Actual Positive'])
plt.title('KNN Confusion Matrix')
plt.xlabel('Predicted Labels')
plt.ylabel('Actual Labels')
```

```
plt.show()

# Setting up the Random Forest (RF) classifier
rf_model = RandomForestClassifier()

# Specifying the hyperparameter grid for tuning the RF model
rf_param_grid = {
    'n_estimators': [50, 100, 150],          # Number of trees in the forest
    'max_depth': [None, 10, 20, 30],         # Maximum depth of each tree
    'min_samples_split': [2, 5, 10],         # Minimum samples required to split an internal
node
    'min_samples_leaf': [1, 2, 4]           # Minimum samples required to be at a leaf node
}

# Setting up GridSearchCV to identify the optimal hyperparameters for the RF model
rf_grid_search = GridSearchCV(
    estimator=rf_model,
    param_grid=rf_param_grid,
    cv=5,
    scoring='accuracy'
)

# Training the Random Forest model
rf_grid_search.fit(X_train, Y_train)

# Retrieving the optimal hyperparameters
rf_best_params = rf_grid_search.best_params_
print("Optimal Hyperparameters for Random Forest: ", rf_best_params)

# Retrieving the highest cross-validation accuracy score achieved
rf_best_score = rf_grid_search.best_score_
```

```
print("Best Cross-Validation Accuracy for Random Forest: ", rf_best_score)

# Random Forest model using the optimal hyperparameters obtained from GridSearchCV
best_rf_model = RandomForestClassifier(
    n_estimators=rf_best_params['n_estimators'],
    max_depth=rf_best_params['max_depth'],
    min_samples_split=rf_best_params['min_samples_split'],
    min_samples_leaf=rf_best_params['min_samples_leaf']
)

# Training the optimized Random Forest model on the training dataset
best_rf_model.fit(X_train, Y_train)

# Making predictions on the test dataset using the trained model
rf_y_pred = best_rf_model.predict(X_test)

# Displaying the accuracy of the Random Forest model
rf_accuracy = accuracy_score(Y_test, rf_y_pred)
print(f"Accuracy of Random Forest Model: {rf_accuracy: .3f}")

# Displaying the precision of the Random Forest model
rf_precision = precision_score(Y_test, rf_y_pred)
print(f"Precision of Random Forest Model: {rf_precision: .3f}")

# Displaying the recall of the Random Forest model
rf_recall = recall_score(Y_test, rf_y_pred)
print(f"Recall of Random Forest Model: {rf_recall: .3f}")

# Displaying the F1 score of the Random Forest model
rf_f1 = f1_score(Y_test, rf_y_pred)
print(f"F1 Score of Random Forest Model: {rf_f1: .3f}")
```

```
# Displaying the error rate of the Random Forest model
rf_error = 1 - rf_accuracy
print(f"Error Rate of Random Forest Model: {rf_error: .3f}")

# Generating a confusion matrix for the Random Forest model
rf_con_matrix = confusion_matrix(y_test, rf_y_pred)

# Visualizing the confusion matrix with a heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(rf_con_matrix, annot=True, fmt="d", cmap="coolwarm", cbar=False,
            xticklabels=['Predicted: Not Harmful', 'Predicted: Harmful'],
            yticklabels=['Actual: Not Harmful', 'Actual: Harmful'])
plt.title('Confusion Matrix - Random Forest Model')
plt.xlabel('Predicted Labels')
plt.ylabel('Actual Labels')
plt.show()

# Initializing the Support Vector Machine (SVM) model
svm_model = SVC()

# Defining a range of hyperparameters for SVM tuning
svm_param_grid = {
    'C': [0.1, 1, 10, 100],  # Regularization parameter
    'kernel': ['linear', 'rbf', 'poly'],  # Different types of kernel functions
    'gamma': ['scale', 'auto']
}

# Setting up GridSearchCV to perform hyperparameter optimization for the SVM model
svm_grid_search = GridSearchCV(estimator=svm_model,
param_grid=svm_param_grid, cv=5, scoring='accuracy')
```

```
# Fitting the SVM grid search to the training data
svm_grid_search.fit(X_train, Y_train)

# Getting the best hyperparameters after the grid search
svm_best_params = svm_grid_search.best_params_
print("SVM Best Hyperparameters: ", svm_best_params)

# Getting the best cross-validation score from the grid search
svm_best_score = svm_grid_search.best_score_
print("SVM Best Score: ", svm_best_score)

# Creating an SVM model with the best hyperparameters
best_svm_model = SVC(
    C=svm_best_params['C'],
    kernel=svm_best_params['kernel'],
    gamma=svm_best_params['gamma']
)

# Training the optimized SVM model with the training dataset
best_svm_model.fit(X_train, Y_train)

# Generating predictions for the test dataset
svm_y_pred = best_svm_model.predict(X_test)

# Evaluating the accuracy of the SVM model
svm_accuracy = accuracy_score(Y_test, svm_y_pred)
print(f"SVM Model Accuracy: {svm_accuracy: .3f}")

# Calculating the precision of the SVM model
svm_precision = precision_score(Y_test, svm_y_pred)
```

```
print(f"SVM Precision Score: {svm_precision: .3f}")

# Calculating the recall of the SVM model
svm_recall = recall_score(Y_test, svm_y_pred)
print(f"SVM Recall Score: {svm_recall: .3f}")

# Calculating the F1 score for the SVM model
svm_f1 = f1_score(Y_test, svm_y_pred)
print(f"SVM F1 Score: {svm_f1: .3f}")

# Calculating the error rate of the SVM model
svm_error = 1 - svm_accuracy
print(f"SVM Model Error Rate: {svm_error: .3f}")

# Generating the confusion matrix for SVM model predictions
svm_con_matrix = confusion_matrix(Y_test, svm_y_pred)

# Plotting the confusion matrix as a heatmap for better visualization
plt.figure(figsize=(8, 6))
sns.heatmap(svm_con_matrix, annot=True, fmt="d", cmap="coolwarm", cbar=False,
            xticklabels=['Predicted 0', 'Predicted 1'],
            yticklabels=['Actual 0', 'Actual 1'])
plt.title('SVM Confusion Matrix')
plt.xlabel('Predicted Values')
plt.ylabel('Actual Values')
plt.show()

# Creating a dictionary to store performance metrics for each algorithm
metrics_dict = {
    'KNN': [knn_accuracy, knn_precision, knn_recall, knn_f1, knn_error],
    'Random Forest': [rf_accuracy, rf_precision, rf_recall, rf_f1, rf_error],
```

```
    'SVM': [svm_accuracy, svm_precision, svm_recall, svm_f1, svm_error]
}

# Converting the dictionary into a DataFrame for easier visualization
metrics_df = pd.DataFrame(metrics_dict, index=['Accuracy', 'Precision', 'Recall', 'F1 Score', 'Error'])

# Displaying the DataFrame
metrics_df

# Plotting a horizontal bar chart to visualize the performance metrics of each algorithm
metrics_df.plot(kind='barh', figsize=(10, 6), colormap='viridis', width=0.8)
plt.title('Comparison of Performance Metrics Across Algorithms')
plt.xlabel('Score')
plt.ylabel('Algorithm')
plt.legend(title='Metrics')
plt.show()
```