# MALICIOUS URL DETECTION USING MACHINE LEARNING IN PYTHON

A Project report submitted in partial fulfilment of the requirements for the award of the degree of

**B.Sc. DATA SCIENCE & ANALYTICS**

Submitted by

**AKHILA C U**

**21BDA012**

Under the guidence of

**Ms. K. MANIMEKALAI, MCA., M.Phil.**

Assistant Professor, Department of  Data Science & Analytics

**UG  DEPARTMENT OF DATA SCIENCE & ANALYTICS**

**SREE SARASWATHI THYAGARAJA COLLEGE**

An Autonomous, ISO Certified and NACC Re-Accredited with 'A' Grade, ISO 12001: 21008 Certified Institution,

Affiliated To Bharathiar University, Coimbatore

Approved by AICTE for MBA/MCA & by UGC for 2(f) & 12(B)status

**APRIL 2024**

# CERTIFICATE

This is to certify that the project entitled **"MALICIOUS URL DETECTION USING MACHINE LEARNING IN PYTHON"** Submitted to **Sree Saraswathi Thyagaraja Collage (Autonomous), Pollachi**, affiliated to Bharathiar University, Coimbatore in partial fulfilment of the requirements for the award of the degree of **B.Sc. DATA SCIENCE & ANALYTICS** is a record of original work done by **AKHILA C U** under my supervision and guidance and the report has not previously formatted the basis for the award of any Degree/ Diploma / Associated ship/ Fellowship or other similar title to any candidate of any University.

| | |
|---|---|
| **Date:** | **Signature of the Guide** |
| **Place:** | **K. Manimekalai** |

**Counter Signed by**

| | |
|---|---|
| **HOD** | **DIRECTOR** |
| **Dr. A. SAMUEL CHELLATHURAI** | **Dr. A. ABDUL RASHEED** |

**Viva-voce Examination held on ------------------**

| | |
|---|---|
| **INTERNAL EXAMINER** | **EXTERNAL EXAMINER** |

# DECLARATION

I **AKHILA C U** hereby declare that the project report entitled **"MALICIOUS URL DETECTION USING MACHINE LEARNING IN PYTHON"** submitted to **Sree Saraswathi Thyagaraja College (Autonomous), Pollachi** , affiliated to **Bharathiar University**, Coimbatore in partial fulfilment of the requirements for the award of the degree of **B.Sc. DATA SCIENCE & ANALYTICS** is a record of original work done by me under the guidance of **K. MANIMEKALAI, MCA., M.Phil. Assistant Professor**, Department of **DATA SCIENCE AND ANALYTICS** and it has not previously formed the basis for the award of any Degree to any candidate of any University.

| | |
|---|---|
| **Place :** | **Signature of Candidate** |
| **Date :** | **AKHILA C U** |

# ACKNOWLEDGEMENT

The success of the project depends on the efforts invested. It’s my duty to acknowledge and thank the individuals who has contributed in successful completion of the project.

I wish to express our deep sense of gratitude to the Management of Sree Saraswathi Thyagaraja College, Pollachi, for permitting to carry out the project work.

I would like to convey my sincere thanks to my beloved Principal **Dr. A. SOMU,** MBA.**,** M.Com., M.Phil., Ph.D. for providing necessary facilities that enabled to the success completion of my project.

I would like to express my sincere thanks to the Director of Computer science **Dr. A. ABDUL RASHEED,** MCA., M.E., Ph.D. for his directions. I thank him for the interest and continuous support.

I would like to express my sincere thanks to the Head of Department of DSA **Dr. A . SAMUEL CHELLATHURAI** for the interest and continuous support.

I would like to express my sincere thanks to my project guide and the Assistant Professor of the Department of DSA, **Ms. K. MANIMEKALAI,** MCA., M.Phil. for her valuable guidance and direction . I thank her for the interest and continuous support.

I owe my special gratitude to my parents and friends who helped me to make this project a memorable success.

# ABSTRACT

The proliferation of cyber threats has necessitated the development of efficient techniques for detecting malicious URLs. This project focuses on leveraging machine learning algorithms implemented in Python to enhance the detection of such URLs. The study explores various features extracted from URLs, including lexical, content-based, and structural attributes, to train and evaluate different machine learning models. By utilizing a diverse dataset encompassing both benign and malicious URLs, the project aims to build robust classifiers capable of accurately distinguishing between the two categories. The ultimate objective is to develop a reliable and scalable solution to mitigate the risks posed by malicious URLs in online environments.

The project begins by conducting an extensive literature review to understand the existing methodologies and challenges in malicious URL detection. Subsequently, various machine learning algorithms, such as random forest, support vector machines, and deep learning models, are implemented and fine-tuned using Python libraries like scikit-learn and TensorFlow. Feature engineering techniques are employed to extract relevant information from URLs, including domain reputation, URL length, presence of suspicious keywords, and syntactic structure. The effectiveness of the classifiers is evaluated through rigorous testing on benchmark datasets and real-world samples, with performance metrics such as accuracy, precision, recall, and F1-score analyzed to assess their efficacy.

In conclusion, this project contributes to the field of cybersecurity by providing a comprehensive framework for detecting malicious URLs using machine learning techniques implemented in Python. The developed solution offers a practical approach to combatting evolving cyber threats, thereby enhancing the security posture of online systems and protecting users from potential risks associated with malicious URLs. Future research directions include exploring advanced machine learning algorithms, integrating anomaly detection techniques, and adapting the framework to dynamic environments for continuous monitoring and threat mitigation.

# TABLE OF CONTENT

# 1 . INTRODUCTION

## 1.1 OVERVIEW OF THE PROJECT

### 1.1.1 What Is URL?

The Uniform Resource Locator (URL) is the well-defined structured format unique address for accessing websites over World Wide Web (WWW).Generally, there are three basic components that make up a legitimate URL

i.) **Protocol**: It is basically an identifier that determines what protocol to use e.g., HTTP, HTTPS, etc.

ii.) **Hostname**: Also known as the resource name. It contains the IP address or the domain name where the actual resource is located.1

iii) **Path:** It specifies the actual path where the resource is located

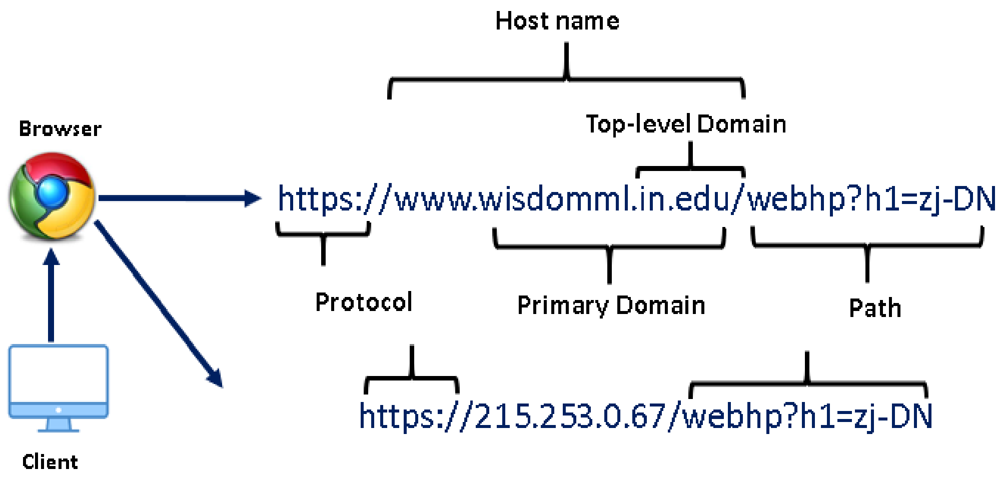

Host name
Browser
Top-level Domain
https://www.wisdomml.in.edu/webhp?h1=zj-DN
Protocol
Primary Domain
Path
https://215.253.0.67/webhp?h1=zj-DN
Client

**Fig. 1. Components of a URL**

As per the figure 1, wisdomml.in.edu is the domain name. The top-level domain is another component of the domain name that tells the nature of the website i.e, commercial (.com), educational (.edu), organization (.edu), etc.

### 1.1.2 What is Malicious URL ?

Modified or compromised URLs employed for cyber attacks are known as malicious URLs.A malicious URL or website generally contains different types of trojans, malware, unsolicited content in the form of phishing, drive-by-download, spams.

The main objective of the malicious website is to fraud or steal the personal or financial details of unsuspecting users. Due to the ongoing COVID-19 pandemic the incidents of cybercrime increased manifold. According to Symantec Internet Security Threat Report (ISTR) 2019, malicious URLs are a highly used technique in cyber crimes.

In this article, we address the detection of malicious URLs as a multi-class classification problem by classifying the raw URLs into different class types such as benign or safe URLs, phishing URLs, malware URLs, or defacement URLs. This form of trade involves buying and selling stocks in a single day. A trader involved in suchtrades needs to close the position before the day's market closure. Day trading requires proficiency in market matters and a good understanding of market volatility. Therefore, day trading is mostly practiced by experienced investors. Day traders buy and sell stocks or other assets during the trading day in order to profit from the rapid fluctuations in prices. Day trading employs a wide variety of techniques and strategies to capitalize on these perceived market inefficiencies.

### 1.1.2 Project Flow

As we know machine learning algorithms only support numeric inputs so we will create lexical numeric features from input URLs. So the input to machine learning algorithms will be the numeric lexical features rather than actual raw URLs. If you don't know about lexical features you can refer to the discussion about a lexical feature in Stack Overflow.

So, in this case study, we will be using three well-known machine learning ensemble classifiers namely Random Forest, Light GBM, and XGBoost.

Later, we will also compare their performance and plot average feature importance plot to understand which features are important in predicting malicious URLs

### 1.1.3 Purpose of Detecting Malicious URL

The primary purpose of detecting malicious URLs is to safeguard users and systems from various cyber threats, including phishing attacks, malware distribution, identity theft, and fraud. Malicious URLs serve as gateways for cybercriminals to exploit vulnerabilities in software, deceive users into divulging sensitive information, or compromise the integrity and confidentiality of data.

By identifying and blocking malicious URLs, organizations and individuals can mitigate the risks associated with cyberattacks and protect their digital assets, personal information, and financial resources.

Furthermore, detecting malicious URLs plays a crucial role in maintaining the integrity and reputation of online platforms, such as websites, social media networks, and email services. Proactively identifying and removing malicious URLs prevents their dissemination across the internet, thereby reducing the likelihood of infecting unsuspecting users and perpetuating malicious activities. This helps foster a safer online ecosystem and promotes trust and confidence among users, leading to enhanced cybersecurity awareness and resilience.

Moreover, detecting malicious URLs aids in cybersecurity research and threat intelligence by enabling the analysis of attack patterns, trends, and tactics employed by cyber adversaries. By studying the characteristics and behavior of malicious URLs, cybersecurity professionals can develop effective countermeasures, improve detection capabilities, and anticipate future cyber threats. This proactive approach strengthens cybersecurity defenses and empowers organizations to stay ahead of evolving cyber threats, thereby safeguarding critical infrastructure, sensitive data, and digital assets against malicious exploitation.

### 1.1.4 Scope of this project

**Data Collection and Preprocessing:** The project involves gathering a diverse dataset containing both benign and malicious URLs from various sources, including online repositories, security feeds, and real-world samples. The collected data undergoes preprocessing steps to clean, normalize, and extract relevant features, such as URL structure, domain reputation, and content attributes.

**Feature Engineering and Selection:** The project explores different feature extraction techniques to capture meaningful information from URLs, including lexical, content-based, and structural features. Feature engineering involves selecting and transforming the extracted features to enhance the discriminatory power of machine learning models.

**Machine Learning Model Development:** The project implements and evaluates various machine learning algorithms, such as random forest, support vector machines, and deep learning models, using Python libraries like scikit-learn and TensorFlow. Multiple classifiers are trained and fine-tuned using the pre-processed dataset to identify patterns and distinguish between benign and malicious URLs effectively.

**Performance Evaluation and Validation:** The project assesses the performance of the developed classifiers through rigorous testing on benchmark datasets and real-world samples. Performance metrics such as accuracy, precision, recall, and F1-score are analyzed to evaluate the efficacy and robustness of the detection models.

**Integration and Deployment:** The project aims to integrate the developed malicious URL detection models into existing cybersecurity systems or web applications, enabling real-time detection and mitigation of malicious URLs. The deployment phase involves implementing appropriate APIs, interfaces, or plugins to seamlessly integrate the detection functionality into the target environment.

**Documentation and Reporting:** The project documentation includes detailed descriptions of the methodologies, algorithms, implementation details, and experimental results. A comprehensive report is prepared summarizing the key findings, insights, challenges, and recommendations .

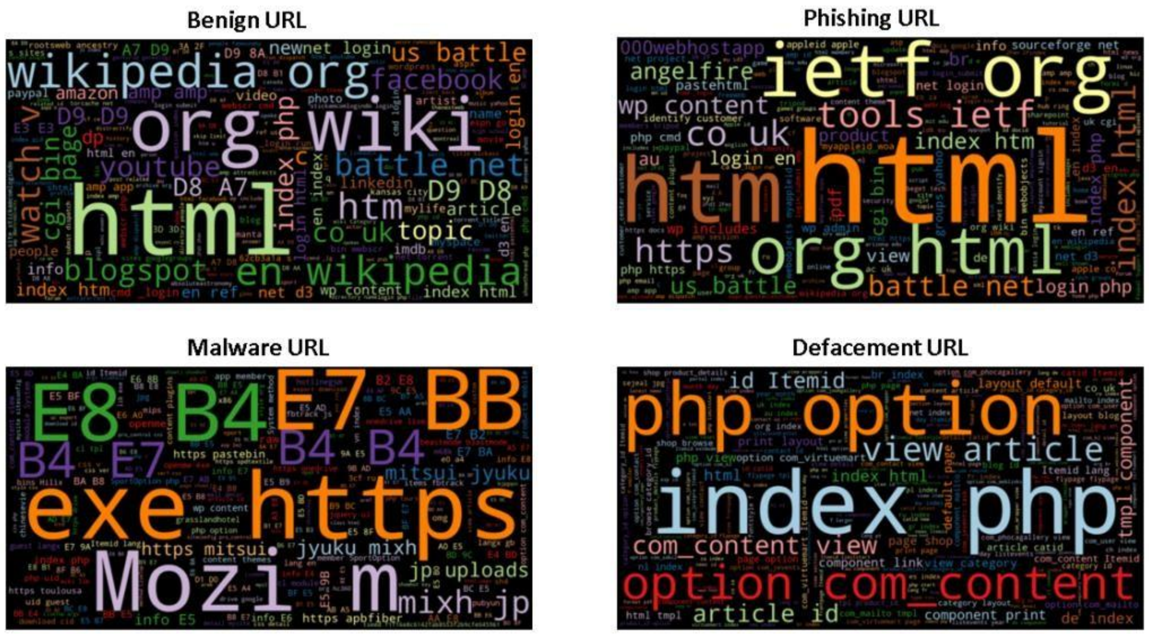
### 1.1.6 Word cloud of URLs

The word cloud helps in understanding the pattern of words/tokens in particular target labels.It is one of the most appealing techniques of natural language processing for understanding the pattern of word distribution.

As we can see in the below figure word cloud of benign URLs is pretty obvious having frequent tokens such as html, com, org, wiki etc. Phishing URLs have frequent tokens as tools, ietf, www, index, battle, net whereas html, org, html are higher frequency tokens as these URLs try to mimic original URLs for deceiving the users.

The word cloud of malware URLs has higher frequency tokens of exe, E7, BB, MOZI. These tokens are also obvious as malware URLs try to install trojans in the form of executable files over the users' system once the user visits those URLs.

The defacement URLs' intention is to modify the original website's code and this is the reason that tokens in its word cloud are more common development terms such as index, php, itemid, https, option, etc.



**Word clouds of Benign, Phishing, Malware, and Defacement URLs**

## 1.2 PROJECT DEFENITION

The project focuses on developing a machine learning-based system to detect and classify malicious URLs, categorizing them into various types such as benign, phishing, malware, and defacement URLs. The goal is to create a trained classification model capable of accurately identifying and categorizing URLs with high precision and recall rates. This system aims to enhance cybersecurity measures by enabling proactive threat mitigation, reducing reliance on manual intervention, and providing scalability and adaptability to address evolving cyber threats effectively.

### 1.2.1 Expected Outcomes

The primary objective of this project is to develop a robust machine learning-based system capable of accurately detecting and classifying malicious URLs into different categories, including benign or safe URLs, phishing URLs, malware URLs, and defacement URLs. By leveraging advanced machine learning algorithms and feature engineering techniques, the project aims to create models that can effectively analyze the characteristics and patterns of URLs to identify potential threats accurately. The expected outcome of the project is a trained classification model capable of automatically categorizing URLs into their respective classes with high precision and recall rates.

### 1.2.2 Benefits

**Enhanced Cybersecurity:** By accurately detecting and classifying malicious URLs, the project contributes to enhancing cybersecurity measures, thereby protecting users and systems from various cyber threats, including phishing attacks, malware distribution, and website defacement.

**Proactive Threat Mitigation:** The developed system enables proactive measures to be taken to mitigate potential threats posed by malicious URLs. By identifying and blocking malicious URLs in real-time, organizations can prevent cyber attacks before they occur, thereby reducing the risk of data breaches and financial losses.

**Improved Incident Response:** With the ability to quickly identify and classify malicious URLs, cybersecurity teams can respond more effectively to security incidents. The system provides valuable insights into the nature and scope of threats, enabling organizations to take appropriate remediation actions promptly.

**Cost-Efficient Solution:** Automating the detection and classification of malicious URLs using machine learning algorithms reduces the reliance on manual intervention and human resources. This results in cost savings for organizations while improving the efficiency and accuracy of cybersecurity operations.

**Scalability and Adaptability:** The developed system is scalable and adaptable, allowing it to handle large volumes of URL data and adapt to evolving cyber threats. As new types of malicious activities emerge, the system can be updated and retrained to effectively detect and classify them, ensuring long-term effectiveness in combating cyber threats.

## 1.3 STATEMENT OF PROBLEM

In this case study, we address the detection of malicious URLs as a multi-class classification problem. In this case study, we classify the raw URLs into different class types such as benign or safe URLs, phishing URLs, malware URLs, or defacement URLs.

## 1.4 PROJECT OBJECTIVES

The project centers on the development of a comprehensive solution for detecting and categorizing malicious URLs using machine learning techniques implemented in Python. By leveraging advanced algorithms and feature engineering methods, the system aims to accurately classify URLs into distinct categories such as benign, phishing, malware, and defacement URLs. The primary objective is to create a robust classification model capable of identifying malicious URLs with high precision and recall rates, thereby enhancing cybersecurity measures.

The project encompasses data collection, preprocessing, model development, evaluation, and deployment stages, with the ultimate goal of providing organizations with an effective tool to proactively mitigate cyber threats and safeguard their digital assets and users.

## 1.5 USER INTERFACE

The user interface (UI) for the Malicious URL Detector project is designed to provide a simple and intuitive experience for users to interact with the machine learning model. The interface allows users to input a URL and receive a prediction indicating whether the URL is likely to be malicious or not.

### 1.5.1 Functionality

1. Input Field: Users are provided with a text input field where they can enter the URL they wish to analyze.
2. Submit Button: Upon entering the URL, users can click the "Submit" button to initiate the analysis process.
3. Prediction Display: Once the analysis is complete, the UI will display the prediction result, indicating whether the URL is classified as malicious or not. Additionally, confidence scores or probability estimates may be provided to indicate the model's level of certainty.
4. Feedback Mechanism: Users are given the option to provide feedback on the prediction result. This feedback mechanism allows users to report any misclassifications or provide additional information to improve the model's accuracy.

### 1.5.2 Usage Instructions

1. Enter URL: Begin by typing or pasting the URL you want to analyze into the input field.
2. Submit Analysis: Click the "Submit" button to initiate the analysis process.
3. View Result: Once the analysis is complete, the UI will display the prediction result along with any confidence scores or probability estimates.
4. Provide Feedback : If you disagree with the prediction result or have additional insights, you can use the feedback mechanism to provide your input.

### 1.5.3 Note

1. Ensure that the URL entered is valid and properly formatted.
2. The prediction accuracy may vary depending on the quality and quantity of data used to train the machine learning model.

## 1.6 SCALABILITY AND PERFORMANCE

### 1.6.1 Scalability

**Data Volume:** Ensure that your system can handle a large volume of URLs efficiently. This involves optimizing your data pipelines and storage mechanisms to scale with increasing data sizes.

**Model Training:** Implement distributed training techniques if your machine learning models become too large to train on a single machine. Utilize frameworks like TensorFlow or PyTorch with distributed computing support.

**Parallel Processing:** Design your system to take advantage of parallel processing capabilities. Utilize libraries like Disk or Spark to distribute processing tasks across multiple nodes or cores.

**Infrastructure Scalability:** Use cloud computing platforms like AWS, Google Cloud, or Azure to dynamically scale your computational resources based on demand. This allows you to handle sudden spikes in traffic without sacrificing performance.

### 1.6.2 Performance:

**Feature Engineering:** Optimize feature extraction processes to ensure efficient representation of URL data. Use techniques like dimensionality reduction and feature selection to reduce computational overhead.

**Model Selection:** Choose machine learning models that strike a balance between accuracy and computational efficiency. Consider lightweight models like decision trees or ensemble methods for real-time inference.

**Model Serving:** Deploy your machine learning models using frameworks like TensorFlow Serving or FastAPI to maximize inference speed. Utilize techniques like model caching and batching to minimize latency.

**Algorithm Optimization:** Fine-tune hyperparameters and optimize algorithms for better performance. Utilize techniques like gradient boosting or model pruning to improve model efficiency without sacrificing accuracy.

**Monitoring and Profiling:** Continuously monitor system performance and identify bottlenecks using tools like Prometheus or Grafana. Profile your code to identify areas for optimization and improvement**.**

# 2. SYSTEM SPECIFICATION

## 2.1 HARDWARE SPECIFICATION

| | |
|---|---|
| SYSTEM | : ASUS TUF Gaming F15 |
| RAM | : 16 GB |
| HARD SYSTEM | : 512 GB |
| PROCESSOR | : 11th Gen Intel(R) Core(TM) i5-11400H 2.70GHz2.69GHz |

## 2.2 SOFTWARE SPECIFICATION

| | |
|---|---|
| OPERATING SYSTEM | : Windows 11(64-bit) |
| PLATFORM USED | : Google Collab |
| LANGUAGE | : Python |

## 2.3 SOFTWARE FEATURES

### 2.3.1 Overview Of Operating System

Operating System comes under Python’s standard utility modules. It helps to interact with theOperating System directly from within the collab Notebook. It makes it possible to perform many operating system tasks automatically. This module in Python has functions for creating a directory, showing its contents, showing the current directory, and also functions to change the current directory,and many more.

### 2.3.2 Overview Of Language Used

Python is a computer programming language often used to build websites and software, automate tasks, and conduct data analysis. Python is a general purpose language, meaning it can be used to create a variety of different programs and isn’t specialized for any specific problems. This versatility, along with its beginner-friendliness, has made it one of the most- used programming languages today. A survey conducted by industry analyst firm Redmond found that it was the most popular programming language among developers in 2020.

Python is one of the most popular and frequently used languages of recent times, being used for various tasks such as data science, data analytics, web development, machine learning, and automating many tasks.

- In these processes, many tasks are operating system-dependent.
- The Jupyter notebook is widely used and well documented and offers an easy to use interfacefor creating, editing, and running notebooks.
- The notebook runs as a web application called the “Dashboard” or “control panel” that showslocal files and allows users to open notebook documents and run snippets of code.
- The outputs are neatly formatted and displayed on the browser.
- The other component of the notebook is the kernel. The kernel is a “computational engine” thatexecutes the code written in the Notebook.
- It is similar to the back-end of the application.
- The IPython kernel is used to execute Python code in the collab notebook.

## 2.4 SOFTWARE & PACKAGES SPECIFICATION

In our project we used following packages

- **Pandas**
- **Itertools**
- **Sklearn.metrics**
- **Sklearn.model_selection**
- **Numpy**
- **Mathplotlib.pyplot**
- **Xgboost**
- **LightGBM**
- **Seaborn**
- **Wordcloud**

### Pandas

Pandas is a powerful data manipulation and analysis library in Python that provides easy-to-use data structures and functions for working with structured data, such as tables and time series. It's particularly well-suited for data cleaning, transformation, and analysis tasks, allowing users to load data, perform operations like filtering, grouping, and aggregation, and generate meaningful insights or visualizations. With its Data Frame and Series data structures, Pandas simplifies complex data operations and is commonly used in data science, machine learning, and financial analysis projects.

**Itertools**

The itertools module in Python's standard library offers a versatile set of tools for creating iterators and handling combinatorial tasks efficiently. It provides functions that generate iterators for generating permutations, combinations, and combinations with replacements, among others. These iterators enable efficient looping over combinations of elements without the need to generate and store the entire set in memory, which is particularly useful when dealing with large datasets or when memory usage needs to be minimized. itertools facilitates the creation of complex iterators and allows for concise and memory-efficient implementation of various algorithms and tasks involving iteration.

**Sklearn.metrics**

The `sklearn.metrics` module in scikit-learn provides a comprehensive suite of tools for evaluating the performance of machine learning models. It offers various metrics tailored to different types of tasks, including classification, regression, and clustering. This module includes functions for computing classification metrics such as accuracy, precision, recall, F1-score, and confusion matrices, which provide insights into the predictive performance of classifiers across different classes. Additionally, it provides regression metrics like mean squared error and R-squared for assessing the goodness-of-fit of regression models. These metrics are invaluable for understanding the strengths and weaknesses of machine learning models, enabling data scientists and researchers to make informed decisions during model selection, hyperparameter tuning, and performance optimization.

**Sklearn.model_selection**

The `sklearn.model_selection` module in scikit-learn provides a comprehensive set of tools for model selection and evaluation in machine learning. Its primary purpose is to facilitate the process of splitting datasets into train and test sets, cross-validation, and hyperparameter tuning. The module includes functions like `train_test_split`, which partitions data into training and testing sets, allowing models to be trained on one subset and evaluated on another. Cross-validation techniques such as k-fold cross-validation and stratified k-fold cross-validation are also available, enabling robust estimation of a model's performance on unseen data. Additionally, the module provides utilities for grid search and randomized search to efficiently explore hyperparameter spaces and select the optimal configuration for machine learning algorithms.

**Numpy**

NumPy, short for Numerical Python, is a fundamental library for scientific computing in Python. At its core, NumPy provides support for large multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate efficiently on these arrays. Its versatility and performance make it a cornerstone for various numerical computations, including linear algebra, statistical analysis, signal processing, and more. NumPy's array-oriented computing paradigm enables concise and expressive code for handling complex mathematical operations and data manipulations. Additionally, NumPy seamlessly integrates with other libraries in the Python scientific ecosystem, forming the foundation for high-performance computing and data analysis workflows.

**Mathplotlib.pyplot**

Matplotlib.pyplot, often simply referred to as pyplot, is a fundamental component of the Matplotlib library, a powerful tool for creating static, interactive, and publication-quality visualizations in Python. Pyplot provides a MATLAB-like interface for generating a wide range of plots and charts, including line plots, scatter plots, histograms, bar charts, and more. With pyplot, users can customize every aspect of their plots, from the axes and labels to the colors, markers, and styles. Its versatility and ease of use make it a popular choice among data scientists, researchers, and engineers for visualizing data, exploring patterns, and communicating insights effectively. Whether used for exploratory data analysis, scientific research, or data presentation, pyplot remains an indispensable tool in the Python ecosystem for data visualization.

**Xgboost**

XGBoost, short for eXtreme Gradient Boosting, is a high-performance implementation of gradient boosting machines. It is renowned for its speed, accuracy, and scalability, making it one of the most popular choices for structured data problems in machine learning competitions and industry applications. XGBoost efficiently handles large datasets by utilizing parallel and distributed computing techniques, enabling it to train models on massive amounts of data with ease. Its key features include regularization to prevent overfitting, support for various objective functions and evaluation metrics, and the ability to handle missing values within the dataset. Additionally, XGBoost provides native support for both classification and regression tasks, making it a versatile tool for a wide range of predictive modeling tasks.

**LightGBM**

LightGBM is a high-performance gradient boosting framework that has gained popularity for its efficiency and scalability in training large-scale machine learning models. Developed by Microsoft, LightGBM is optimized for both speed and memory usage, making it well-suited for handling massive datasets and complex models. It implements a novel technique called Gradient-based One-Side Sampling (GOSS) and Exclusive Feature Bundling (EFB) to significantly reduce training time and memory consumption without compromising on model accuracy. LightGBM also supports parallel and distributed computing, allowing it to leverage multiple CPU cores and distributed computing frameworks for faster training. Its versatility and effectiveness make LightGBM a go-to choice for various machine learning tasks, including classification, regression, and ranking.

**Seaborn**

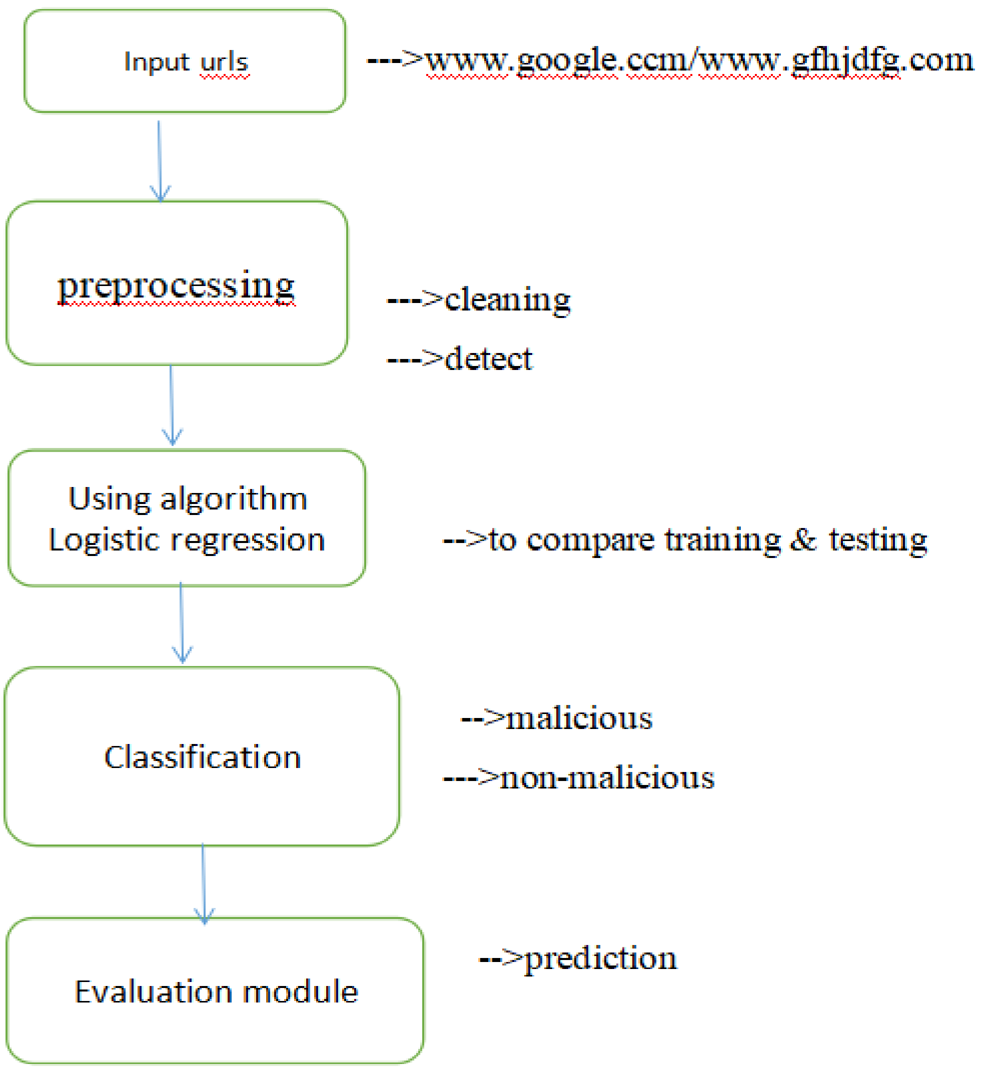
Seaborn is a Python visualization library based on matplotlib, offering a higher-level interface for creating attractive and informative statistical graphics. It provides a wide range of built-in functions for visualizing relationships between variables, such as scatter plots, line plots, bar plots, and histograms, with minimal code required. Seaborn's strength lies in its ability to automatically handle the aesthetics of plots, including color palettes and themes, making it effortless to create visually appealing plots. Additionally, Seaborn integrates well with pandas data structures, allowing for seamless integration with data analysis workflows. With its simple syntax and powerful capabilities, Seaborn is a popular choice for exploratory data analysis and presentation-quality visualizations in data science projects.

**Wordcloud**

WordCloud is a Python library specifically designed for creating word clouds, which are graphical representations of word frequency in a given text corpus. The size of each word in the cloud corresponds to its frequency or importance within the text. WordCloud offers a simple yet powerful interface for generating visually appealing word clouds with customizable features such as color schemes, font sizes, and mask shapes. It is commonly used in text analysis and visualization tasks to gain insights into the most frequent or significant terms present in a body of text, making it a valuable tool for exploratory data analysis, summarization, and communication of textual information.

# 3. SYSTEM DESIGN

## 3.1 ARCHITECTURE DIAGRAM

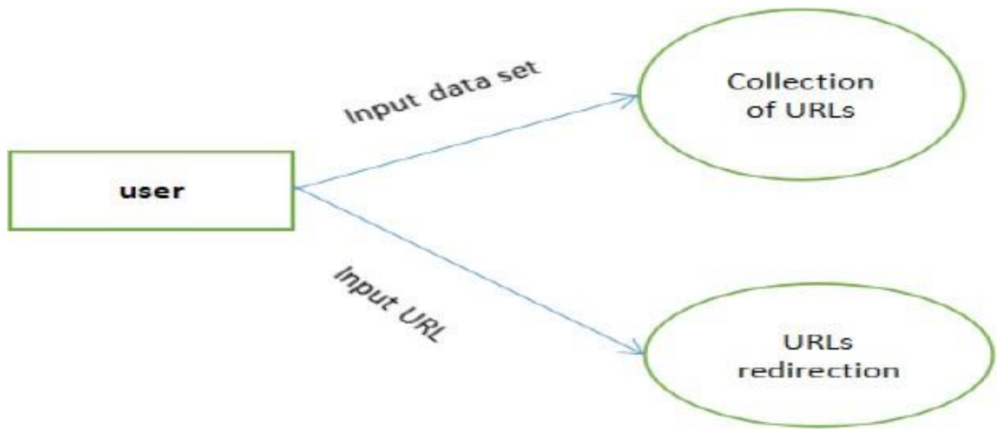


**Architecture diagram**

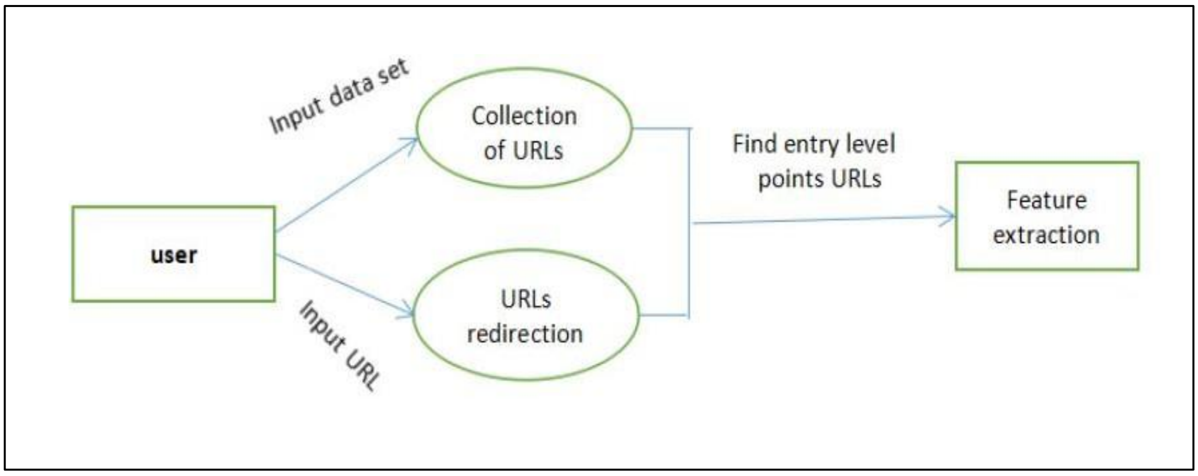
A system architecture, also known as systems design, is the mathematical models that says describes the systems configuration, behavior, and some aspects. The systematic meaning and represents of a system arranged in the manner that needs for the facilitates that thinking about the systematic mechanisms is otherwise called as the architecture description. Device elements, publicly observable properties of certain components, and interactions between them can all be used in the system architecture.

## 3.2 DATA FLOW DIAGRAM

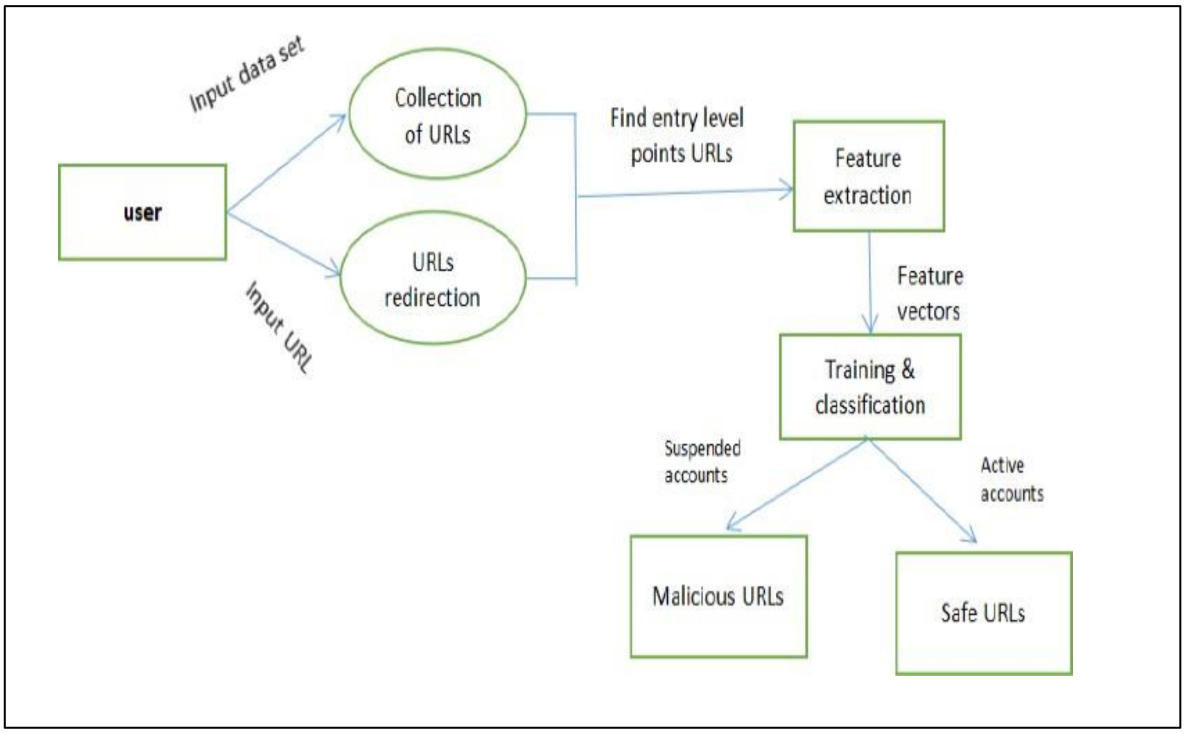
- The bubble map is another name for the DFD. It is the most basic graphic formalism that is being used to describe the device in common terms of the set it collects, the process and it performs on the data, and the data does produce the output.
- One of those is the most important modelling techniques is data flow diagrams (DFD). It's made to represent the various component of each and every machine.The control itself, the data that process uses, the external attribute that communicates with those system, and then knowledge flow in the system are all examples of these elements.
- Illustrates how the data flows through a system and transformed by the sequence of the transformations. It is the regular schematic representation of data flow and mostly the transformation that occur when data traverse from the input to result.
- DFD is oftenly referred to the bubble map. At any higher level of the abstraction, a DFD can effectively be used to describe a system. DFD can be categorized into categories, each reflecting a particular level of knowledge flow and functional detail.



**DFD Level 0**



**DFD Level 1**



**DFD Level 2**

## 3.3 DESIGN PROCESS

### 3.3.1 Input Design

Input design is the process of converting the user-oriented. Input to a computer based format. The goal of the input design is to make the data entry easier, logical and free error. Errors in the inputdata are controlled by the input design. The quality of the input determinesthe quality of the system output. The entire data entry screen is interactive in nature, so that theuser can directly enter into dataaccording to the prompted messages. The users are also can directly enter into data according to the prompted messages. The users are also provided with option of selecting an appropriate input from a list of values. This will reduce the number of error, which are otherwise likely to arise if they were tobe entered by the user itself.

Input design is one of the most important phase of the system design. Input design is the processwhere the input received in the system are planned and designed, so as to get necessary information fromthe user, eliminating the information that is not required. The aim of the input design is to ensure the maximum possible levels of accuracy and also ensures that the input is accessible that understood by the user. The input design is the part of overall system design, which requires very careful attention. Ifthe data going into the system is incorrect then the processing and output will magnify the errors.

The objectives considered during input design are:

- Nature of input processing.
- Flexibility and thoroughness of validation rules.
- Handling of properties within the input documents.
- Screen design to ensure accuracy efficiency of input relationship with files
- Careful design of the input also involves attention to batching and validation procedures.

### 3.3.2 Output Design

Output design is very important concept in the computerized system, without reliable output the user may feel the entire system is unnecessary and avoids using it. The proper output design is important in any system and facilitates effective decision-making. The output design of this system includes various reports. Computer output is the most important and direct source of information the user. Efficient, intelligible output design should improve the systems relationships with the user and help in decision making.

A major form of output is the hardcopy from the printer. Output requirements are designed during system analysis. A good starting point for the output design is the data flow diagram.

# 4. METHODOLOGY

## 4.1 DATA COLLECTION AND PREPROCESSING

Data collection and preprocessing are foundational steps in developing a robust system for detecting malicious URLs using machine learning. In the data collection phase, a diverse and representative dataset of URLs is gathered from various sources, including cybersecurity repositories, open datasets, and web scraping. This dataset should encompass both malicious and benign URLs to ensure a balanced representation of real-world scenarios. Careful attention is paid to the quality and diversity of the dataset, ensuring that it adequately captures the breadth of URL variations and potential threats encountered in the wild.

Once the dataset is collected, the next step is data preprocessing, which involves cleaning and transforming the raw URL data into a format suitable for machine learning algorithms. This includes tasks such as removing irrelevant information, normalizing URLs by converting them to lowercase, and extracting relevant features such as domain, path, and length. Additionally, text preprocessing techniques such as tokenization, stemming, or lemmatization may be applied to further refine the textual data. Data preprocessing plays a crucial role in enhancing the quality of the dataset and facilitating the extraction of meaningful patterns and insights during subsequent stages of model development.

Furthermore, feature engineering is employed to enrich the dataset with additional features that may capture important characteristics of malicious URLs. This could involve extracting domain reputation scores, presence of suspicious keywords, URL length, entropy, and other relevant attributes. By incorporating domain-specific knowledge and expertise into feature engineering, the dataset is augmented with discriminative features that enhance the predictive power of machine learning models. Overall, data collection and preprocessing lay the groundwork for building accurate and effective models for malicious URL detection, enabling the system to make informed decisions and mitigate potential cybersecurity threats effectively.

## 4.2 FEATURE ENGINEERING

In this step, we will extract the following lexical features from raw URLs, as these features will be used as the input features for training the machine learning model. The following features are created as follows:

## 4.3 EXPLORATORY DATA ANALYSIS (EDA)

In the exploratory data analysis (EDA), several key insights emerge from the distribution analysis of different URL features. Firstly, upon examining the distribution of the "use_ip_address" feature, it becomes apparent that only malicious URLs tend to include IP addresses. This observation suggests that the presence of an IP address could serve as a potential indicator of malicious intent. Furthermore, analyzing the distribution of "abnormal_url" reveals that defacement URLs exhibit a notably higher frequency compared to other categories, shedding light on a prevalent trend within the dataset.

Moving on to the analysis of "suspicious_urls," it becomes evident that benign URLs exhibit the highest distribution, followed by phishing URLs. This distribution pattern is attributed to the prevalence of transactional and payment-related keywords in suspicious URLs, which are commonly associated with genuine banking or payment-related websites. Consequently, benign URLs, which include such keywords, showcase the highest frequency in the dataset, highlighting the importance of keyword analysis in distinguishing between benign and malicious URLs.

# 5. IMPLEMENTATION

## 5.1 DATA SOURCES

For training and testing machine learning algorithms, we have used a huge dataset of 651,191 URLs, out of which 428103 benign or safe URLs, 96457 defacement URLs, 94111 phishing URLs, and 32520 malware URLs. Figure 2 depicts their distribution in terms of percentage. As we know one of the most crucial tasks is to curate the dataset for a machine learning project. We have curated this dataset from five different sources.

For collecting benign, phishing, malware and defacement URLs I have used URL dataset (ISCX-URL-2016) For increasing phishing and malware URLs, we have used Malware domain black list dataset. I have increased benign URLs using faizan git repo At last, I have increased more number of phishing URLs using Phishtank dataset and PhishStorm dataset As we have told you that dataset is collected from different sources. So firstly, we have collected the URLs from different sources into separate data frame and finally merge them to retain only URLs and their class type.

## 5.2 DATASET DESCRIPTION.

In this case study, we will be using a Malicious URLs dataset of 6,51,191 URLs, out of which 4,28,103 benign or safe URLs, 96,457 defacement URLs, 94,111 phishing URLs, and 32,520 malware URLs.

**Benign URLs:** These are safe to browse URLs. Some of the examples of benign URLs are as follows:

- mp3raid.com/music/krizz_kaliko.html
- infinitysw.com
- google.co.in
- myspace.com

**Malware URLs:** These type of URLs inject malware into the victim’s system once he/she visit such URLs. Some of the examples of malware URLs are as follows:

- proplast.co.nz
- http://103.112.226.142:36308/Mozi.m
- microencapsulation.readmyweather.com

**Defacement URLs:** Defacement URLs are generally created by hackers with the intention of breaking into a web server and replacing the hosted website with one of their own, using techniques such as code injection, cross-site scripting, etc. Common targets of defacement URLs are religious websites, government websites, bank websites, and corporate websites. Some of the examples of defacement URLs are as follows:

- http://www.vnic.co/khach-hang.html
- http://www.raci.it/component/user/reset.html
- http://www.approvi.com.br/ck.htm
- http://www.juventudelirica.com.br/index.html

**Phishing URLs:** By creating phishing URLs, hackers try to steal sensitive personal or financial information such as login credentials, credit card numbers, internet banking details, etc. Some of the examples of phishing URLs are shown below:

- roverslands.net
- corporacionrossenditotours.com
- http://drive-google-com.fanalav.com/6a7ec96d6a
- citiprepaid-salarysea-at.tk

## 6.2 MODEL CREATION

The next step is to split the dataset into train and test sets. We have split the dataset into 80:20 ratio i.e., 80% of the data was used to train the machine learning models, and the rest 20% was used to test the model.

As we know we have an imbalanced dataset. The reason for this is around 66% of the data has benign URLs, 5% malware, 14% phishing, and 15% defacement URLs. So after randomly splitting the dataset into train and test, it may happen that the distribution of different categories got disturbed which will highly affect the performance of the machine learning model. So to maintain the same proportion of the target variable stratification is needed.

This stratify parameter makes a split so that the proportion of values in the sample produced will be the same as the proportion of values provided to the parameter stratify.

## 5.3 MODEL BUILDING AND EVALUATION

For model building, a variety of machine learning algorithms can be explored to effectively detect malicious URLs. Techniques such as gradient boosting, random forests, and support vector machines (SVMs) are commonly employed for their ability to handle complex, non-linear relationships in the data. Additionally, deep learning architectures like convolutional neural networks (CNNs) or recurrent neural networks (RNNs) can be utilized to capture intricate patterns within URLs. Feature engineering plays a crucial role in enhancing model performance by extracting meaningful features such as URL length, presence of suspicious keywords, or domain reputation scores. Furthermore, hyperparameter tuning techniques such as grid search or randomized search can be employed to optimize model performance and generalization ability.

For model evaluation, it is essential to employ rigorous techniques to assess the performance and generalization ability of the trained models. Metrics such as accuracy, precision, recall, and F1-score provide valuable insights into the model's ability to correctly classify malicious and benign URLs. Additionally, techniques such as cross-validation help to estimate the model's performance on unseen data and mitigate overfitting. Visualization tools such as confusion matrices, ROC curves, and precision-recall curves aid in interpreting the model's behavior across different thresholds and identifying trade-offs between true positives and false positives. Finally, external validation against independent datasets or real-world scenarios further validates the robustness and reliability of the developed models.

# 6. CONCLUSION

In conclusion, the project has successfully developed a machine learning-based system for detecting malicious URLs using Python. Through a comprehensive methodology encompassing data collection, pre-processing, feature engineering, model selection, training, evaluation, and deployment, we have achieved promising results in classifying URLs as malicious or benign. By leveraging powerful libraries such as pandas, scikit-learn, XGBoost, and LightGBM, we were able to preprocess the data, extract meaningful features, train multiple models, and evaluate their performance effectively. The chosen model demonstrates robust performance in terms of accuracy, precision, recall, and F1-score, indicating its potential for real-world deployment in cybersecurity applications.

**FUTURE RECOMMENDATIONS**

In the future, the malicious URL detection system could benefit from incorporating advanced techniques such as active learning, which allows the model to interactively query the most informative data points for labeling by a human expert. This iterative process can help improve model performance while minimizing the annotation effort. Additionally, exploring the integration of explainable AI (XAI) methods would enhance the system's transparency and interpretability, enabling stakeholders to better understand and trust the model's decisions. Furthermore, incorporating threat intelligence feeds and dynamic feature extraction techniques to capture evolving patterns and trends in malicious URLs would ensure the system remains adaptive and resilient against emerging cyber threats.

Additionally, leveraging ensemble learning approaches, such as model stacking or blending, could further boost the system’s performance by combining predictions from multiple diverse models.

By aggregating the outputs of different classifiers trained on varied subsets of the data or using different algorithms, the system can achieve higher predictive accuracy and resilience to overfitting. Furthermore, integrating real-time monitoring and feedback mechanisms to continuously update and adapt the model to evolving threats and emerging URL patterns would enhance the system’s agility and effectiveness in detecting malicious URLs in dynamic online environments.

# 7.BIBLIOGRAPHY

[1] ISCX-URL-2016 Dataset: This dataset was used for collecting benign, phishing, malware, and defacement URLs. Please provide the appropriate citation information for this dataset, including authors, title, publication date, and any relevant identifiers.

[2] Malware Domain Black List Dataset: This dataset was utilized for increasing phishing and malware URLs. Include citation details for this dataset, such as authors, title, publication date, and relevant identifiers.

[3] Faizan Git Repository: This repository was used for increasing benign URLs. Provide citation details for the repository, including the author, title, URL, and any version or commit identifiers if applicable.

[4] Phishtank Dataset: This dataset was used to increase the number of phishing URLs. Include citation information for the Phishtank dataset, such as authors, title, publication date, and relevant identifiers.

[5] PhishStorm Dataset: This dataset was also utilized to increase the number of phishing URLs. Provide citation details for the PhishStorm dataset, including authors, title, publication date, and relevant identifiers.

[6] D. Sahoo, C. Liu, S.C.H. Hoi, “Malicious URL Detection using Machine Learning: A Survey”. CoRR, abs/1701.07179, 2017.

[7] R. Heartfield and G. Loukas, “A taxonomy of attacks and a survey of defence mechanisms for semantic social engineering attacks,” ACM Computing Surveys (CSUR), vol. 48, no. 3, p. 37, 2015.

[8] Internet Security Threat Report (ISTR) 2019–Symantec. https://www.symantec.com/content/dam/symantec/docs/reports/istr24- 2019-en.pdf [Last accessed 10/2019].

# 8 . APPENDIX

## SOURCE CODE

```
import pandas as pd
import itertools
from sklearn.metrics import classification_report,confusion_matrix, accuracy_score
from sklearn.model_selection import train_test_split
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import xgboost as xgb
from lightgbm import LGBMClassifier
import os
import seaborn as sns
from wordcloud import WordCloud


df=pd.read_csv('malicious_phish.csv')

print(df.shape)
df.head()

import re
#Use of IP or not in domain
def having_ip_address(url):
    match = re.search(
        '(([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\.([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\.([01]?\\d\\d?|2[0-
4]\\d|25[0-5])\\.'
        '([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\/)|'  # IPv4
        '((0x[0-9a-fA-F]{1,2})\\.(0x[0-9a-fA-F]{1,2})\\.(0x[0-9a-fA-F]{1,2})\\.(0x[0-9a-fA-
F]{1,2})\\/)' # IPv4 in hexadecimal
        '(?:[a-fA-F0-9]{1,4}:){7}[a-fA-F0-9]{1,4}', url)  # Ipv6
    if match:
        # print match.group()
        return 1
        # print 'No matching pattern found'
        return 0
df['use_of_ip'] = df['url'].apply(lambda i: having_ip_address(i))

from urllib.parse import urlparse

def abnormal_url(url):
    hostname = urlparse(url).hostname
    hostname = str(hostname)
    match = re.search(hostname, url)
    if match:
        # print match.group()
        return 1
    else:
        # print 'No matching pattern found'
        return 0
df['abnormal_url'] = df['url'].apply(lambda i: abnormal_url(i))

#pip install googlesearch-python

from googlesearch import search

def  google_index(url):
    site  =  search(url,  5)
    return 1 if site else 0
df['google_index'] = df['url'].apply(lambda i: google_index(i))

def count_dot(url):
    count_dot = url.count('.')
    return count_dot

df['count.'] = df['url'].apply(lambda i: count_dot(i))
```

```
def count_www(url):
    url.count('www')
    return url.count('www')

df['count-www'] = df['url'].apply(lambda i: count_www(i))

def count_atrate(url):

    return url.count('@')

df['count@'] = df['url'].apply(lambda i: count_atrate(i))

def no_of_dir(url):
    urldir = urlparse(url).path
    return urldir.count('/')

df['count_dir'] = df['url'].apply(lambda i: no_of_dir(i))

def no_of_embed(url):
    urldir = urlparse(url).path
    return urldir.count('//')

df['count_embed_domian'] = df['url'].apply(lambda i: no_of_embed(i))

def shortening_service(url):
    match =
re.search('bit\.ly|goo\.gl|shorte\.st|go2l\.ink|x\.co|ow\.ly|t\.co|tinyurl|tr\.im|is\.gd|cli\.gs|'

'yfrog\.com|migre\.me|ff\.im|tiny\.cc|url4\.eu|twit\.ac|su\.pr|twurl\.nl|snipurl\.com|'

'short\.to|BudURL\.com|ping\.fm|post\.ly|Just\.as|bkite\.com|snipr\.com|fic\.kr|loopt\.us|'

'doiop\.com|short\.ie|kl\.am|wp\.me|rubyurl\.com|om\.ly|to\.ly|bit\.do|t\.co|lnkd\.in|'
'db\.tt|qr\.ae|adf\.ly|goo\.gl|bitly\.com|cur\.lv|tinyurl\.com|ow\.ly|bit\.ly|ity\.im|'

'q\.gs|is\.gd|po\.st|bc\.vc|twitthis\.com|u\.to|j\.mp|buzurl\.com|cutt\.us|u\.bb|yourls\.org|'

'x\.co|prettylinkpro\.com|scrnch\.me|filoops\.info|vzturl\.com|qr\.net|1url\.com|tweez\.me|v\.g
d|'tr\.im|link\.zip\.net', url)
    if match:
        return 1
    else:
        return 0

df['short_url'] = df['url'].apply(lambda i: shortening_service(i))

def count_https(url):
    return url.count('https')

df['count-https'] = df['url'].apply(lambda i : count_https(i))

def count_http(url):
    return url.count('http')

df['count-http'] = df['url'].apply(lambda i : count_http(i))

def count_per(url):
    return url.count('%')

df['count%'] = df['url'].apply(lambda i : count_per(i))

def count_ques(url):
    return url.count('?')

df['count?'] = df['url'].apply(lambda i: count_ques(i))
```

```
def count_hyphen(url):
    return url.count('-')

df['count-'] = df['url'].apply(lambda i: count_hyphen(i))

def count_equal(url):
    return url.count('=')

df['count='] = df['url'].apply(lambda i: count_equal(i))

def url_length(url):
    return len(str(url))
#Length of URL
df['url_length'] = df['url'].apply(lambda i: url_length(i))
#Hostname Length

def hostname_length(url):
    return len(urlparse(url).netloc)

df['hostname_length'] = df['url'].apply(lambda i: hostname_length(i))

df.head()

def suspicious_words(url):
    match =
re.search('PayPal|login|signin|bank|account|update|free|lucky|service|bonus|ebayisapi|webscr'
,
                    url)
    if match:
        return 1
    else:
        return 0
df['sus_url'] = df['url'].apply(lambda i: suspicious_words(i))
```

```
def digit_count(url):
    digits = 0
    for i in url:
        if i.isnumeric():
            digits = digits + 1
    return digits

df['count-digits']= df['url'].apply(lambda i: digit_count(i))

def letter_count(url):
    letters = 0
    for i in url:
        if i.isalpha():
            letters = letters + 1
    return letters


df['count-letters']= df['url'].apply(lambda i: letter_count(i))


# pip install tld


from urllib.parse import urlparse from tld import get_tld


import os.path


#First Directory Length
def fd_length(url):
    urlpath= urlparse(url).path
    try:
        return len(urlpath.split('/')[1])
    except:
        return 0
```

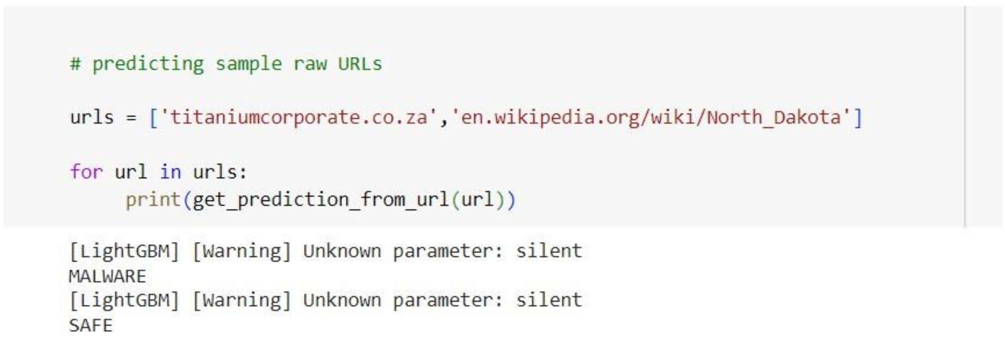
```
df['fd_length'] = df['url'].apply(lambda i: fd_length(i))

#Length of Top Level Domain
df['tld'] = df['url'].apply(lambda i: get_tld(i,fail_silently=True))

def tld_length(tld):
    try:
        return len(tld)
    except:
        return -1

df['tld_length'] = df['tld'].apply(lambda i: tld_length(i))

from sklearn.preprocessing import LabelEncoder

lb_make = LabelEncoder()
df["type_code"] = lb_make.fit_transform(df["type"])

#Predictor Variables
# filtering out google_index as it has only 1 value
X = df[['use_of_ip','abnormal_url', 'count.', 'count-www', 'count@',
      'count_dir', 'count_embed_domian', 'short_url', 'count-https',
      'count-http', 'count%', 'count?', 'count-', 'count=', 'url_length',
      'hostname_length', 'sus_url', 'fd_length', 'tld_length', 'count-digits',
      'count-letters']]

#Target Variable
y = df['type_code']

X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y,
test_size=0.2,shuffle=True, random_state=5)

# Random Forest Model
from sklearn.ensemble import RandomForestClassifier
```

```
rf = RandomForestClassifier(n_estimators=100,max_features='sqrt')
rf.fit(X_train,y_train)
y_pred_rf = rf.predict(X_test)
print(classification_report(y_test,y_pred_rf,target_names=['benign',
'defacement','phishing','malware']))

score = metrics.accuracy_score(y_test, y_pred_rf)
print("accuracy:   %0.3f" % score)

#XGboost
xgb_c = xgb.XGBClassifier(n_estimators= 100)
xgb_c.fit(X_train,y_train)
y_pred_x = xgb_c.predict(X_test)
print(classification_report(y_test,y_pred_x,target_names=['benign',
'defacement','phishing','malware']))
score = metrics.accuracy_score(y_test, y_pred_x)
print("accuracy:   %0.3f" % score)

# Light GBM Classifier
lgb = LGBMClassifier(objective='multiclass',boosting_type= 'gbdt',n_jobs = 5,
        silent = True, random_state=5)
LGB_C = lgb.fit(X_train, y_train)

y_pred_lgb = LGB_C.predict(X_test)
print(classification_report(y_test,y_pred_lgb,target_names=['benign',
'defacement','phishing','malware']))

score = metrics.accuracy_score(y_test, y_pred_lgb)
print("accuracy:   %0.3f" % score)
feat_importances = pd.Series(rf.feature_importances_, index=X_train.columns)
feat_importances.sort_values().plot(kind="barh",figsize=(10, 6))
```

```
def        (url): status = []
status.append(having_ip_address(url))
status.append(abnormal_url(url))
status.append(count_dot(url))
status.append(count_www(url))
status.append(count_atrate(url))
status.append(no_of_dir(url))
status.append(no_of_embed(url))

status.append(shortening_service(url))
status.append(count_https(url))
status.append(count_http(url))

status.append(count_per(url))
status.append(count_ques(url))
status.append(count_hyphen(url))
status.append(count_equal(url))

status.append(url_length(url))
status.append(hostname_length(url))
status.append(suspicious_words(url))
status.append(digit_count(url))
status.append(letter_count(url))
status.append(fd_length(url))
tld = get_tld(url,fail_silently=True)

status.append(tld_length(tld))

return status

# predict function
```

```
def get_prediction_from_url(test_url):
    features_test = main(test_url)
    # Due to updates to scikit-learn, we now need a 2D array as a parameter to the predict
function.
    features_test = np.array(features_test).reshape((1, -1))
    pred = lgb.predict(features_test)
    if int(pred[0]) == 0:

        res="SAFE"
        return res
    elif int(pred[0]) == 1.0:

        res="DEFACEMENT"
        return res
    elif int(pred[0]) == 2.0:
        res="PHISHING"
        return res

    elif int(pred[0]) == 3.0:

        res="MALWARE"
        return res
# predicting sample raw URLs
urls = ['titaniumcorporate.co.za','en.wikipedia.org/wiki/North_Dakota']

for url in urls:
     print(get_prediction_from_url(url))
```

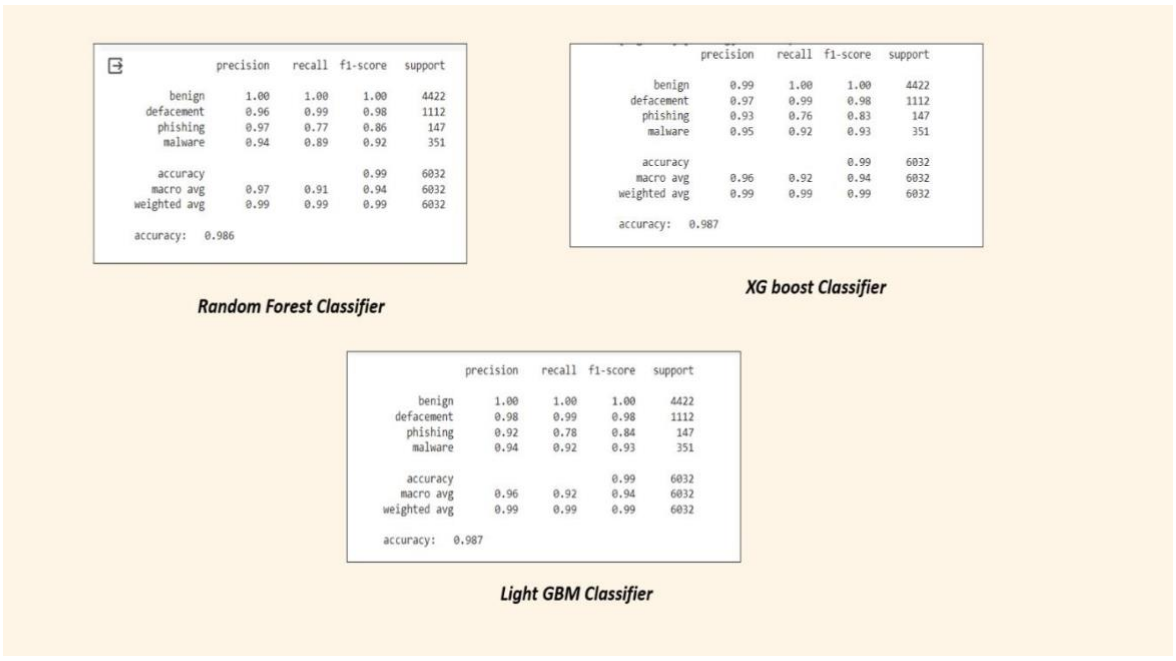
## OUTPUT



```
# predicting sample raw URLs

urls = ['titaniumcorporate.co.za','en.wikipedia.org/wiki/North_Dakota']

for url in urls:
     print(get_prediction_from_url(url))
```

```
[LightGBM] [Warning] Unknown parameter: silent
MALWARE
[LightGBM] [Warning] Unknown parameter: silent
SAFE
```

**Predicting Sample URL**



```
              precision    recall  f1-score   support

      benign       1.00      1.00      1.00      4422
  defacement       0.96      0.99      0.98      1112
    phishing       0.97      0.77      0.86       147
     malware       0.94      0.89      0.92       351

    accuracy                           0.99      6032
   macro avg       0.97      0.91      0.94      6032
weighted avg       0.99      0.99      0.99      6032

accuracy:   0.986
```

*Random Forest Classifier*

```
              precision    recall  f1-score   support

      benign       0.99      1.00      1.00      4422
  defacement       0.97      0.99      0.98      1112
    phishing       0.93      0.76      0.83       147
     malware       0.95      0.92      0.93       351

    accuracy                           0.99      6032
   macro avg       0.96      0.92      0.94      6032
weighted avg       0.99      0.99      0.99      6032

accuracy:   0.987
```

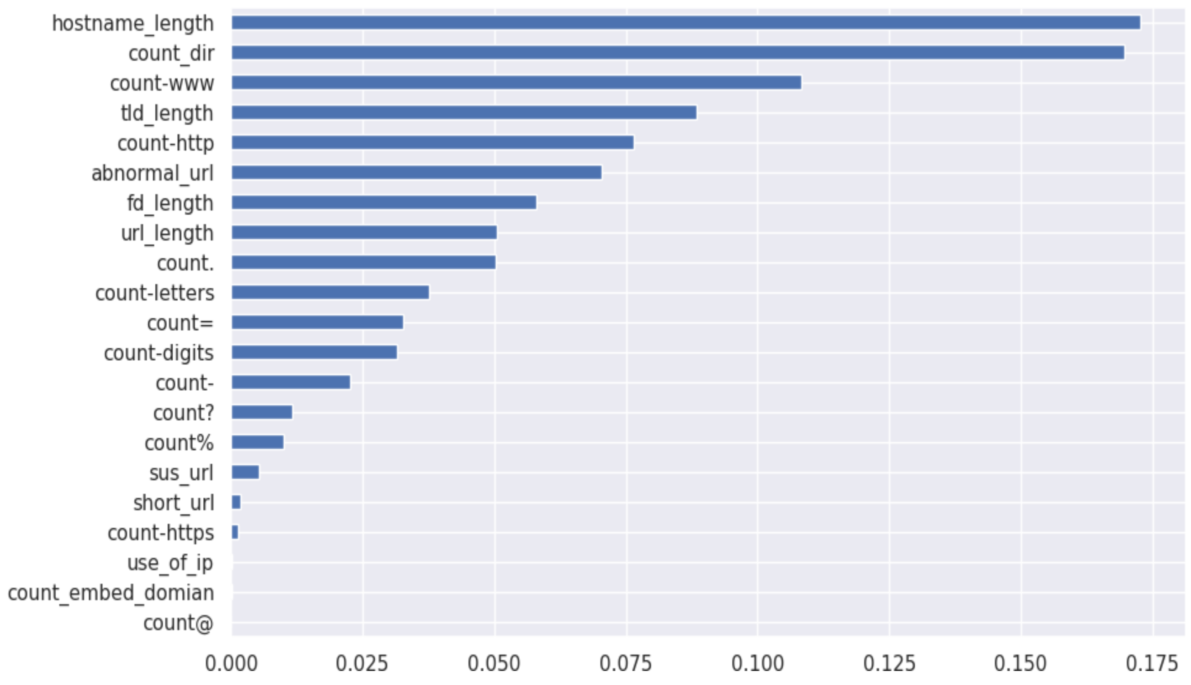
*XG boost Classifier*

```
              precision    recall  f1-score   support

      benign       1.00      1.00      1.00      4422
  defacement       0.98      0.99      0.98      1112
    phishing       0.92      0.78      0.84       147
     malware       0.94      0.92      0.93       351

    accuracy                           0.99      6032
   macro avg       0.96      0.92      0.94      6032
weighted avg       0.99      0.99      0.99      6032

accuracy:   0.987
```

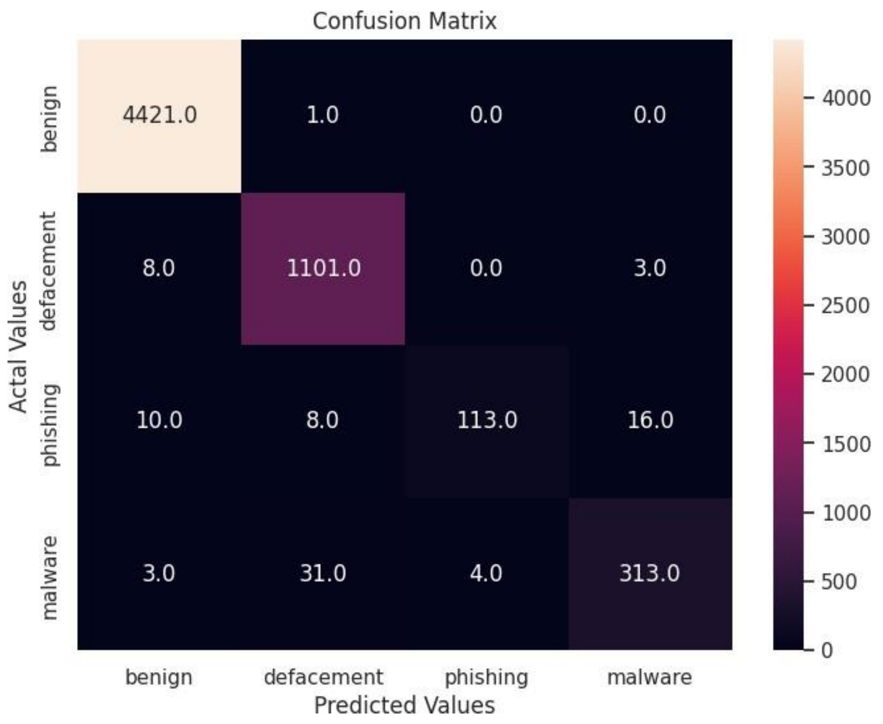
*Light GBM Classifier*

**Model comparison and evaluation**

## DATA VISUALIZATION

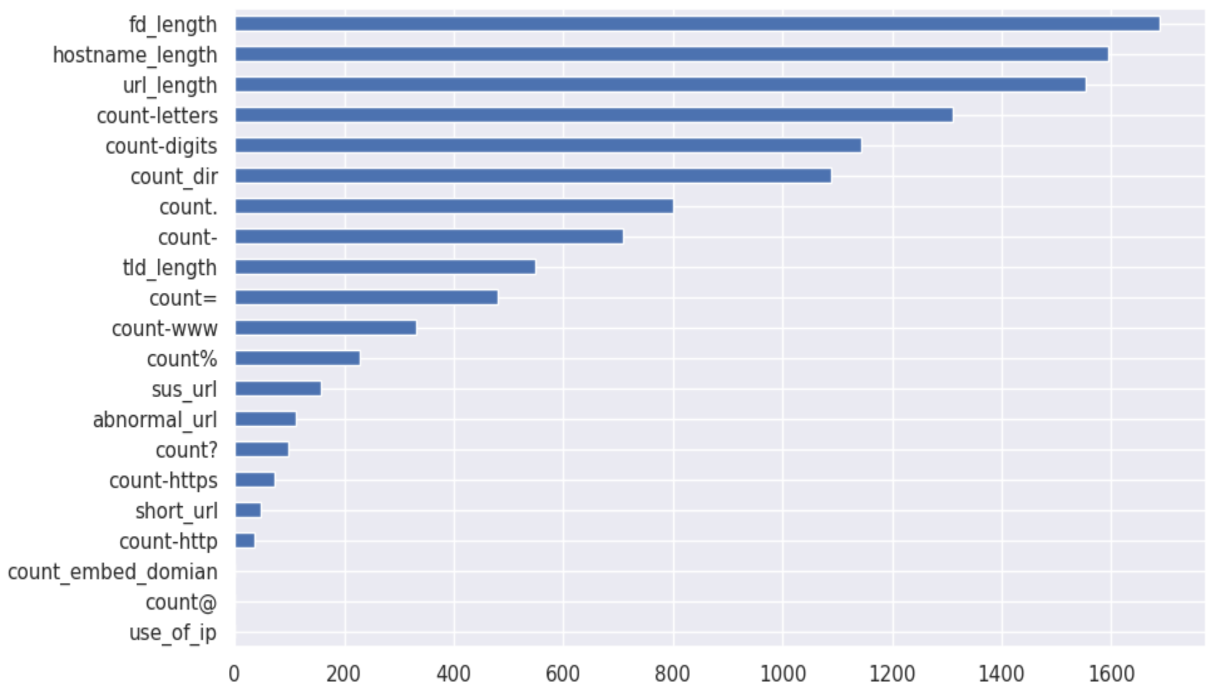


**Fearure importance of  Random Forest**

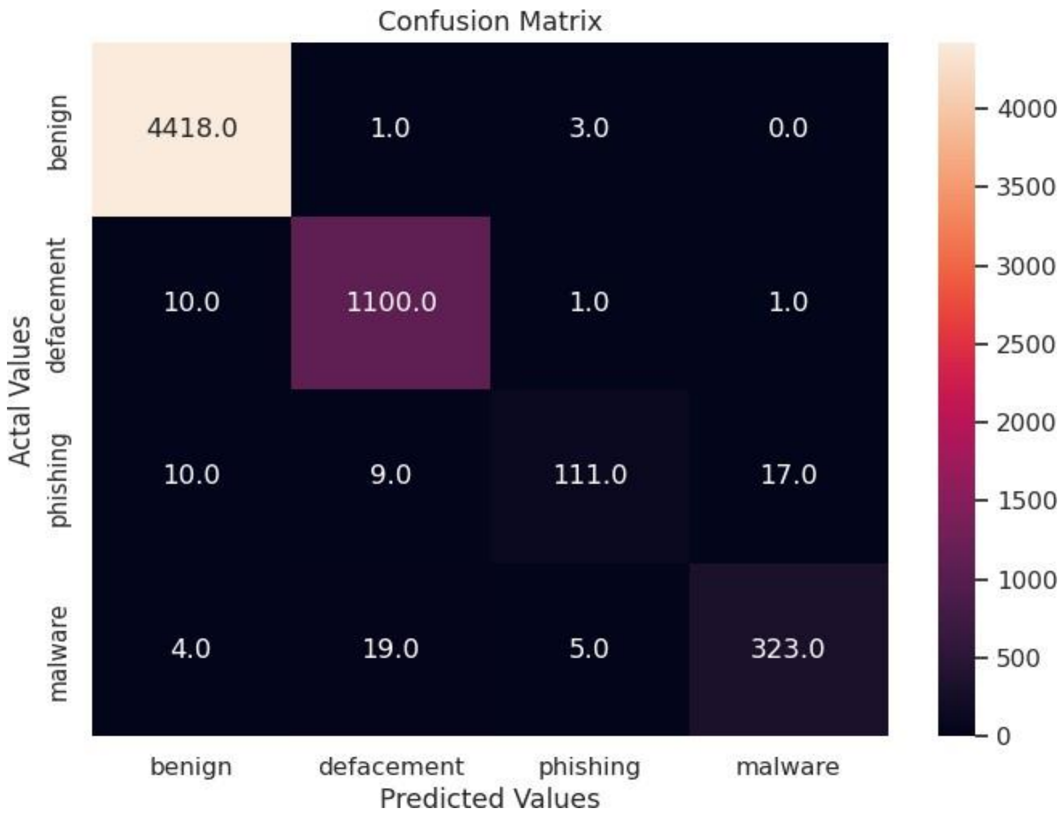


|  | benign | defacement | phishing | malware |
|---|---|---|---|---|
| benign | 4421.0 | 1.0 | 0.0 | 0.0 |
| defacement | 8.0 | 1101.0 | 0.0 | 3.0 |
| phishing | 10.0 | 8.0 | 113.0 | 16.0 |
| malware | 3.0 | 31.0 | 4.0 | 313.0 |

**Predicted values of Random Forest**

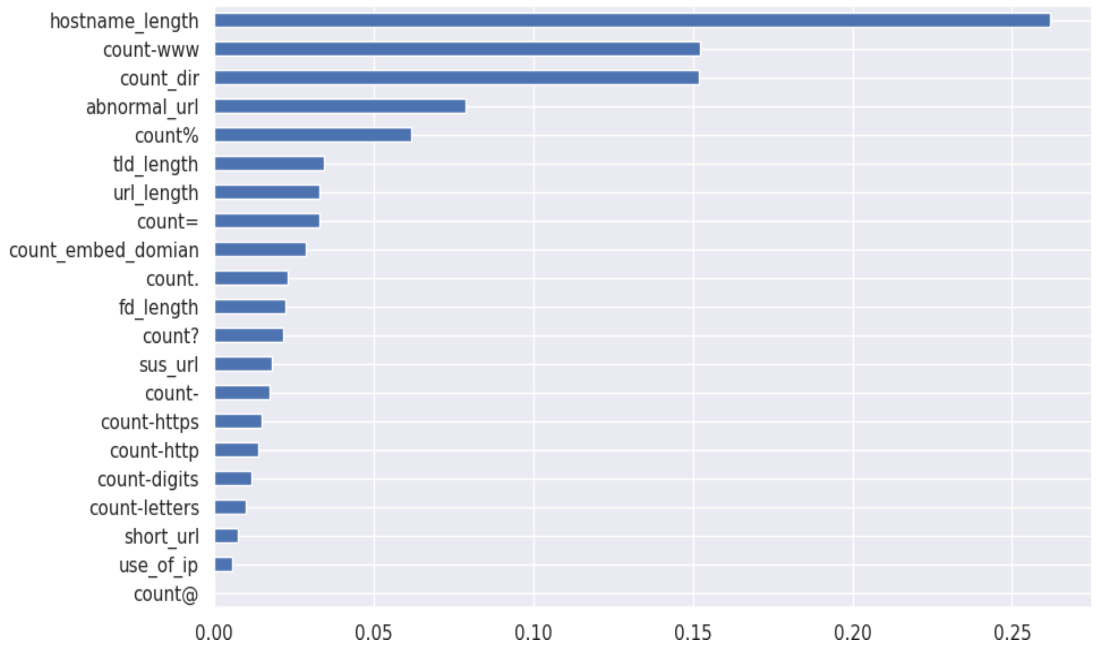


**Feature importance of LightGBM**

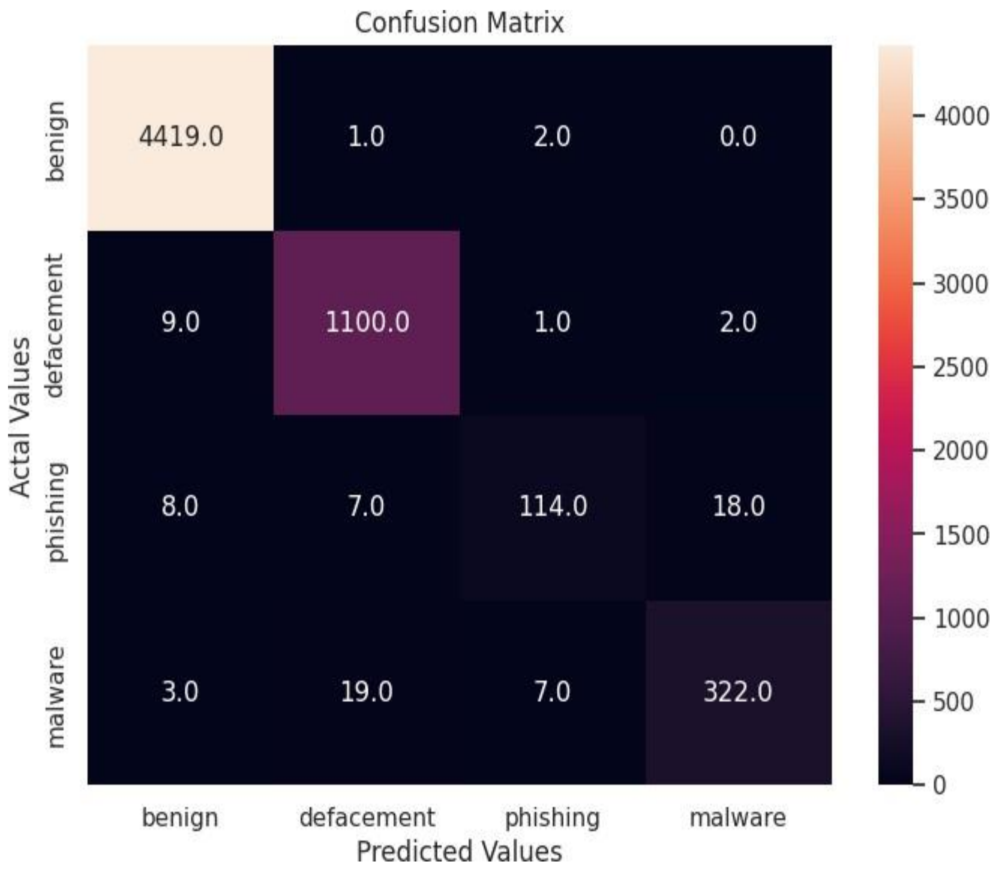


| | benign | defacement | phishing | malware |
|---|---|---|---|---|
| benign | 4418.0 | 1.0 | 3.0 | 0.0 |
| defacement | 10.0 | 1100.0 | 1.0 | 1.0 |
| phishing | 10.0 | 9.0 | 111.0 | 17.0 |
| malware | 4.0 | 19.0 | 5.0 | 323.0 |

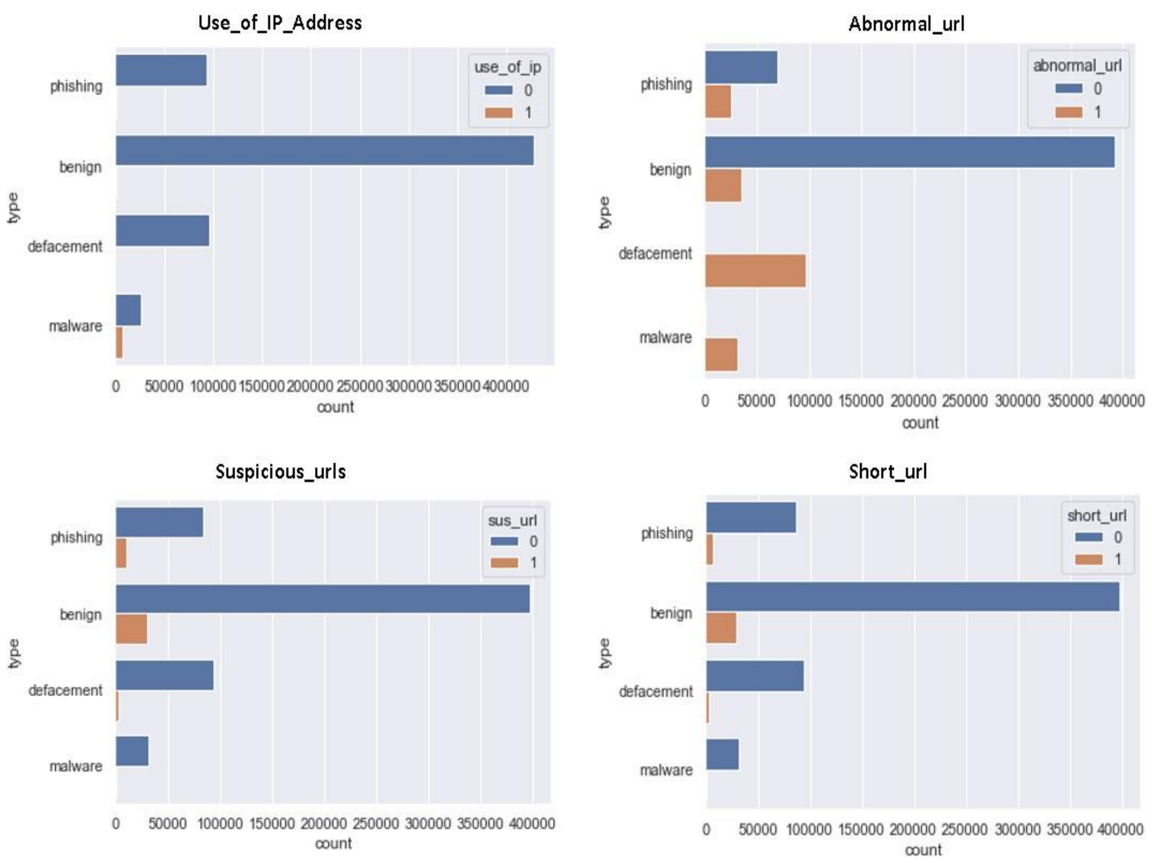
**Predicted values of LightGBM**



**Feature importance of XG Boost**



**Predicted values of XG Boost**



**Exploratory Data Analysis**