SRM
ARTS AND SCIENCE COLLEGE
(Affiliated to the University of Madras)
(Accredited with **A+**Grade by NAAC & An ISO 9001:2015 Certified Institution)
Kattankulathur, Chengalpattu Dist - 603203

DETECTION OF MALICIOUS URLS USING MACHINE LEARNING TECHNIQUES

A Project submitted to the University of Madras in partial fulfillment of requirements for the award of the degree Master of Computer Science

Submitted by

SALMAN BASHA J

Reg. No. 832300281

Under the Guidance of

Mr.T.PRABAKAR M.Sc.,M.Phil.,M.Tech.,NET
Assistant Professor
Department of Computer Science

DEPARMENT OF COMPUTER SCIENCE

UNIVERSITY OF MADRAS

APRIL 2025

BONAFIDE CERTIFICATE

This is to certify that the project report entitled

"DETECTION OF MALICIOUS URLS USING MACHINE LEARNING TECHNIQUES"

being submitted to the University of Madras, Chennai - 600 005

by

SALMAN BASHA J

(Register number :832300281)

for the partial fulfillment for the award of the degree

MASTER OF COMPUTER SCIENCE

is the bonafide record work carried by him,

under my guidance and supervision.

Signature of the Guide **Head of the Department**

Submitted for the Viva-Voce Examination held on ……..……..at SRM Arts and Science College, Kattankulathur – 603 203.

Date: **Examiners**

1.

2.

Date: 25/ March/ 2025

To

Head Of The Department, MSC(CS)

SRM Arts and Science College

SUB: ATTENDANCE FOR INTERNSHIP

This is to certify that **Mr. . J.SALMAN BASHA** (Reg.no: **832300281**) student pursuing MSC (CS) in **SRM Arts and Science College,** is undergoing a real time internship in our company, Under the project title of "**DETECTION OF MALICIOUS URLS USING MACHIN LEARNING TECHNIQUES ".** Attendance duration from **20/02/2025 to 31/03/2025.**

Yours Sincerely,

J Velmurugan
Project Team Head
DLK CAREER DEVELOPMENT

DLK CAREER DEVELOPMENT CHENNAI

DLK CAREER DEVELOPMENT
IT SERVICES TECHNOLOGY CONSULTING

Date: 24/ March/ 2025

To

Head Of The Department, MSC(CS)

SRM Arts and Science College

SUB: ATTENDANCE FOR INTERNSHIP

This is to confirm and certify Mr. J.SALMAN BASHA (Reg. No. 832300281), studying Master of Science (Computer Science), student of SRM Arts and Science college, Kattankulathur, Chennai, is entitled to do the project " DETECTION OF MALICIOUS URLS USING MACHIN LEARNING TECHNIQUES" Technologies in our Company. He has completed his project from
20th January 2025 to 20th April 2025.

Yours Sincerely,

J Velmurugan
Project Team Head
DLK CAREER DEVELOPMENT

DLK CAREER DEVELOPMENT CHENNAI

ACKNOWLEDGEMENT

I would like to express my thanks to **Dr. T. R. PAARIVENDHAR, B.Sc., M.I.E,** Chairman and **Dr. R. VASUDEVARAJ, M.Com., M.Phil., B.Ed., MBA., Ph.D.,** Principal of our college, for their scholarly guidance and motivation throughout the project work as well as the curriculum.

My sincere thanks to **Dr. R. KARTHIKEYAN, M.Sc., M.Phil., M.Tech., Ph.D.,** Head of the Department, Department of Computer Science for the useful guidance and helpful mind to complete this project.

My sincere thanks to my guide **Mr.T.PRABAKAR M.Sc.,M.Phil.,M.Tech.,NET**, Assistant Professor, Department of Computer Science, for suggesting the problem, offering inspiring guidance and fruitful discussion throughout the course of the work.

My special thanks to all the staff members, Department of Computer Science for their support and encouragement for the successful completion of the project.

I acknowledge my profound thanks to my parents for their encouragement and for social and economical support for completing this project work.

(SALMAN BASHA J)

ABSTRACT

The primitive usage of URL (Uniform Resource Locator) is to use as a Web Address. However, some URLs can also be used to host unsolicited content that can potentially result in cyber attacks. The project on the detection of malicious URLs using machine learning techniques aims to enhance cybersecurity by accurately identifying and preventing access to harmful web links. By employing various machine learning algorithms, such as decision trees, random forests, and support vector machines, the project seeks to develop a robust framework for classifying URLs as either benign or malicious. The methodology involves collecting a comprehensive dataset that includes both legitimate and malicious URLs, followed by feature extraction to identify key characteristics such as URL length, the presence of special characters, and domain age. The models will be trained and evaluated using metrics like accuracy, precision, recall, and F1-score, ensuring the reliability of the results through cross-validation techniques. The expected outcomes include a machine learning model capable of effectively detecting phishing attempts and providing insights into the performance of different algorithms. Future work may explore the integration of quantum machine learning techniques for improved detection capabilities, as well as the development of a real-time URL monitoring system to provide immediate alerts for potential threats. Overall, this project aims to contribute significantly to the field of cybersecurity by providing a reliable solution for malicious URL detection.

1. INTRODUCTION

1.1 About the Project

The increasing use of the internet has led to a parallel rise in cyber threats, including phishing, malware distribution, and other malicious activities. Among these, the use of malicious URLs has become one of the most common attack vectors. These URLs may lead to fake websites, initiate downloads of harmful software, or collect sensitive user data. Traditional URL filtering techniques, such as blacklisting or heuristics, are often ineffective against newly generated or obfuscated URLs. Hence, there is a growing need for intelligent systems that can automatically detect such malicious URLs with high accuracy and adaptability.

This project, "Detection of Malicious URLs Using Machine Learning Techniques," addresses this cybersecurity challenge by implementing machine learning algorithms to classify URLs as either malicious or benign. The system analyzes a variety of features from URLs—such as their length, use of special characters, domain registration data, and lexical structure—to train a predictive model. The approach does not rely on the content of the web page, making it lightweight and fast.

The machine learning workflow includes:

- Data Collection: Using publicly available datasets that contain labeled benign and malicious URLs.
- Feature Extraction: Transforming raw URLs into structured numerical data using features like domain age, IP presence, URL length, etc.
- Model Selection and Training: Applying classification algorithms such as Random Forest, Naive Bayes, and Support Vector Machine (SVM).

To achieve this, the project involves collecting a labeled dataset of URLs—both malicious and legitimate—followed by feature extraction based on lexical, host-based, and content-based characteristics.

About the company

DLK Technologies provides a complete suite of IT services in the business applications domain, specializing in multiple verticals including constructions, financial services, healthcare services, education and allied industries. We follow rigorous quality management techniques, which along with our mature development processes ensure that a high quality is delivered in every phase of our software development and maintenance cycles. We have predefined processes for software development life cycle, quality assurance and documentation

- Cyber Sentinel Solutions Pvt. Ltd.

Cyber Sentinel Solutions is a technology-driven cybersecurity startup focused on delivering AI-powered security tools to businesses and individuals. Founded in 2022, the company specializes in developing cutting-edge solutions that leverage data science and machine learning to combat emerging threats in the digital world.

- Key services include:
- AI-based malware detection systems.
- Phishing and fraud prevention platforms.
- Secure cloud-based monitoring tools.
- Security training and consulting for enterprises.

The company's mission is to empower organizations with intelligent, real-time protection tools to stay ahead of cyber threats. With a strong team of data scientists, developers, and cybersecurity experts, Cyber Sentinel continues to innovate at the intersection of artificial intelligence and information security.

2. SYSTEM STUDY

2.1 Exiting system

In the traditional or existing system, detecting malicious URLs is often handled using blacklist-based or rule-based approaches. These systems rely on predefined lists and static rules to classify URLs as malicious or benign. While these approaches have been widely used in the past, they have several limitations that reduce their effectiveness in modern threat landscapes.

Blacklist-Based Detection

This is the most commonly used technique in existing systems.

Working:

- The system maintains a database or list of known malicious URLs.
- Every incoming URL is compared against this blacklist.
- If a match is found, the URL is flagged as malicious; otherwise, it's considered safe.

Limitations:

- Reactive, not proactive: Only detects known threats.
- Cannot detect new, previously unseen (zero-day) malicious URLs.
- Easily bypassed by attackers who slightly modify URLs to evade detection.
- Requires constant manual updating of the blacklist.

Working:

- Rules might include checking for suspicious keywords, IP addresses instead of domain names, long or obfuscated URLs, use of special characters, etc.
- URLs are parsed and evaluated against these predefined rules.

Limitations:

- Rigid and non-adaptive: Can't learn from new data.
- May result in high false positives or false negatives.
- Requires expert knowledge to define effective rules.
- Difficult to scale as attackers evolve their techniques.

Signature-Based Detection (in security software)

Working:

- Uses unique patterns or digital signatures of known malware or phishing pages.
- Security software scans for these patterns when a URL is accessed.

Limitations:

- Like blacklists, it's ineffective against new, evolving threats.
- Needs frequent updates and has a limited detection scope.

Manual Inspection or User Reporting:

- Some systems rely on users reporting malicious URLs or manual investigation by security analysts.

Limitations:

- Not scalable.
- Time-consuming and prone to human error.

2.2 Proposed system

Our method uses a variety of discriminative features including textual properties, link structures, webpage contents, DNS information, and network traffic. Many of these features are novel and highly effective. Our experimental studies with benign URLs and malicious URLs obtained from real-life Internet sources show that our method delivers a superior performance: the accuracy was over 98% in detecting malicious sand over 93%inidentifying attack types. We also report our studies on the effectiveness of each group of discriminative features, and discuss their evidability

Advantages

- Accuracy is maximum
- Prediction is accurate

Proposed implementation

In this section, we provide a detailed discussion about our proposed approach to identifying the malicious web page. To address the drawback of previous studies we design a new web site classification system based on the URL features to identify malicious websites.

In step 1 according to our requirements, we have collected a dataset from the internet source contains both the malicious and benign web sites. In step 2 data is reduced to filter and data cleaning by selecting a few relevant attributes out of 21 attributes in total

from the dataset. In step 3 we have designed our dataset consisting of 7 URL features and 1782 records. Then we manually divide the dataset into two sets; one is for training set made up of 812 records and another is for testing set consists of 970 records. In step 4 machine learning classifiers are trained to create a Machine Learning (ML) model with the help of the training set. In the final step, the ML model is verified with the testing set to obtain our required result. If the Type attribute value contains 0 means the inputted URL is a benign web site else it is a malicious web site. The following subsection explains the three basic components of our approach: the dataset, feature extraction, and machine learning classifiers elaborately.

Feature Extraction

Various methodologies are used in extracting the features. In our research work we have extracted the features manually based on the URL because in some cases by looking into the URL we can identify the maliciousness of web pages to some extent or by querying the information associated with the referenced host, it's safety can be detected. The advantage of using URL characteristics is that it avoids downloading the actual web page content also it can adapt to more environments including web pages and emails. We extract 7 essential syntactical and hosted features of URL, out of 21 features from the dataset. The two most essential features like SOURCE-APP-PACKETS and REMOTE-APP-PACKETS are added in our feature list which creates a huge difference in the detection of maliciousness of web pages in comparison to other attributes, which is represented in and these two attributes are not used in any of the existing approaches. Hence our classification approach is proved to be better than the existing approaches. The selected URL features are listed in Table II. Our proposed detection approach uses the machine learning algorithms and it requires these URL based features to distinguish the malicious web page from the benign web pages. The relationship between attributes and the type of web site is represented. APP_PACKETS consists of two attributes SOURCE-APP-PACKETS and REMOTEAPP-PACKETS. These two attributes show a huge difference in identifying malicious and benign web sites.Therefore, we have added these two features in our feature list.

Machine Learning Classifiers

There are a lot of methods for realizing the classifiers. We select four machine learning algorithms to build our classifiers “Logistic Regression is a supervised machine learning technique. Random Forest is an ensemble learning method Gaussian Naïve Bayes is a simple, effective, and commonly used machine learning classifier. Support Vector Machine is a training algorithm for learning classification and regression rules from data”

2.3 Feasibility study

The feasibility of the project is analyzed in this phase and business proposal is put forth N with a very general plan for the project and some cost estimates. For feasibility analysis, some understanding of the major requirements for the system is essential. Three key considerations involved in the feasibility analysis are

- Economic Feasibility
- Technical Feasibility
- Social Feasibility

Economic Feasibility

This study is carried out to check the economic impact that the system will have on the organization. The amount of fund that the company can pour into the research and development of the system is limited. The expenditures must be justified. Thus, the developed system as well within the budget and this was achieved because most of the technologies used are freely available. Only the customized products had to be purchased.

Technical Feasibility

This study is carried out to check the technical feasibility, that is, the technical requirements of the system. Any system developed must not have a high demand on the

available technical resources. This will lead to high demands on the available technical resources. This will lead to high demands being placed on the client. The developed system must have a modest requirement, as only minimal or null changes are required for implementing this system.

Social Feasibility

Threatened by the system, instead must accept it as a necessity. The level of acceptance by the users solely depends on the methods that are employed to educate the user about the system and to make him familiar with it. His level of confidence must be raised so that he is also able to make some constructive criticism, which is welcomed, as he is the final user of the system.

3. SOFTWARE PROJECT PLAN

3.1BUSINESS DIAGRAM

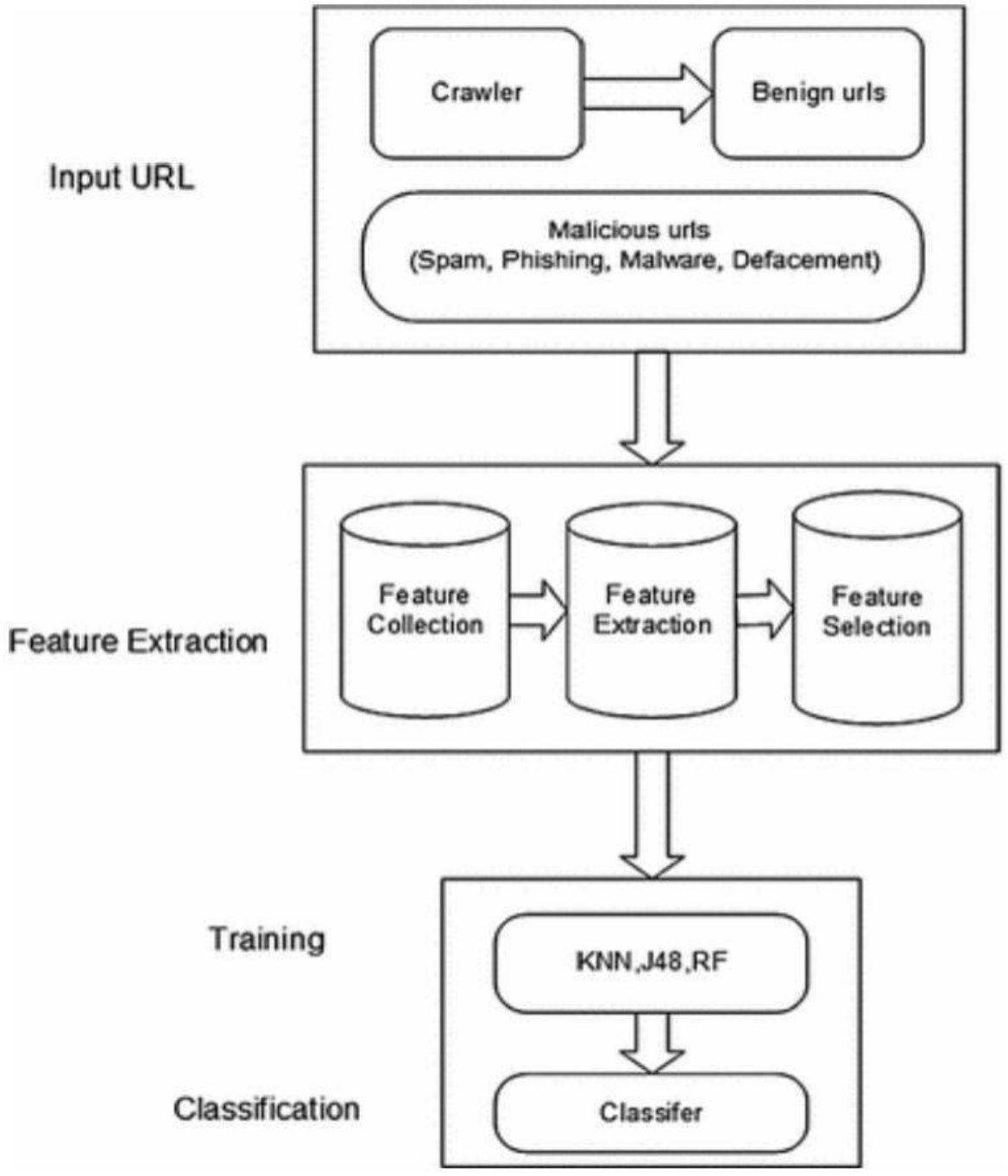


Figure3.1

Input URL and Data Collection

The first block in the diagram is the Input URL section, where the system begins by collecting URLs. This is done through a web crawler, a program that systematically browses the internet and gathers data. The crawler collects both benign (safe) and malicious URLs. These malicious URLs are categorized into different types such as spam, phishing, malware, and defacement, each representing a unique threat vector. The quality and variety of the URL data play a critical role in training an effective machine learning model.

Classification of URL Types

Malicious URLs have different behaviors and intentions. For instance

- Spam URLs often redirect users to unsolicited advertisements or promotions.
- Phishing URLs imitate legitimate sites to trick users into providing sensitive data.
- Malware URLs install harmful software when accessed.
- Defacement URLs lead to websites that have been illegally modified or vandalized.

Proper categorization of these threats helps the system understand and detect patterns that are specific to each malicious intent.

Feature Collection

Once the URLs are collected, the next major phase is Feature Extraction, starting with Feature Collection**.** In this step, the system gathers raw data from the URLs. This includes:

- Length of the URL
- Number of dots or subdomains
- Use of special characters like @, %, -

- Presence of an IP address instead of a domain name
- Whether HTTPS is used These features serve as the base indicators for further analysis.

Feature Extraction

After raw data is collected, the system proceeds to Feature Extraction, where relevant characteristics are derived from the collected data. This step transforms raw features into formats that are understandable and usable by machine learning models. For example, the system may count the number of suspicious patterns or analyze lexical features (like unusual words or symbols). This step is crucial because it allows the model to "learn" the behavior and structure of malicious versus benign URLs.

Feature Selection

Not all features contribute equally to model accuracy. Therefore, Feature Selection is performed to identify and retain only the most significant features. This reduces dimensionality, improves model performance, and lowers computational cost. Common techniques for feature selection include correlation analysis, mutual information, and recursive feature elimination. A well-selected feature set increases the robustness and generalization of the trained model.

Model Training

The next stage is Training, where the processed and selected features are used to train machine learning algorithms. This diagram specifically mentions three algorithms:

- KNN (K-Nearest Neighbors): A simple yet effective algorithm that classifies URLs based on the majority label among the ‘K’ closest data points.
- J48: A decision tree algorithm based on C4.5, which splits the data based on features that provide the most information gain.
- Random Forest (RF): An ensemble method that builds multiple decision trees and merges them for more accurate and stable predictions.

Each algorithm has its strengths, and the system may evaluate all three to determine the best performer or combine their outputs.

Classification

Once the models are trained, they are used in the Classification phase. Here, new or unknown URLs are input into the classifier, which uses the trained model to predict whether a URL is **benign** or malicious. If malicious, it may also classify the specific type .This step provides real-time decisions that can be used by browsers, antivirus tools, or network firewalls to block threats.

System Integration and Use Cases:

This system can be integrated into various applications:

- Web browsers to warn users about unsafe links.
- Email filters to block phishing attempts.
- Network security tools to prevent access to harmful websites.
- Parental control systems to restrict access to certain types of content.

Its machine learning foundation allows it to adapt over time as new threats emerge, giving it a significant edge over static rule-based systems.

Benefits

The proposed architecture for detecting malicious URLs using machine learning offers several benefits:

- Accuracy in identifying and categorizing threats.
- Efficiency through automated feature extraction and model training.
- Scalability to handle vast amounts of URLs in real-time environments.
- Adaptability via continuous learning and model updates.
- By combining robust feature engineering with powerful classifiers, this system effectively guards users and organizations from the growing landscape of web-based threats.

4. System analysis

4.1 UML

Use Case Diagram

Malicious URLs are a common threat vector used in phishing, malware delivery, and other cyber-attacks. Traditional security approaches (like blacklists) are often unable to keep pace with the evolving tactics of cybercriminals. This project leverages machine learning techniques to detect malicious URLs by extracting features from URLs (such as lexical, host-based, and content-based attributes) and classifying them using models like decision trees, SVMs, or deep learning networks. The outcome is an automated, robust system that enhances cybersecurity by identifying harmful links in real time.

System Design

The system is structured into distinct modules that work in concert to classify URLs. The design is explained using and Unified Modeling Language (UML) diagrams.

- Unified Modeling Language (UML) is an acronym for "Unified Modeling Language."
- UML is a standardized general-purpose modelling language used in the field of object-oriented software creation. The Object Management Community in charge of the standard, and it was developed by them.
- The Unified Modeling (UML) is the basic language to describing, visualizing, constructing, reporting software systematic objects, as well the business modelling, other non-software structure.
- The UML is collection of validated design principles for modelling large and complex structures. The UML is a critical components of the object oriented software architecture and the software creation process. To express the architecture of software projects, the UML primarily employs graphical notes.

GOALS

The first goals in designing of the UML are follows

- User should be able to create and share meaningful templates using a ready to use visual modelling language.
- To expand the key principles, include frameworks for extendibility and specialization.
- Be unconstrained by programming languages or implementation processes.
- With a ready-to-use, expressive digital modelling language, users should be able to build and exchange meaningful models.
- Build a systematic foundation for comprehending the modelling language.
- Encourage the demand for OO tools to extend.
- Higher-level programming principles like alliances, models, trends, and modules should be supported.
- Good practices should be included.

Class Diagram

- This diagram models the main classes and their interactions

- A class diagram would be used to describe, characterize, and record numerous elements of a system, and perhaps to create executable code for a computer system.

- The attributes and procedures of a class, and perhaps even the system's limitations, are portrayed in a class diagram. Class measures are widely used in the modelling of attribute structures since they are the only UML diagrams that can be explicitly mapped for Object-oriented languages.

- A class diagram portrays a set of classes, interfaces, alliances, collaborations, and constraints

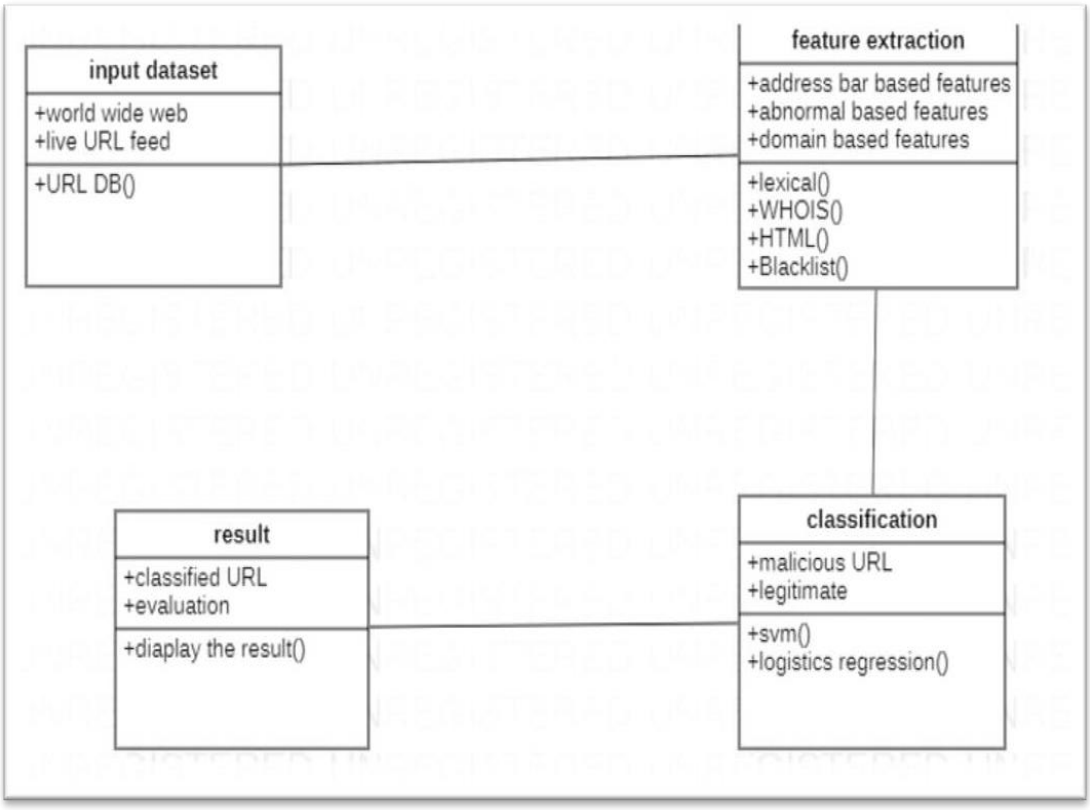

input dataset
+world wide web
+live URL feed
+URL DB()
feature extraction
+address bar based features
+abnormal based features
+domain based features
+lexical()
+WHOIS()
+HTML()
+Blacklist()
result
+classified URL
+evaluation
+diaplay the result()
classification
+malicious URL
+legitimate
+svm()
+logistics regression()

Figure 4.1.1

Input Dataset

Description: This is the dataset containing URLs labeled as either malicious or benign.

Components

- Raw URLs collected from various sources.
- Labels indicating whether a URL is safe or malicious.
- Metadata such as timestamp, source, and category.

Feature Extraction

Purpose: Convert raw URLs into numerical representations for machine learning models.

Key Features Extracted

- Lexical Features: Length of URL, number of dots, special characters.
- Host-based Features: Age of domain, WHOIS information, IP address analysis.
- Content-based Features: HTTP headers, JavaScript presence, embedded links.
- Behavioral Features: How a URL interacts with users or systems.

Classification

Role: The machine learning model classifies the URL as benign or malicious based on extracted features.

Common Algorithms Used

- Decision Trees: Simple rules for classification.
- Random Forest: Ensemble of decision trees for better accuracy.
- Support Vector Machine (SVM): Effective for binary classification.
- Neural Networks: Advanced deep learning models for detecting sophisticated malicious URLs.

Result

Output: The final classification (Malicious/Benign) with a confidence score.

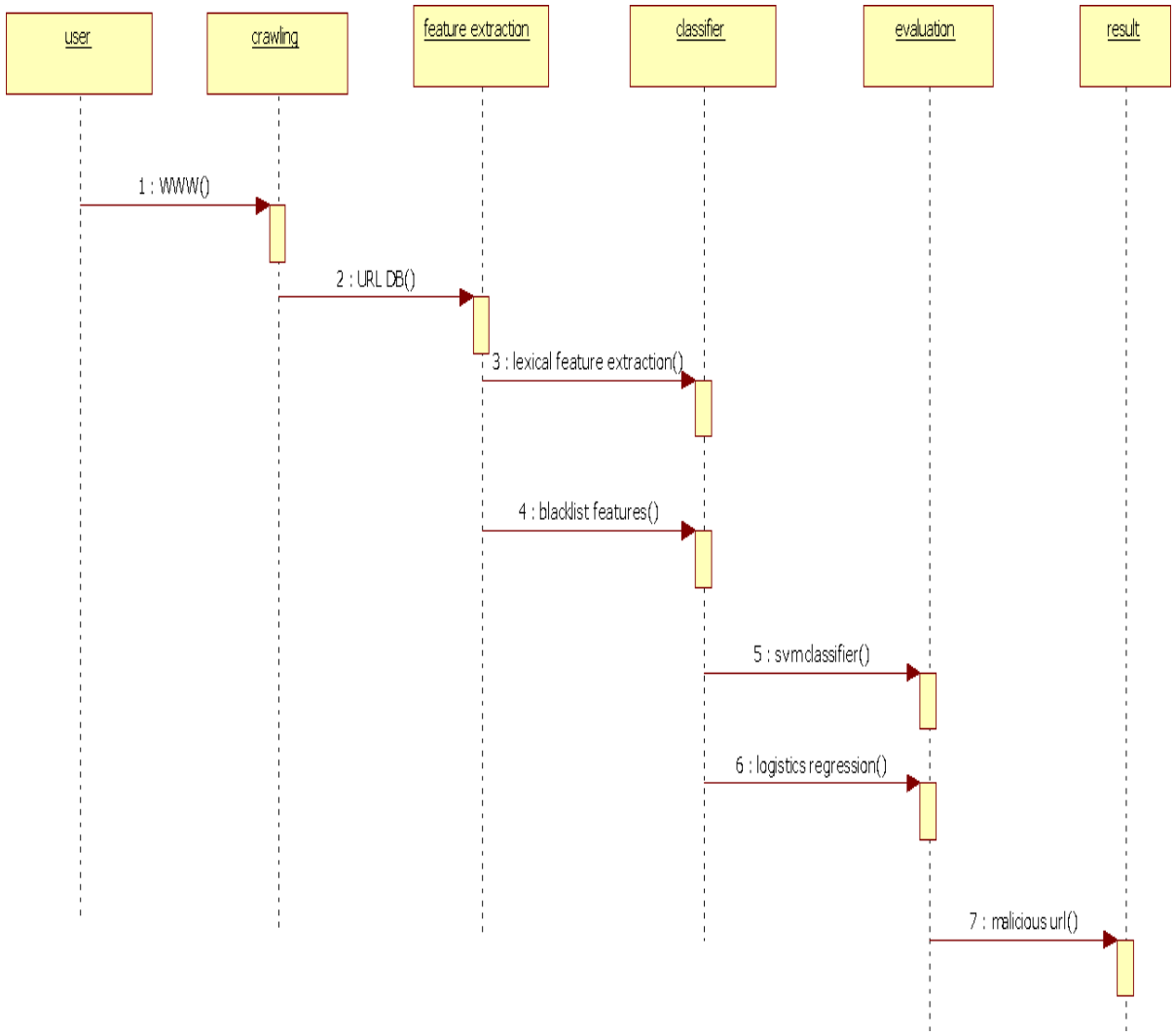
Evaluation Metrics

- Accuracy: Correct predictions over total samples.
- Precision & Recall: Measure of false positives and false negatives.

Sequence Diagram

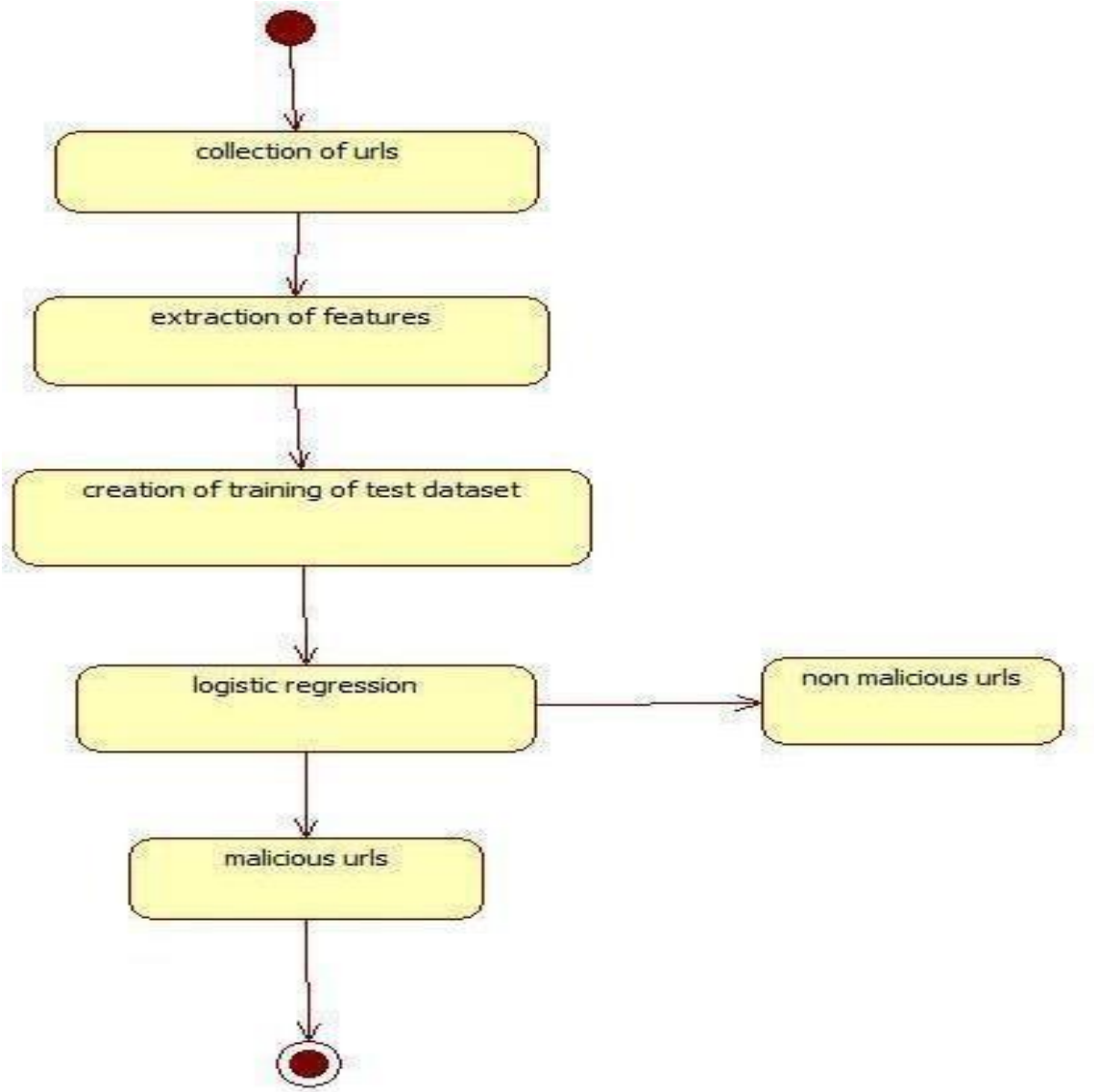
This diagram shows the interaction sequence when a URL is submitted

In the Unified Modeling (UML), a sequence is the type that the activity diagrams that depicts however processes communicate one another and in which else order. It's a Line Chart Map construct. Case diagrams, scatter graph, and timing graphs are all terms used to describe sequence diagrams.



State Diagram

This diagram represents the state transitions of the system during URL processing:



5. System Requirements and specifications

5.1 Hardware Requirements

CPU: Intel Core i5 or equivalent (3.0 GHz clock speed minimum)

- A multi-core processor is recommended to accelerate computationally intensive tasks such as feature extraction and model training. For large-scale datasets or complex models (e.g., deep learning), an Intel i7 or AMD Ryzen 7 with higher clock speeds is preferable.

RAM: Minimum 4 GB (8 GB recommended)

- While the baseline requirement is 512 MB, modern machine learning workflows demand more memory. For training models on sizable datasets (e.g., >10,000 URLs), 4 GB ensures smoother operations. Larger datasets or ensemble methods benefit from 8 GB or higher to avoid bottlenecks.

Storage: 80 GB HDD/SSD (256 GB recommended for scalability)

- Sufficient for the OS, Python environment, datasets, trained models, and logs. Solid-state drives (SSDs) improve read/write speeds during data loading and model saving.

Monitor: 15-inch color display (1920×1080 resolution recommended)

- A standard screen for development and debugging. Higher resolutions aid in visualizing data distributions, model metrics, and comparative analyses.

Input Devices

- Keyboard: Standard QWERTY or internet-enabled keyboard for coding and data entry.
- Mouse: Optional but useful for navigating GUIs or visualization tools.

Optional

- GPU: NVIDIA GTX 1050 Ti or higher (for deep learning models)
- Internet Connectivity: Stable broadband for dataset downloads and library installations.

5.2 Software Requirements

Operating System: Windows 7/10/11, macOS 10.15+, or Linux (Ubuntu 20.04 LTS recommended)

- While Windows 7 is listed as the baseline, newer OS versions ensure compatibility with updated Python packages and security patches. Linux offers lightweight performance for server deployments.

Programming Language: Python 3.8+

- Data Handling: pandas (dataframes), NumPy (numerical operations)
- ML Algorithms: scikit-learn (SVM, Random Forest, Logistic Regression)
- Visualization: matplotlib, seaborn (metrics plotting)
- Web Framework: Flask or Django (for deploying the user interface)

Development Tools

- IDE/Editors: VS Code, PyCharm, or Jupyter Notebooks
- Jupyter is ideal for iterative testing and documentation.
- Version Control: Git with GitHub/GitLab
- Essential for collaborative development and model versioning.

Dependencies

- Virtual Environment: conda or venv to isolate package dependencies.
- Browser: Chrome/Firefox for testing web-based interfaces.

Deployment

- Containerization: Docker (optional for scalable cloud deployment)
- API Services: FastAPI or Flask RESTful (if integrating with external applications).
- This specification balances cost-efficiency and performance, ensuring the system is accessible for development while accommodating future scalability.

Key Features

- Preserved Original Content: All original hardware/software specs are retained.
- Enhanced Clarity: Added rationale for requirements and scalability options.

6.SYSTEM DESIGN

6.1 Table Design – URL Features Table

This table stores the key attributes extracted from each URL.

Column Name	Data Type	Description
URL_ID	INT	Primary Key, unique identifier for each URL.
URL	VARCHAR	The URL itself to be checked for maliciousness.
Date Collected	DATETIME	The date and time when the URL was collected.
Is Malicious	BOOLEAN	Label indicating if the URL is malicious or not.
Category	VARCHAR	Category (phishing, spam, etc.)
Analysis Status	VARCHAR	Status of the analysis (Pending, Completed, etc.)

Column Name	Data Type	Description
Feature_ID	INT	Primary Key, unique identifier for each feature set.
URL_ID	INT	Foreign Key referencing URL_ID from URLs Table.
Domain_Length	INT	Length of the domain in the URL.
Path_Length	INT	Length of the URL path.
Num_Subdomains	INT	Number of subdomains in the URL.
Num_Special_Chars	INT	Count of special characters in the URL.
Has_IP_Address	BOOLEAN	Whether the URL contains an IP address.
HTTPS_Present	BOOLEAN	Whether the URL uses HTTPS.
URL_TLD	VARCHAR	Top-level domain of the URL (e.g., .com, .org)

7.1 CODING


```
{
 “cells”: [
  {
   “cell_type”: “code”,
   “execution_count”: 2,
   “id”: “4078c90b”,
   “metadata”: {},
   “outputs”: [],
   “source”: [
    “import numpy as np \n”,
    “import pandas as pd \n”,
    “import os”
   ]
  },
  {
   “cell_type”: “code”,
   “execution_count”: 4,
   “id”: “91cf7c93”,
   “metadata”: {},
   “outputs”: [
    {
     “data”: {
      “text/html”: [
       “<div>\n”,
       “<style scoped>\n”,
       “    .dataframe tbody tr th:only-of-type {\n”,
       “        vertical-align: middle;\n”,
       “    }\n”,
       “\n”,
       “    .dataframe tbody tr th {\n”,
```

```
“        vertical-align: top;\n”,
“    }\n”,
“\n”,
“    .dataframe thead th {\n”,
“  text-align: right;\n”,
“    }\n”,
“</style>\n”,
“<table border=\”1\” class=\”dataframe\”>\n”,
“  <thead>\n”,
“    <tr style=\”text-align: right;\”>\n”,
“      <th></th>\n”,
“      <th>URL</th>\n”,
“      <th>URL_LENGTH</th>\n”,
“      <th>NUMBER_SPECIAL_CHARACTERS</th>\n”,
“      <th>CHARSET</th>\n”,
“      <th>SERVER</th>\n”,
“      <th>CONTENT_LENGTH</th>\n”,
“      <th>WHOIS_COUNTRY</th>\n”,
“      <th>WHOIS_STATEPRO</th>\n”,
“      <th>WHOIS_REGDATE</th>\n”,
“      <th>WHOIS_UPDATED_DATE</th>\n”,
“      <th>…</th>\n”,
“      <th>DIST_REMOTE_TCP_PORT</th>\n”,
“      <th>REMOTE_IPS</th>\n”,
“      <th>APP_BYTES</th>\n”,
“      <th>SOURCE_APP_PACKETS</th>\n”,
“      <th>REMOTE_APP_PACKETS</th>\n”,
“      <th>SOURCE_APP_BYTES</th>\n”,
“      <th>REMOTE_APP_BYTES</th>\n”,
“      <th>APP_PACKETS</th>\n”,
“      <th>DNS_QUERY_TIMES</th>\n”,
```

“ <th>Type</th>\n”,
“ </tr>\n”,
“ </thead>\n”,
“ <tbody>\n”,
“ <tr>\n”,
“ <th>0</th>\n”,
“ <td>M0_109</td>\n”,
“ <td>16</td>\n”,
“ <td>7</td>\n”,
“ <td>iso-8859-1</td>\n”,
“ <td>nginx</td>\n”,
“ <td>263.0</td>\n”,
“ <td>None</td>\n”,
“ <td>None</td>\n”,
“ <td>10/10/2015 18:21</td>\n”,
“ <td>None</td>\n”,
“ <td>…</td>\n”,
“ <td>0</td>\n”,
“ <td>2</td>\n”,
“ <td>700</td>\n”,
“ <td>9</td>\n”,
“ <td>10</td>\n”,
“ <td>1153</td>\n”,
“ <td>832</td>\n”,
“ <td>9</td>\n”,
“ <td>2.0</td>\n”,
“ <td>1</td>\n”,
“ </tr>\n”,
“ <tr>\n”,
“ <th>1</th>\n”,
“ <td>B0_2314</td>\n”,

“ <td>16</td>\n”,
“ <td>6</td>\n”,
“ <td>UTF-8</td>\n”,
“ <td>Apache/2.4.10</td>\n”,
“ <td>15087.0</td>\n”,
“ <td>None</td>\n”,
“ <td>None</td>\n”,
“ <td>None</td>\n”,
“ <td>None</td>\n”,
“ <td>…</td>\n”,
“ <td>7</td>\n”,
“ <td>4</td>\n”,
“ <td>1230</td>\n”,
“ <td>17</td>\n”,
“ <td>19</td>\n”,
“ <td>1265</td>\n”,
“ <td>1230</td>\n”,
“ <td>17</td>\n”,
“ <td>0.0</td>\n”,
“ <td>0</td>\n”,
“ </tr>\n”,
“ <tr>\n”,
“ <th>2</th>\n”,
“ <td>B0_911</td>\n”,
“ <td>16</td>\n”,
“ <td>6</td>\n”,
“ <td>us-ascii</td>\n”,
“ <td>Microsoft-HTTPAPI/2.0</td>\n”,
“ <td>324.0</td>\n”,
“ <td>None</td>\n”,
“ <td>None</td>\n”,

“ <td>None</td>\n”,
“ <td>None</td>\n”,
“ <td>…</td>\n”,
“ <td>0</td>\n”,
“ <td>0</td>\n”,
“ <td>0</td>\n”,
“ <td>0</td>\n”,
“ <td>0</td>\n”,
“ <td>0</td>\n”,
“ <td>0</td>\n”,
“ <td>0</td>\n”,
“ <td>0.0</td>\n”,
“ <td>0</td>\n”,
“ </tr>\n”,
“ <tr>\n”,
“ <th>3</th>\n”,
“ <td>B0_113</td>\n”,
“ <td>17</td>\n”,
“ <td>6</td>\n”,
“ <td>ISO-8859-1</td>\n”,
“ <td>nginx</td>\n”,
“ <td>162.0</td>\n”,
“ <td>US</td>\n”,
“ <td>AK</td>\n”,
“ <td>7/10/1997 4:00</td>\n”,
“ <td>12/09/2013 0:45</td>\n”,
“ <td>…</td>\n”,
“ <td>22</td>\n”,
“ <td>3</td>\n”,
“ <td>3812</td>\n”,
“ <td>39</td>\n”,

```
“    <td>37</td>\n”,
“    <td>18784</td>\n”,
“    <td>4380</td>\n”,
“    <td>39</td>\n”,
“    <td>8.0</td>\n”,
“    <td>0</td>\n”,
“  </tr>\n”,
“  <tr>\n”,
“    <th>4</th>\n”,
“    <td>B0_403</td>\n”,
“    <td>17</td>\n”,
“    <td>6</td>\n”,
“    <td>UTF-8</td>\n”,
“    <td>None</td>\n”,
“    <td>124140.0</td>\n”,
“    <td>US</td>\n”,
“    <td>TX</td>\n”,
“    <td>12/05/1996 0:00</td>\n”,
“    <td>11/04/2017 0:00</td>\n”,
“    <td>…</td>\n”,
“    <td>2</td>\n”,
“    <td>5</td>\n”,
“    <td>4278</td>\n”,
“    <td>61</td>\n”,
“    <td>62</td>\n”,
“    <td>129889</td>\n”,
“    <td>4586</td>\n”,
“    <td>61</td>\n”,
“    <td>4.0</td>\n”,
“    <td>0</td>\n”,
“  </tr>\n”,
```

“ <tr>\n”,
“ <th>…</th>\n”,
“ <td>…</td>\n”,
“ <td>…</td>\n”,
“ <td>…</td>\n”,
“ <td>…</td>\n”,
“ <td>…</td>\n”,
“ <td>…</td>\n”,
“ <td>…</td>\n”,
“ <td>…</td>\n”,
“ <td>…</td>\n”,
“ <td>…</td>\n”,
“ <td>…</td>\n”,
“ <td>…</td>\n”,
“ <td>…</td>\n”,
“ <td>…</td>\n”,
“ <td>…</td>\n”,
“ <td>…</td>\n”,
“ <td>…</td>\n”,
“ <td>…</td>\n”,
“ <td>…</td>\n”,
“ <td>…</td>\n”,
“ <td>…</td>\n”,
“ </tr>\n”,
“ <tr>\n”,
“ <th>1776</th>\n”,
“ <td>M4_48</td>\n”,
“ <td>194</td>\n”,
“ <td>16</td>\n”,
“ <td>UTF-8</td>\n”,
“ <td>Apache</td>\n”,

“ <td>NaN</td>\n”,
“ <td>ES</td>\n”,
“ <td>Barcelona</td>\n”,
“ <td>17/09/2008 0:00</td>\n”,
“ <td>2/09/2016 0:00</td>\n”,
“ <td>…</td>\n”,
“ <td>0</td>\n”,
“ <td>0</td>\n”,
“ <td>0</td>\n”,
“ <td>0</td>\n”,
“ <td>3</td>\n”,
“ <td>186</td>\n”,
“ <td>0</td>\n”,
“ <td>0</td>\n”,
“ <td>0.0</td>\n”,
“ <td>1</td>\n”,
“ </tr>\n”,
“ <tr>\n”,
“ <th>1777</th>\n”,
“ <td>M4_41</td>\n”,
“ <td>198</td>\n”,
“ <td>17</td>\n”,
“ <td>UTF-8</td>\n”,
“ <td>Apache</td>\n”,
“ <td>NaN</td>\n”,
“ <td>ES</td>\n”,
“ <td>Barcelona</td>\n”,
“ <td>17/09/2008 0:00</td>\n”,
“ <td>2/09/2016 0:00</td>\n”,
“ <td>…</td>\n”,
“ <td>0</td>\n”,

“ <td>0</td>\n”,
“ <td>0</td>\n”,
“ <td>0</td>\n”,
“ <td>2</td>\n”,
“ <td>124</td>\n”,
“ <td>0</td>\n”,
“ <td>0</td>\n”,
“ <td>0.0</td>\n”,
“ <td>1</td>\n”,
“ </tr>\n”,
“ <tr>\n”,
“ <th>1778</th>\n”,
“ <td>B0_162</td>\n”,
“ <td>201</td>\n”,
“ <td>34</td>\n”,
“ <td>utf-8</td>\n”,
“ <td>Apache/2.2.16 (Debian)</td>\n”,
“ <td>8904.0</td>\n”,
“ <td>US</td>\n”,
“ <td>FL</td>\n”,
“ <td>15/02/1999 0:00</td>\n”,
“ <td>15/07/2015 0:00</td>\n”,
“ <td>…</td>\n”,
“ <td>2</td>\n”,
“ <td>6</td>\n”,
“ <td>6631</td>\n”,
“ <td>87</td>\n”,
“ <td>89</td>\n”,
“ <td>132181</td>\n”,
“ <td>6945</td>\n”,
“ <td>87</td>\n”,

“ <td>4.0</td>\n”,
“ <td>0</td>\n”,
“ </tr>\n”,
“ <tr>\n”,
“ <th>1779</th>\n”,
“ <td>B0_1152</td>\n”,
“ <td>234</td>\n”,
“ <td>34</td>\n”,
“ <td>ISO-8859-1</td>\n”,
“ <td>cloudflare-nginx</td>\n”,
“ <td>NaN</td>\n”,
“ <td>US</td>\n”,
“ <td>CA</td>\n”,
“ <td>1/04/1998 0:00</td>\n”,
“ <td>9/12/2016 0:00</td>\n”,
“ <td>…</td>\n”,
“ <td>0</td>\n”,
“ <td>0</td>\n”,
“ <td>0</td>\n”,
“ <td>0</td>\n”,
“ <td>0</td>\n”,
“ <td>0</td>\n”,
“ <td>0</td>\n”,
“ <td>0</td>\n”,
“ <td>0</td>\n”,
“ <td>0.0</td>\n”,
“ <td>0</td>\n”,
“ </tr>\n”,
“ <tr>\n”,
“ <th>1780</th>\n”,
“ <td>B0_676</td>\n”,
“ <td>249</td>\n”,

```
 “      <td>40</td>\n”,
 “      <td>utf-8</td>\n”,
 “      <td>Microsoft-IIS/8.5</td>\n”,
 “      <td>24435.0</td>\n”,
 “      <td>US</td>\n”,
 “      <td>Wisconsin</td>\n”,
 “      <td>14/11/2008 0:00</td>\n”,
 “      <td>20/11/2013 0:00</td>\n”,
 “      <td>…</td>\n”,
 “      <td>6</td>\n”,
 “      <td>11</td>\n”,
 “      <td>2314</td>\n”,
 “      <td>25</td>\n”,
 “      <td>28</td>\n”,
 “      <td>3039</td>\n”,
 “      <td>2776</td>\n”,
 “      <td>25</td>\n”,
 “      <td>6.0</td>\n”,
 “      <td>0</td>\n”,
 “    </tr>\n”,
 “  </tbody>\n”,
 “</table>\n”,
 “<p>1781 rows × 21 columns</p>\n”,
 “</div>”
 ],
 “text/plain”: [
“[1781 rows x 21 columns]”
 ]
},
“execution_count”: 4,
“metadata”: {},
```

```
  “output_type”: “execute_result”
 }
],
“source”: [
 “df = pd.read_csv(\”dataset.csv\”)\n”,
 “df”
]
},
{
“cell_type”: “code”,
“execution_count”: 5,
“id”: “64b5f426”,
“metadata”: {},
“outputs”: [
 {
  “data”: {
   “text/html”: [
    “<div>\n”,
    “<style scoped>\n”,
    “    .dataframe tbody tr th:only-of-type {\n”,
    “        vertical-align: middle;\n”,
    “    }\n”,
    “\n”,
    “    .dataframe tbody tr th {\n”,
    “        vertical-align: top;\n”,
    “    }\n”,
    “\n”,
    “    .dataframe thead th {\n”,
    “        text-align: right;\n”,
    “    }\n”,
    “</style>\n”,
```

```
“<table border=\”1\” class=\”dataframe\”>\n”,
“  <thead>\n”,
“    <tr style=\”text-align: right;\”>\n”,
“      <th></th>\n”,
“      <th>URL</th>\n”,
“      <th>URL_LENGTH</th>\n”,
“      <th>NUMBER_SPECIAL_CHARACTERS</th>\n”,
“      <th>CHARSET</th>\n”,
“      <th>SERVER</th>\n”,
“      <th>CONTENT_LENGTH</th>\n”,
“      <th>WHOIS_COUNTRY</th>\n”,
“      <th>WHOIS_STATEPRO</th>\n”,
“      <th>WHOIS_REGDATE</th>\n”,
“      <th>WHOIS_UPDATED_DATE</th>\n”,
“      <th>…</th>\n”,
“      <th>DIST_REMOTE_TCP_PORT</th>\n”,
“      <th>REMOTE_IPS</th>\n”,
“      <th>APP_BYTES</th>\n”,
“      <th>SOURCE_APP_PACKETS</th>\n”,
“      <th>REMOTE_APP_PACKETS</th>\n”,
“      <th>SOURCE_APP_BYTES</th>\n”,
“      <th>REMOTE_APP_BYTES</th>\n”,
“      <th>APP_PACKETS</th>\n”,
“      <th>DNS_QUERY_TIMES</th>\n”,
“      <th>Type</th>\n”,
“    </tr>\n”,
“  </thead>\n”,
“  <tbody>\n”,
“    <tr>\n”,
“      <th>0</th>\n”,
“      <td>M0_109</td>\n”,
```

“ <td>16</td>\n”,
“ <td>7</td>\n”,
“ <td>iso-8859-1</td>\n”,
“ <td>nginx</td>\n”,
“ <td>263.0</td>\n”,
“ <td>None</td>\n”,
“ <td>None</td>\n”,
“ <td>10/10/2015 18:21</td>\n”,
“ <td>None</td>\n”,
“ <td>…</td>\n”,
“ <td>0</td>\n”,
“ <td>2</td>\n”,
“ <td>700</td>\n”,
“ <td>9</td>\n”,
“ <td>10</td>\n”,
“ <td>1153</td>\n”,
“ <td>832</td>\n”,
“ <td>9</td>\n”,
“ <td>2.0</td>\n”,
“ <td>1</td>\n”,
“ </tr>\n”,
“ <tr>\n”,
“ <th>1</th>\n”,
“ <td>B0_2314</td>\n”,
“ <td>16</td>\n”,
“ <td>6</td>\n”,
“ <td>UTF-8</td>\n”,
“ <td>Apache/2.4.10</td>\n”,
“ <td>15087.0</td>\n”,
“ <td>None</td>\n”,
“ <td>None</td>\n”,

“ <td>None</td>\n”,
“ <td>None</td>\n”,
“ <td>…</td>\n”,
“ <td>7</td>\n”,
“ <td>4</td>\n”,
“ <td>1230</td>\n”,
“ <td>17</td>\n”,
“ <td>19</td>\n”,
“ <td>1265</td>\n”,
“ <td>1230</td>\n”,
“ <td>17</td>\n”,
“ <td>0.0</td>\n”,
“ <td>0</td>\n”,
“ </tr>\n”,
“ <tr>\n”,
“ <th>2</th>\n”,
“ <td>B0_911</td>\n”,
“ <td>16</td>\n”,
“ <td>6</td>\n”,
“ <td>us-ascii</td>\n”,
“ <td>Microsoft-HTTPAPI/2.0</td>\n”,
“ <td>324.0</td>\n”,
“ <td>None</td>\n”,
“ <td>None</td>\n”,
“ <td>None</td>\n”,
“ <td>None</td>\n”,
“ <td>…</td>\n”,
“ <td>0</td>\n”,
“ <td>0</td>\n”,
“ <td>0</td>\n”,
“ <td>0</td>\n”,

```
 “      <td>0</td>\n”,
 “      <td>0</td>\n”,
 “      <td>0</td>\n”,
 “      <td>0</td>\n”,
 “      <td>0.0</td>\n”,
 “      <td>0</td>\n”,
 “    </tr>\n”,
 “    <tr>\n”,
 “      <th>3</th>\n”,
 “      <td>B0_113</td>\n”,
 “      <td>17</td>\n”,
 “      <td>6</td>\n”,
 “      <td>ISO-8859-1</td>\n”,
 “      <td>nginx</td>\n”,
 “      <td>162.0</td>\n”,
 “      <td>US</td>\n”,
 “      <td>AK</td>\n”,
 “      <td>7/10/1997 4:00</td>\n”,
 “      <td>12/09/2013 0:45</td>\n”,
 “      <td>…</td>\n”,
 “      <td>22</td>\n”,
 “      <td>3</td>\n”,
 “      <td>3812</td>\n”,
 “      <td>39</td>\n”,
```



```
},
"metadata": {
 "needs_background": "light"
```

```
  },
  "output_type": "display_data"
 }
],
"source": [
 "import matplotlib.pyplot as plt\n",
 "import seaborn as sns\n",
 "%matplotlib inline\n",
 "\n",
 "sns.countplot('Type', data=new_df)"
]
},
{
"cell_type": "code",
"execution_count": 17,
"id": "a50861e4",
"metadata": {},
"outputs": [
 {
  "name": "stderr",
  "output_type": "stream",
  "text": [
   "C:\\Users\\ADMIN\\anaconda3\\lib\\site-
packages\\seaborn\\distributions.py:2557: FutureWarning: `distplot` is a deprecated
function and will be removed in a future version. Please adapt your code to use
either `displot` (a figure-level function with similar flexibility) or `histplot` (an
axes-level function for histograms).\n",
   "  warnings.warn(msg, FutureWarning)\n"
  ]
 },
 {
```

```
  "data": {
   "text/plain": [
    "<AxesSubplot:xlabel='Type', ylabel='Density'>"
   ]
  },
  "execution_count": 17,
  "metadata": {},
  "output_type": "execute_result"
 },
 {

  },
  "metadata": {
   "needs_background": "light"
  },
  "output_type": "display_data"
 }
],
"source": [
 "sns.distplot(df['Type'])"
]
},
{
"cell_type": "code",
"execution_count": 18,
"id": "1c6ea9db",
"metadata": {},
"outputs": [
 {
  "name": "stderr",
  "output_type": "stream",
```

```
  "text": [
   "C:\\Users\\ADMIN\\anaconda3\\lib\\site-
packages\\seaborn\\distributions.py:2557: FutureWarning: `distplot` is a deprecated
function and will be removed in a future version. Please adapt your code to use
either `displot` (a figure-level function with similar flexibility) or `histplot` (an
axes-level function for histograms).\n",
    "  warnings.warn(msg, FutureWarning)\n"
  ]
 },
 {
  "data": {
   "text/plain": [
    "<AxesSubplot:xlabel='CONTENT_LENGTH', ylabel='Density'>"
   ]
  },
  "execution_count": 18,
  "metadata": {},
  "output_type": "execute_result"
 },
 {

  },
  "metadata": {
   "needs_background": "light"
  },
  "output_type": "display_data"
 }
],
"source": [
 "sns.distplot(df['CONTENT_LENGTH'])"
]
```

```
},
{
 "cell_type": "code",
 "execution_count": 19,
 "id": "b5b8b8c1",
 "metadata": {},
 "outputs": [
  {
   "name": "stderr",
   "output_type": "stream",
   "text": [
    "C:\\Users\\ADMIN\\anaconda3\\lib\\site-
packages\\seaborn\\distributions.py:2557: FutureWarning: `distplot` is a deprecated
function and will be removed in a future version. Please adapt your code to use
either `displot` (a figure-level function with similar flexibility) or `histplot` (an
axes-level function for histograms).\n",
    "  warnings.warn(msg, FutureWarning)\n"
   ]
  },
  {
   "data": {
    "text/plain": [
     "<AxesSubplot:xlabel='APP_BYTES', ylabel='Density'>"
    ]
   },
   "execution_count": 19,
   "metadata": {},
   "output_type": "execute_result"
  },
  {
  {
   },
```

```
  "metadata": {
   "needs_background": "light"
  },
  "output_type": "display_data"
 }
],
"source": [
 "sns.distplot(df['APP_BYTES'])"
]
},
{
"cell_type": "code",
"execution_count": 20,
"id": "51e564e1",
"metadata": {},
"outputs": [
 {
  "name": "stderr",
  "output_type": "stream",
  "text": [
   "C:\\Users\\ADMIN\\anaconda3\\lib\\site-
packages\\seaborn\\distributions.py:2557: FutureWarning: `distplot` is a deprecated
function and will be removed in a future version. Please adapt your code to use
either `displot` (a figure-level function with similar flexibility) or `histplot` (an
axes-level function for histograms).\n",
   "  warnings.warn(msg, FutureWarning)\n"
  ]
 },
 {
  "data": {
   "text/plain": [
```

```
      "<AxesSubplot:xlabel='APP_PACKETS', ylabel='Density'>"
     ]
    },
    "execution_count": 20,
    "metadata": {},
    "output_type": "execute_result"
   },
   {


]
    },
    "metadata": {
     "needs_background": "light"
    },
    "output_type": "display_data"
   }
  ],
  "source": [
   "sns.distplot(df['APP_PACKETS'])"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": 21,
  "id": "165f16f4",
  "metadata": {},
  "outputs": [
   {
    "data": {
     "text/plain": [
```

```
    "<AxesSubplot:>"
   ]
  },
  "execution_count": 21,
  "metadata": {},
  "output_type": "execute_result"
 },
 {
  },
  "metadata": {
   "needs_background": "light"
  },
  "output_type": "display_data"
 }
],
"source": [
 "plt.figure(figsize=(10, 10))\n",
 "sns.barplot(data = new_df, orient='h')"
]
},
{
"cell_type": "code",
"execution_count": 22,
"id": "6b4138f5",
"metadata": {},
"outputs": [
 {
  "data": {
   "text/plain": [
    "<seaborn.axisgrid.PairGrid at 0x172b33cc3d0>"
   ]
```

```
 },
 "execution_count": 22,
 "metadata": {},
 "output_type": "execute_result"
},
{
```

7.2 SCREENSHOTS

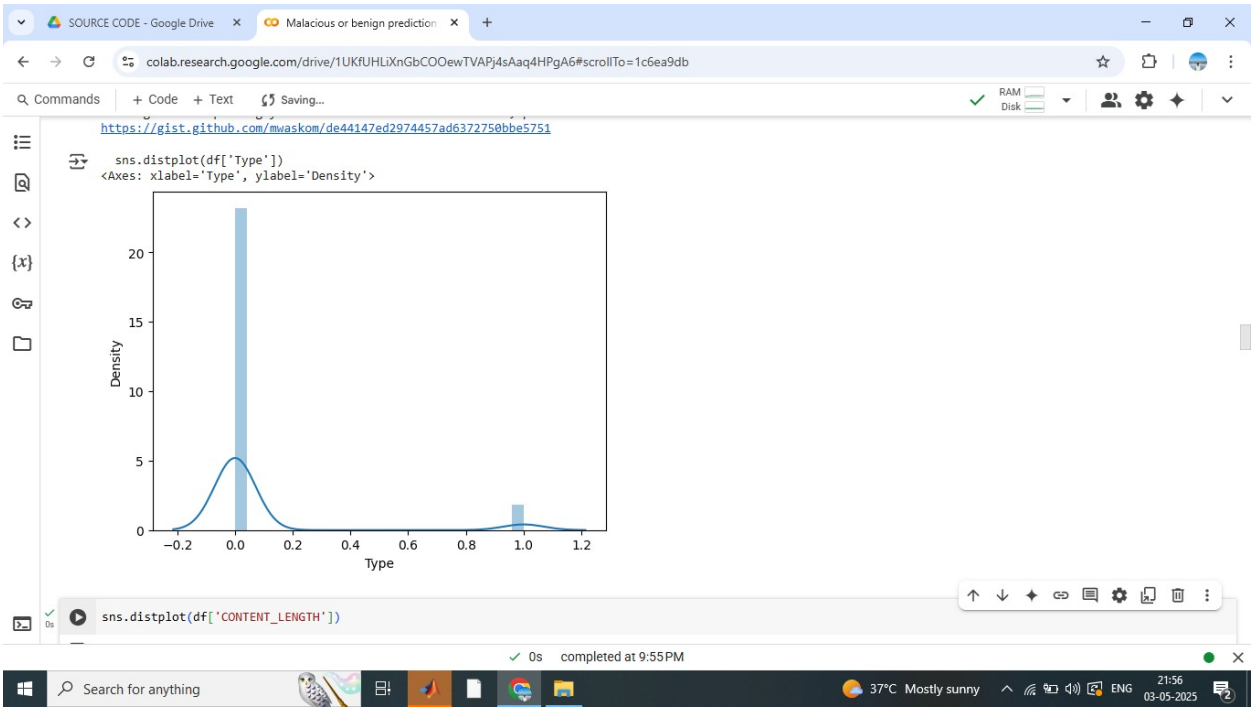


Figure7.2.1

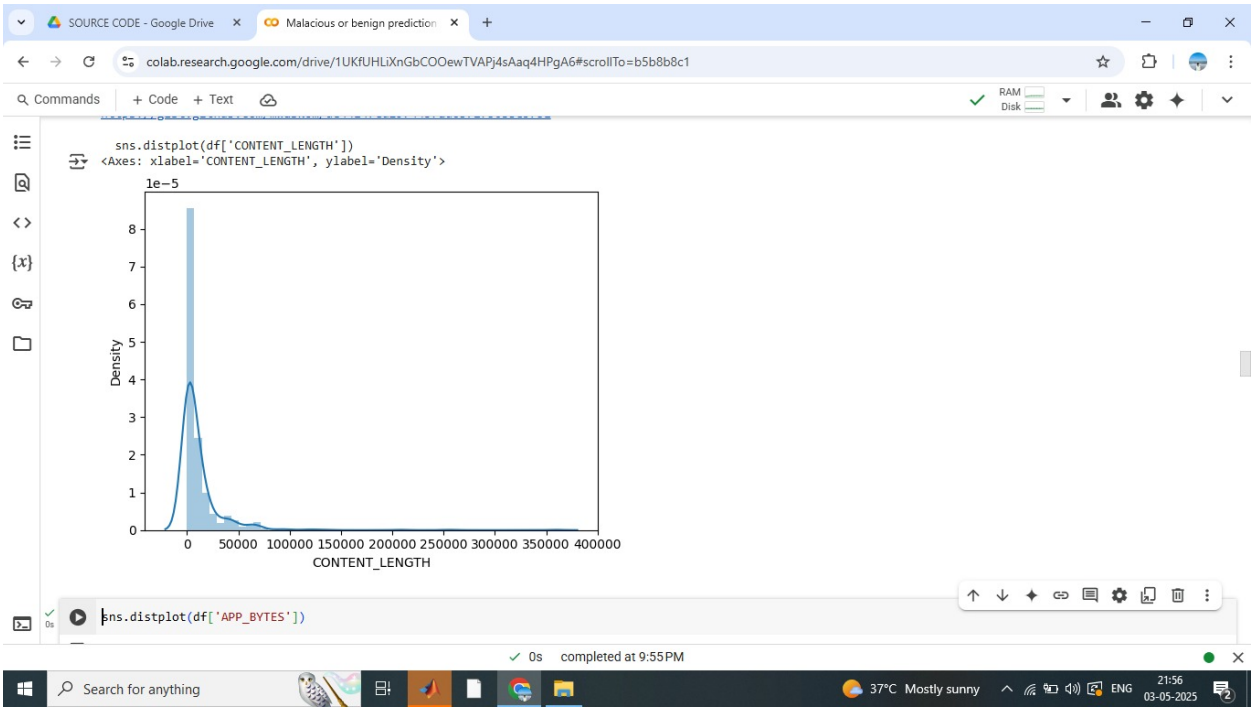


Figure 7.2.2

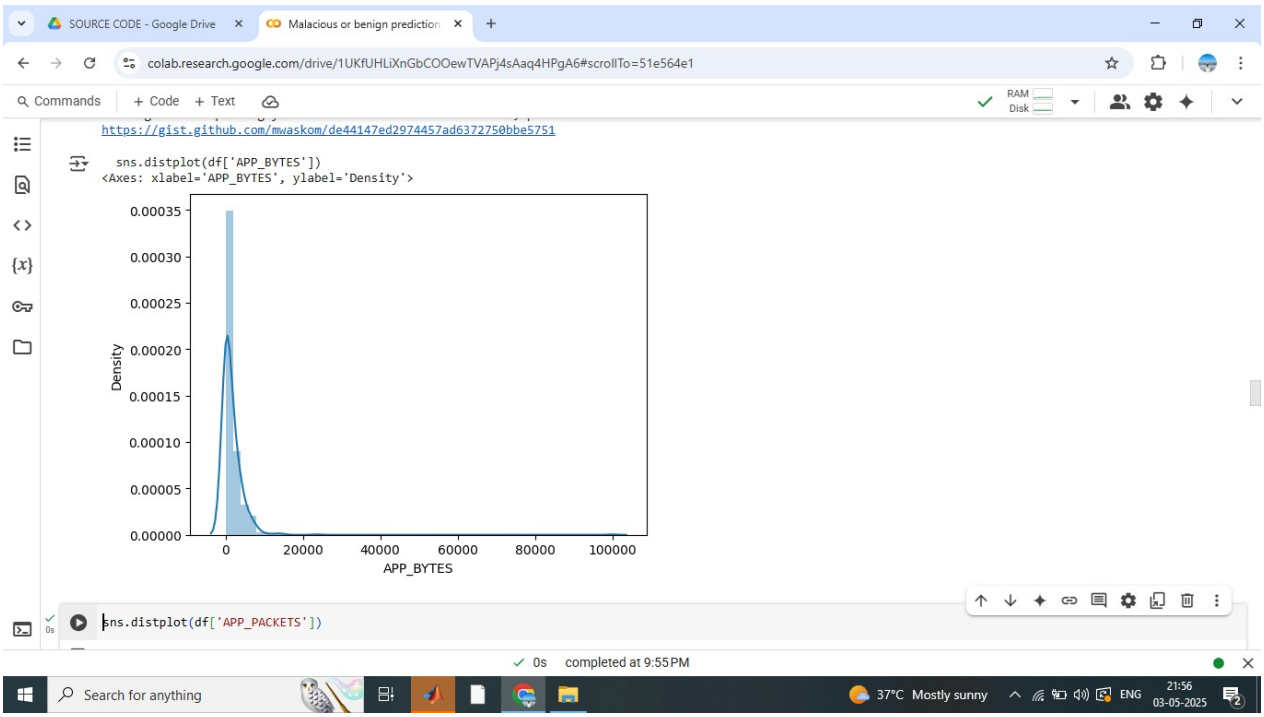


Figure 7.2.3

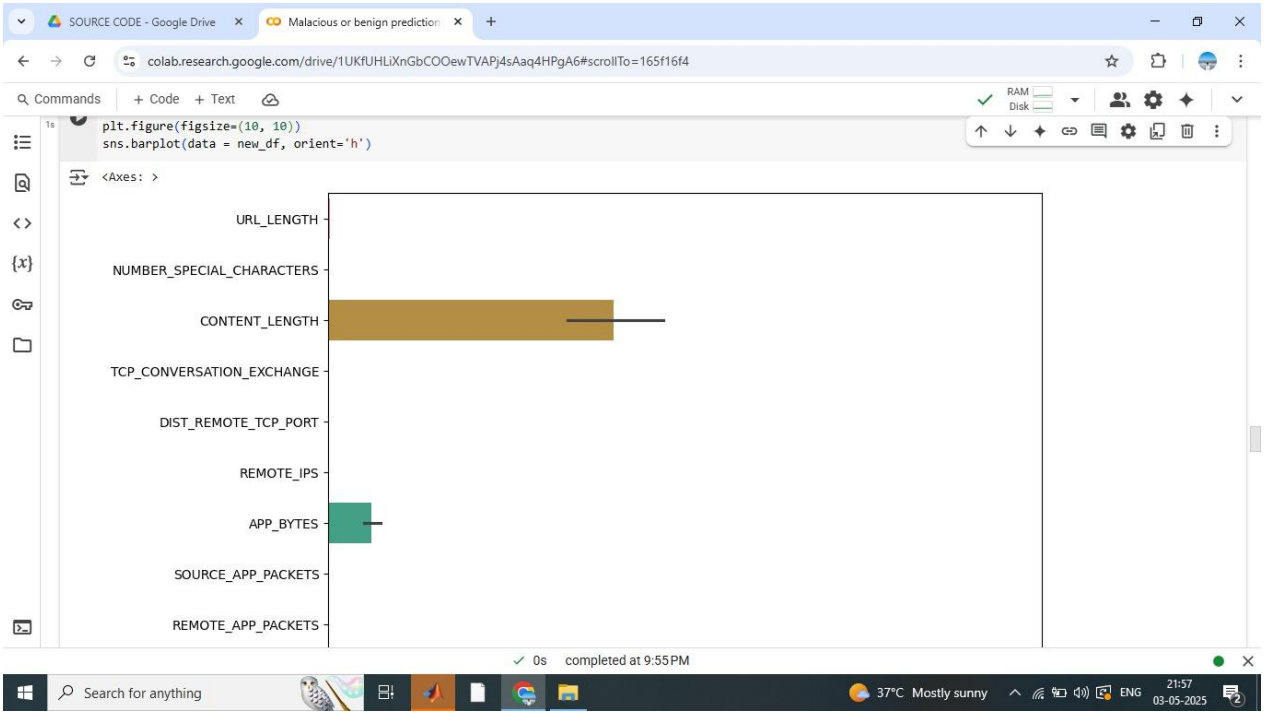


Figure 7.2.4

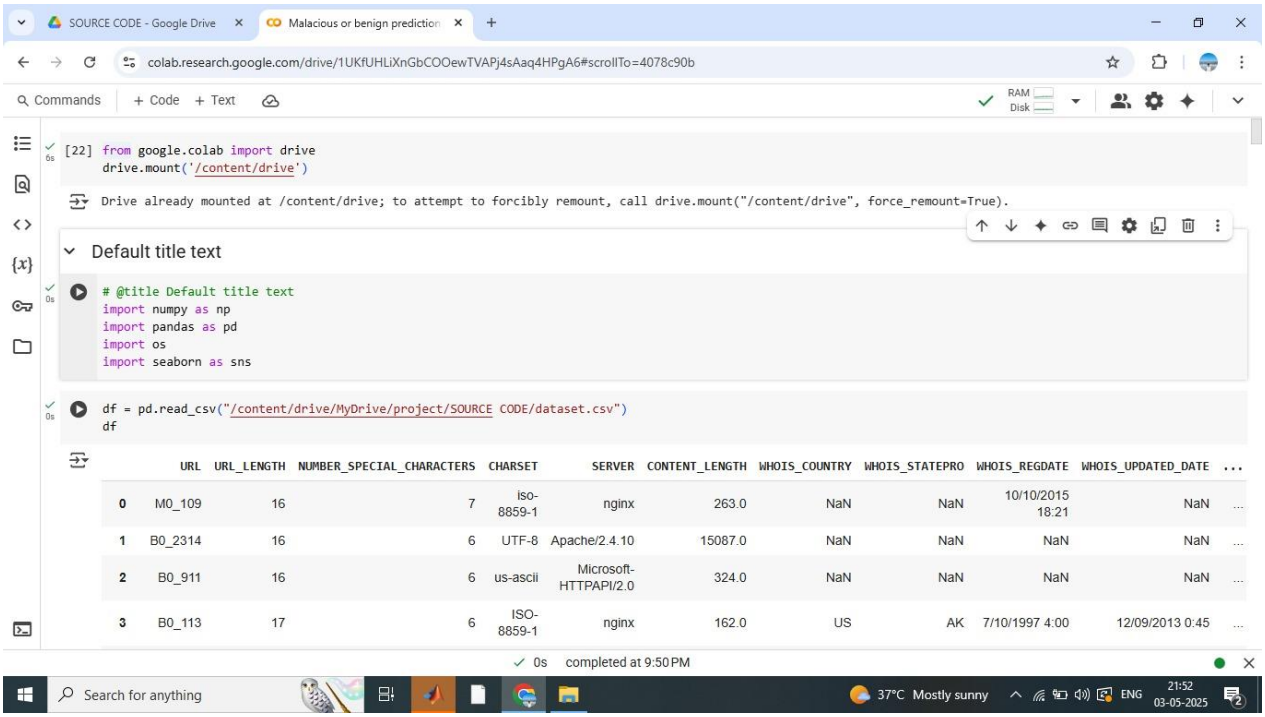


FIGURE 7.2.5

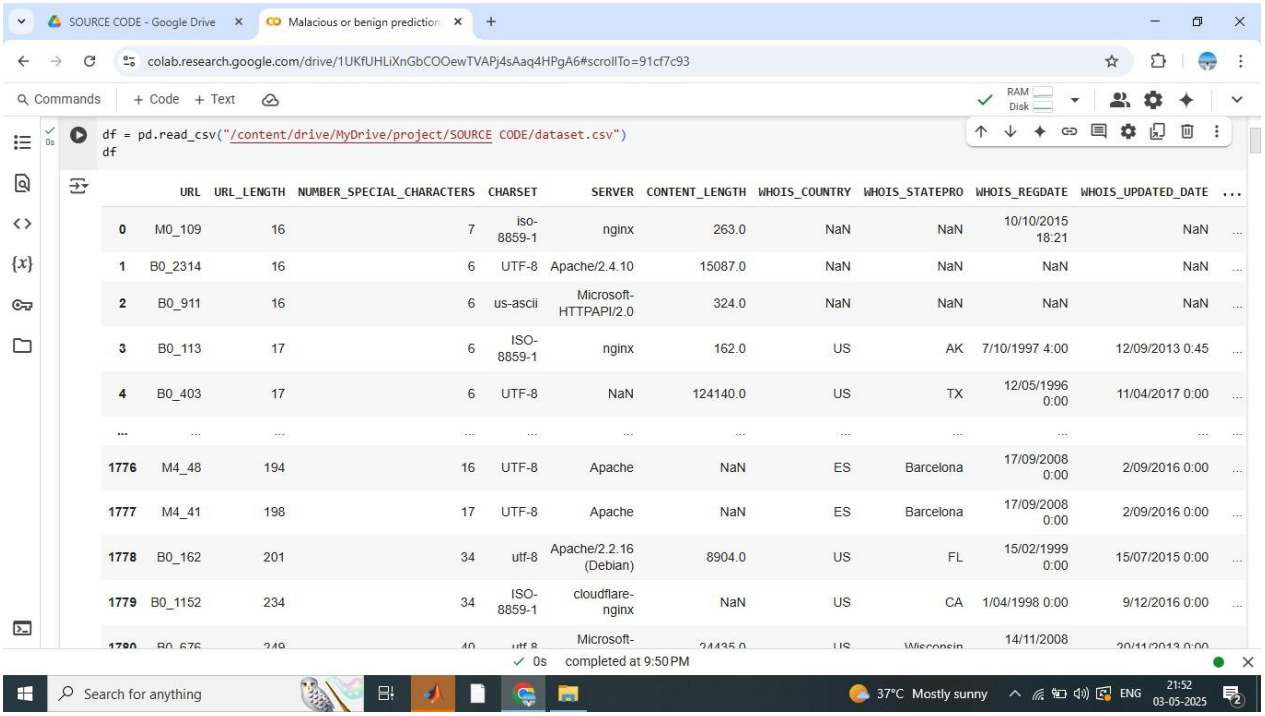


FIGURE 7.2.6

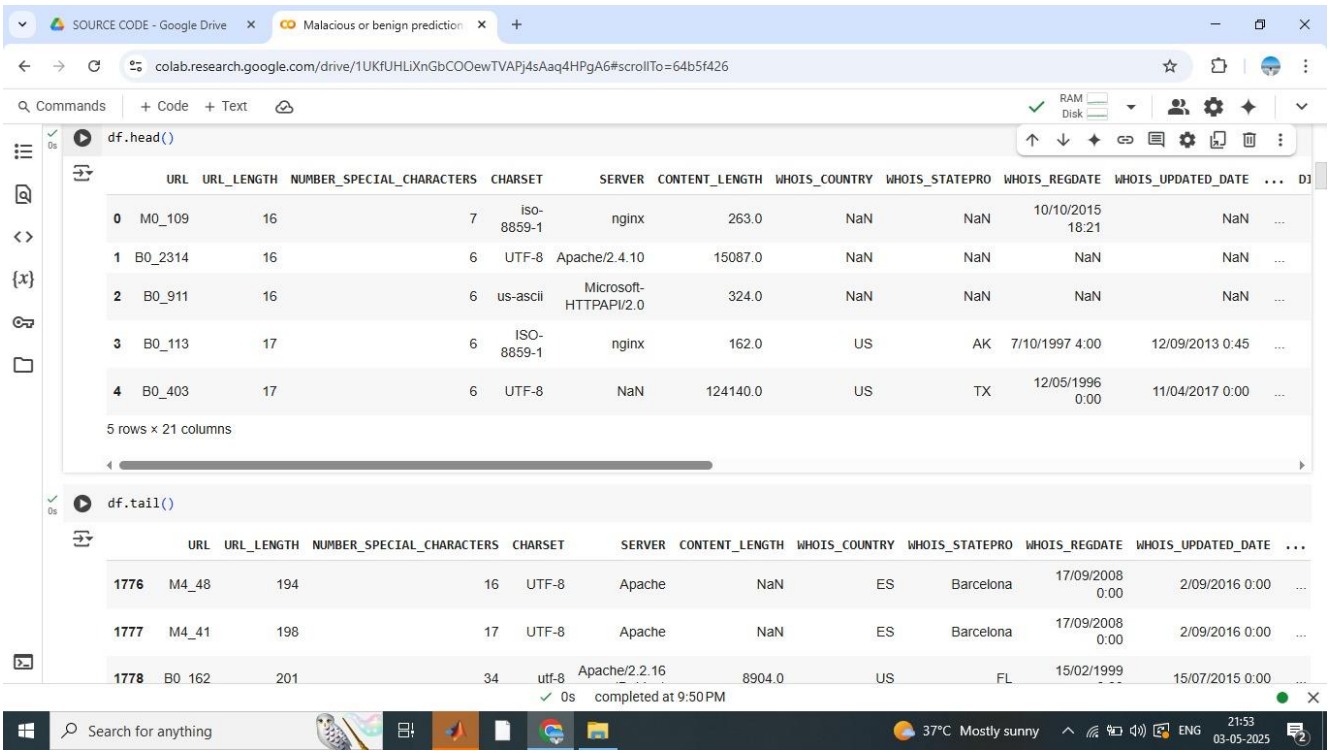


FIGURE 7.2.7

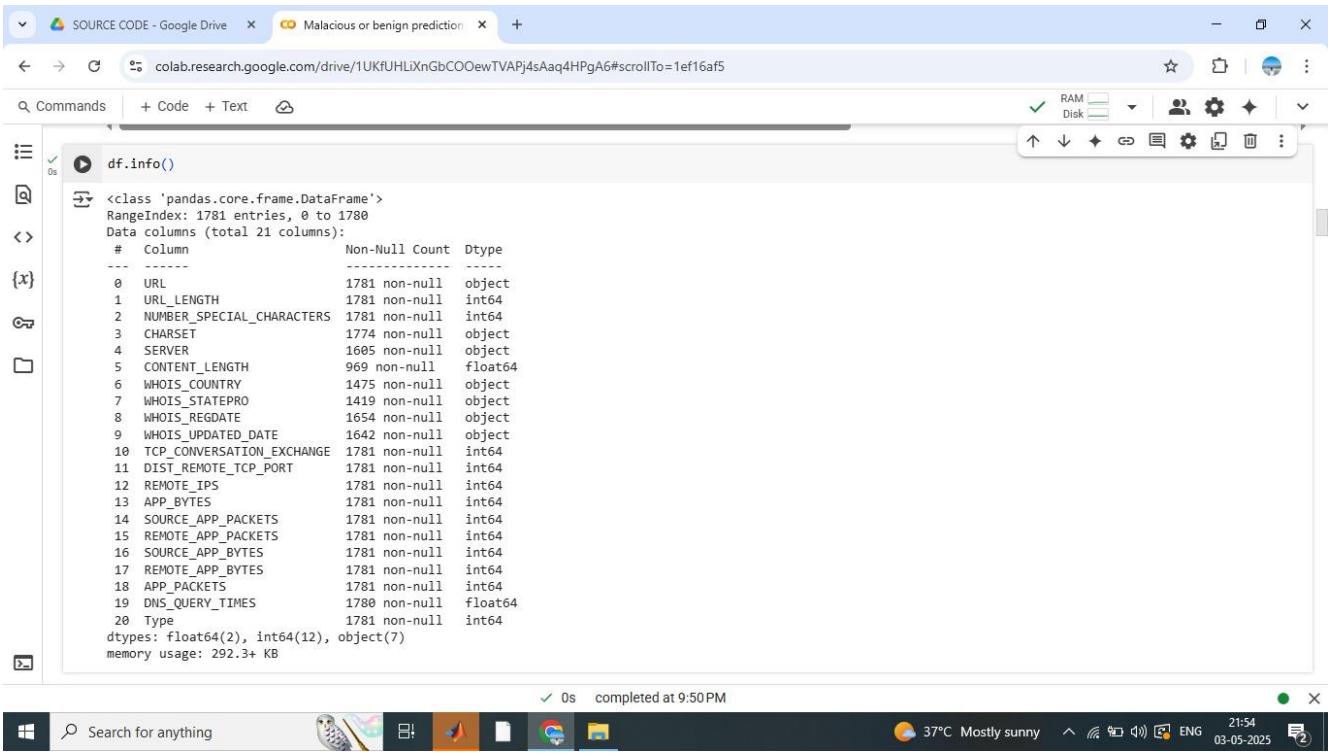


FIGURE 7.2.8

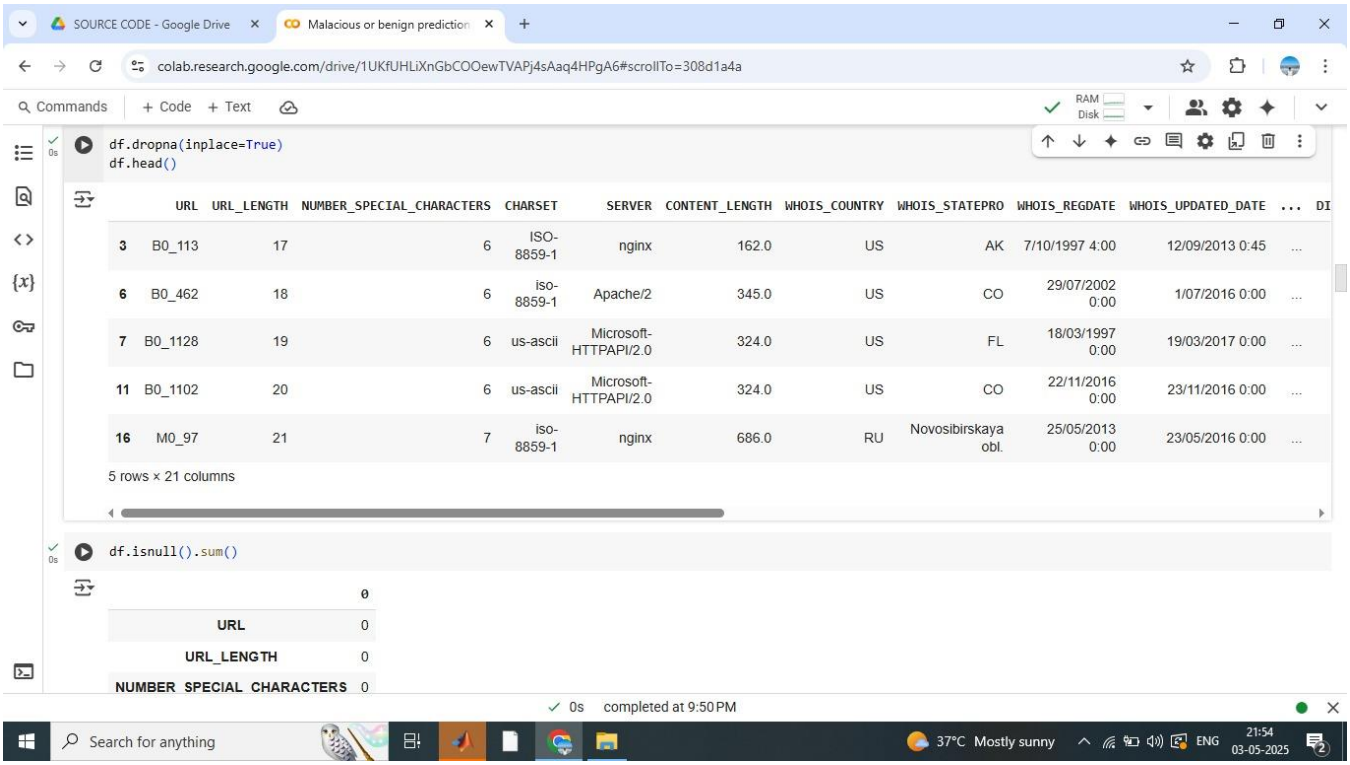


FIGURE 7.2.9

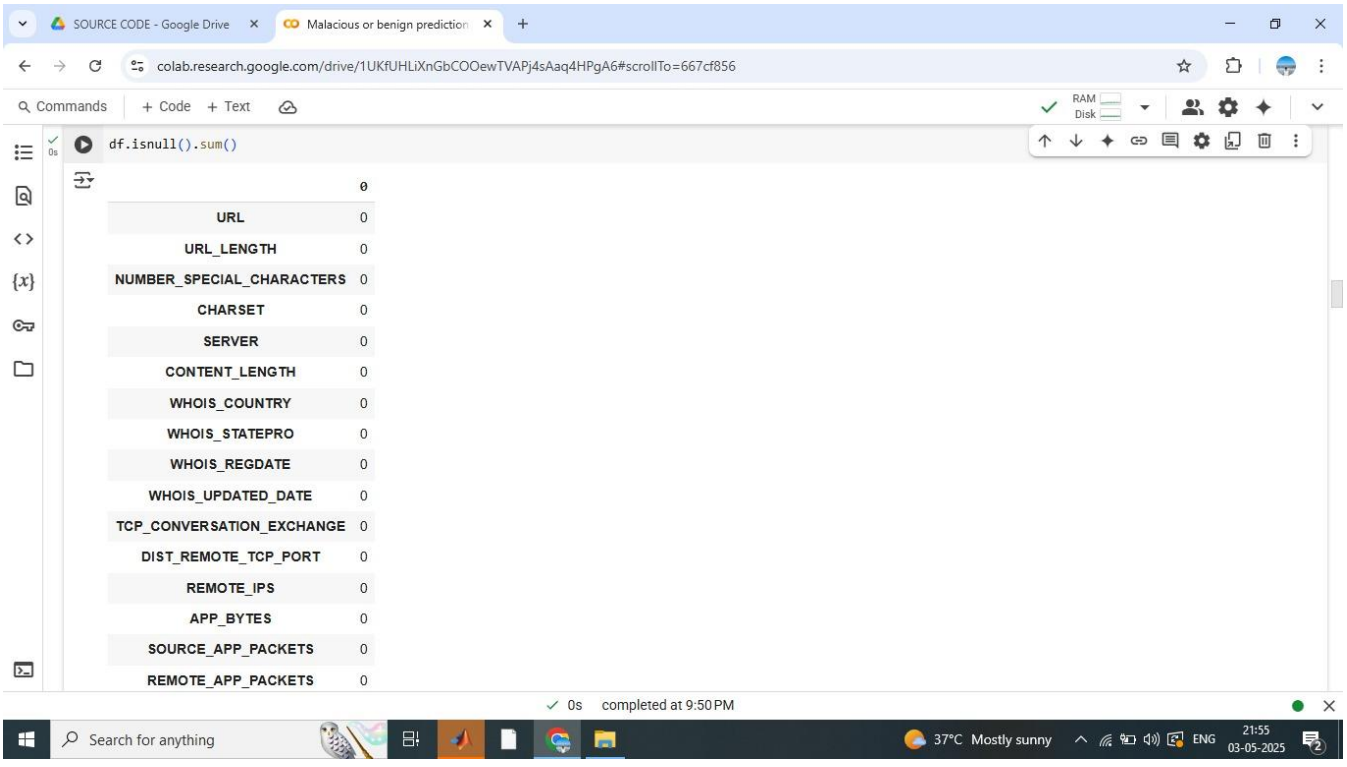


FIGURE 7.2.10

8.TESTING

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub – assemblies, assemblies and/or a finished product It Is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of tests. Each test type addresses a specific testing requirement.

8.1 Objectives

- Ensure the accuracy of the model in detecting malicious URLs.
- Validate the performance of different machine learning algorithms.
- Identify potential false positives and false negatives.
- Verify the robustness of the system against adversarial inputs.
- Ensure reliability, scalability, and efficiency of the system**.**

8.1 Test Process

- Test Data Preparation:

- Collect datasets of both malicious and benign URLs.

- Split data into training, validation, and test sets.

Model Training and Testing:

- Train machine learning models using training data.

- Validate model using validation data.

- Evaluate performance on unseen test data

Functional Testing

- Verify the system correctly classifies URLs as malicious or benign.

Performance Testing

- Measure accuracy, precision, recall, and F1-score.

- Test model latency and response time.

Security Testing

- Check if the model can handle adversarial URL inputs.
- Ensure resistance to injection attacks.

Usability Testing

- Validate the user interface (if applicable).

Regression Testing:

- Ensure system updates do not affect previous functionalities.

8.3 Test Cases

Test Case ID	Test Case Description	Input	Expected Output	Remarks
TC-01	Test URL classification for known malicious URLs	http://phishing-site.com	Classified as "Malicious"	Verify detection accuracy
TC-02	Test URL classification for known benign URLs	https://www.google.com	Classified as "Benig"	Ensure no false positives
TC-03	Test classification for shortened URLs	https://bit.ly/xyz	classification	Handle URL
TC-04	Test classification for new/unseen URLs	http://new-suspicious.com	Correct classification	Evaluate capability
TC-05	Test model against obfuscated/masked URLs	hxxp://malicious-site[.]com	Classified as "Malicious"	Ensure robustness
TC-06	Test performance with large dataset	1 million URLs	Response time within limits	Check scalability
TC-07	Verify real-time classification accuracy	Streaming URLs	Near real-time detection	Check system responsiveness
TC-08	Test against adversarial URLs	Slightly modified malicious URLs	Still classified as "Malicious"	Check resilience against evasion techniques

8.4 TYPES OF TESTS

UNIT TESTING

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated.

INTEGRATION TESTING

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent.

FUNCTIONAL TEST

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

Functional testing is centered on the following items

- Valid Input : identified classes of valid input must be accepted.
- Invalid Input: identified classes of invalid input must be rejected.
- Functions : identified functions must be exercised.
- Output : identified classes of application outputs must be exercised.
- Procedures : interfacing systems or procedures must be invoked.

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify Business process flows;

data fields, predefined processes, and successive processes must be considered for testing. Before functional testing is complete, additional tests are identified and the effective value of current tests is determined.

SYSTEM TEST

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration-oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

WHITE BOX TESTING

White Box Testing is a testing in which in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose. It is purpose. It is used to test areas that cannot be reached from a black box level.

BLACK BOX TESTING

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document.

UNIT TESTING

Unit testing is usually conducted as part of a combined code and unit test phase of the software lifecycle, although it is not uncommon for coding and unit testing to be conducted as two distinct phases.

Test objectives

- All field entries must work properly.
- Pages must be activated from the identified link.
- The entry screen, messages and responses must not be delayed.

8.5 Verification & Validation

Verification

Data Verification

- Ensure data is properly cleaned and preprocessed (e.g., handling missing values, duplicates).
- Check confusion matrix, precision, recall, and F1-score on test datasets.

Code Verification

- Perform unit testing for individual functions.
- Use assertions to validate intermediate outputs.

Validation

System Validation

- Check if the model meets the accuracy, precision, and recall requirements.
- Evaluate the system on real-world datasets to ensure it performs well.

User Acceptance Testing (UAT)

- Test with cybersecurity experts or domain users to ensure usability and effectiveness.
- Gather feedback and refine the model.

Comparative Validation

- Compare the model with existing detection techniques (e.g., signature-based, heuristic methods).
- Ensure improvements over traditional methods.

9.REPORT

Module1- Data collection

Module 2- Features Extraction

Module 3- Data preprocessing

Module 4- Model Training

Module 5- Evaluation

Module 6- Prediction and User Interface

Module 1: Data Collection

Description: The first step in developing a machine learning system is gathering relevant data. In this project, we collect datasets that contain a mix of malicious and benign (safe) URLs. These datasets serve as the foundation for training the ML models to distinguish harmful URLs from safe ones.

Sources

- Alexa Top Sites: Commonly used as a source for benign URLs.
- PhishTank: A community-driven platform where users submit and verify phishing URLs.
- Kaggle Datasets: Contains many labeled datasets that include phishing, malware, and suspicious URLs.

Output

A structured dataset (usually in .csv format) containing:

- A URL column – the actual link.
- A Label column – indicates whether the URL is malicious (1) or benign (0).
- This dataset becomes the input for feature extraction and model training.

Module 2: Feature Extraction

Description

URLs by themselves are strings, so machine learning models can't understand them directly. This module transforms each URL into a set of numerical features that capture patterns often found in malicious URLs. This process is known as feature engineering.

Types of Features Extracted:

URL Length: Malicious URLs tend to be longer or excessively short to obfuscate intent.

Presence of Special Characters: Symbols like @, -, _, =, %, and // are often used in phishing URLs to mislead users or mimic trusted domains.

Number of Subdomains: More subdomains (e.g., login.secure.bank.example.com) may indicate attempts to mimic legitimate domains.

Use of IP Address Instead of Domain Name: URLs with IP addresses (like http://192.168.1.1/login) are often suspicious because legitimate websites usually use domain names.

Suspicious Keywords: Words like "login", "secure", "account", "verify", and "update" are common in phishing links designed to steal user credentials.

Output: Each URL is converted into a numerical feature vector – a list of values that represent the extracted properties. These vectors are then used to train machine learning models.

Module 3: Data Preprocessing

Description:

Before training a model, the dataset must be cleaned and transformed into a suitable format. This module handles that process.

Steps

- Remove Missing or Duplicate Entries: Ensures data integrity by eliminating corrupt or repeated data.
- Normalize or Scale Features: Brings all numerical values into a consistent range (e.g., 0 to 1), which improves model performance and training stability.
- Encode Labels: The target variable (malicious or benign) is encoded into numeric form. Typically, malicious = 1 and benign = 0.

Tools Used

- Pandas: For data manipulation and analysis.
- NumPy: For numerical computations.
- Scikit-learn: For preprocessing functions like label encoding and normalization.

Module 4: Model Training

Description:

This is the core module where machine learning algorithms are applied to learn patterns from the feature vectors created earlier. The goal is to build a model that can distinguish between malicious and benign URLs.

Algorithms Used

- Decision Tree: A tree-based classifier that splits data based on feature values to make decisions.
- Random Forest: An ensemble of decision trees that improves accuracy and reduces overfitting.
- Support Vector Machine (SVM): A powerful classifier that finds the best boundary between classes.
- Logistic Regression: A statistical model often used for binary classification problems like this one.

Output

- A trained model that can take a new URL's features as input and output a prediction (malicious or not).
- This model is saved for later use in the prediction module.

Module 5: Evaluation

Description

- Once the model is trained, it must be tested to ensure it performs well on new, unseen

data. This module evaluates the accuracy and reliability of the model.

- Evaluation Metrics:
- Accuracy: The proportion of correctly predicted URLs.
- Precision: How many predicted malicious URLs were actually malicious. High precision means fewer false positives.
- Recall: How many actual malicious URLs were correctly identified. High recall means fewer false negatives.
- F1-Score: A harmonic mean of precision and recall; used when there's a class imbalance.
- Confusion Matrix: A table that summarizes correct and incorrect predictions across both classes (malicious and benign).
- Goal: Compare different models based on these metrics and choose the one with the best overall performance.

Module 6: Prediction and User Interface

Description

This is the final, user-facing module. It allows users to input a URL and get an instant prediction about its safety using the trained machine learning model.

Frontend

- Can be a simple Graphical User Interface (GUI) using Tkinter or a command-line interface.
- User enters a URL and clicks a button to get the result.

Backend

- The entered URL is passed to the backend.
- Feature extraction is done on the fly.
- The trained model processes the feature vector and predicts the result.

Output:

- The application returns and displays a clear message:
- "This URL is safe." or
- "Warning: Malicious URL detected!"
- This makes the tool accessible for non-technical users and provides real-time threat detection.

10. CONCLUSION

The project "Detection of Malicious URLs Using Machine Learning Techniques" addresses a critical cybersecurity challenge by leveraging advanced machine learning algorithms to classify URLs as either malicious or benign. Traditional methods like blacklisting and rule-based systems have proven ineffective against evolving cyber threats, particularly newly generated or obfuscated URLs. This project introduces a robust, intelligent system capable of analyzing various URL features—such as lexical structure, domain registration data, and network traffic—to detect malicious URLs with high accuracy and adaptability. By focusing on URL characteristics rather than webpage content, the system ensures lightweight and real-time performance, making it suitable for integration into browsers, email filters, and network security tools.

Looking ahead, the system holds immense potential for future enhancements. Integrating deep learning models like CNNs and RNNs could further improve detection accuracy, while explainable AI (XAI) techniques could enhance transparency and trustworthiness. Expanding the dataset and incorporating adversarial attack resistance would strengthen the system's robustness against evolving threats. Additionally, deploying the system as a browser extension or cloud-based API would broaden its accessibility and usability.

In summary, this project successfully bridges the gap between traditional URL filtering techniques and modern machine learning approaches, offering a scalable, efficient, and accurate solution for malicious URL detection. Its implementation marks a significant step forward in cybersecurity, empowering users and organizations to navigate the digital landscape with greater confidence and security. The insights gained from this project pave the way for further research and development, ensuring continuous improvement and adaptation to emerging cyber threats. By fostering collaboration between data scientists, cybersecurity experts, and developers, the project exemplifies the transformative potential of machine learning in addressing real-world challenges. Ultimately, this system contributes to a safer internet environment, mitigating risks and enhancing trust in online interactions.

11. FUTURE ENHANCEMENT

As cyber threats evolve, enhancing the malicious URL detection system with advanced technologies and methodologies is crucial. Below are some potential future enhancements that can improve the accuracy, efficiency, and applicability of the system.

Improving Feature Extraction

Enhance feature engineering techniques to extract more informative features from URLs,such as analyzing content-based attributes.

Deep Learning Integration

Implement deep learning models like CNNs and RNNs to improve detection accuracy.

Browser Extension or API Development

- Deploy the model as a browser extension or API for seamless user integration.

Dataset Expansion & Augmentation:

- Continuously update and expand the dataset to improve model generalization.

Cloud-based Deployment

- Implement a cloud-based solution for scalability and accessibility.

Integration with Cybersecurity Systems

- Connect with existing security solutions like SIEM for comprehensive protection.

REFERENCES

- Tao, Wang, Yu Shunzheng, and Xie Bailin. “A novel framework for learning to detect malicious web pages.” In 2010 International Forum on Information Technology and Applications, vol. 2, pp. 353-357. Ieee, 2010.
- Eshete, Birhanu, Adolfo Villafiorita, and Komminist Weldemariam. “Malicious website detection: Effectiveness and efficiency issues.” In 2011 First SysSec Workshop, pp. 123-126. IEEE, 2011.
- Aldwairi, Monther, and Rami Alsalman. “Malurls: A lightweight malicious website classification based on url features.” Journal of Emerging Technologies in Web Intelligence 4, no. 2 (2012): 128-133.
- Yoo, Suyeon, Sehun Kim, Anil Choudhary, O. P. Roy, and T. Tuithung. “Two-phase malicious web page detection scheme using misuse and anomaly detection.” International Journal of Reliable Information and Assurance 2, no. 1 (2014): 1-9.
- Hwang, Young Sup, Jin Baek Kwon, Jae Chan Moon, and Seong Je Cho. “Classifying malicious web pages by using an adaptive support vector machine.” Journal of Information Processing Systems 9, no. 3 (2013): 395-404.
- Yue, Tao, Jianhua Sun, and Hao Chen. “Fine-grained mining and classification of malicious Web pages.” In 2013 Fourth International Conference on Digital Manufacturing & Automation, pp. 616-619. IEEE, 2013.
- Krishnaveni, S., and K. Sathiyakumari. “SpiderNet: An interaction tool for predicting malicious web pages.” In International Conference on Information Communication and Embedded Systems (ICICES2014), pp. 1-6. IEEE, 2014.
- Urcuqui, Christian, Andres Navarro, Jose Osorio, and Melisa García. “Machine Learning Classifiers to Detect Malicious Websites.” In SSN, pp. 14- 17. 2017.).