

A

Major Project Report

On

URL RISK ASSESSMENT USING MACHINE LEARNING

*Submitted to CMR Engineering College (UGC AUTONOMOUS)
, HYDERABAD*

In Partial Fulfillment of the requirements for the Award of Degree of

**BACHELOR OF TECHNOLOGY
IN COMPUTER SCIENCE AND ENGINEERING - DATA
SCIENCE**

Submitted

By

G.Bhavya (218R1A6780)

O. Kalyan Ram (228R5A6710)

A.Venkatesh (218R1A6767)

R.VenkataHema (218R1A67C0)

Under the Esteemed guidance of

B.Kumaraswamy

Assistant Professor -Data Science



Department of Computer Science & Engineering -Data Science

CMR ENGINEERING COLLEGE

UGC AUTONOMOUS

(Approved by AICTE, NEW DELHI, Affiliated to JNTU, Hyderabad)

Kandlakoya, Medchal Road, R.R. Dist. Hyderabad-501 401.

2024-2025

CMR ENGINEERING COLLEGE

UGC AUTONOMOUS

(Accredited by NBA, Approved by AICTE NEW DELHI, Affiliated to JNTU, Hyderabad)

Kandlakoya, Medchal Road, Hyderabad-501 401

Department of Computer Science & Engineering - Data Science



CERTIFICATE

This is to certify that the project entitled “URL RISK ASSESMENT USING MACHINE LEARNING” is a bonafide work carried out by

G.Bhavya (218R1A6780)

O.Kalyan Ram (228R5A6710)

A.Venkatesh (218R1A6767)

R.VenkataHema(218R1A67C0)

in partial fulfillment of the requirement for the award of the degree of **BACHELOR OF TECHNOLOGY** in **COMPUTER SCIENCE AND ENGINEERING (DATA SCIENCE)** from CMR Engineering College, affiliated to JNTU, Hyderabad, under our guidance and supervision.

The results presented in this Major project have been verified and are found to be satisfactory. The results embodied in this Major project have not been submitted to any other university for the award of any other degree or diploma.

Internal Guide	Major Project Coordinator	Head of the Department	External Examiner
Mr.B.Kumaraswamy	Mr. B. KumaraSwamy (Ph. D, JNTUH)	Dr. M. Laxmaiah	
Associate Professor	Associate Professor	Professor & HOD	
CSE (Data Science), CMREC	CSE (Data Science), CMREC	CSE (Data Science), CMREC	

DECLARATION

This is to certify that the work reported in the present Major project entitled "**URL RISK ASSESSMENT USING MACHINE LEARNING**" is a record of bonafide work done by us in the Department of Computer Science and Engineering (Data Science), CMR Engineering College, JNTU Hyderabad. The reports are based on the project work done entirely by us and not copied from any other source. We submit our project for further development by any interested students who share similar interests to improve the project in the future.

The results embodied in this Major project report have not been submitted to any other University or Institute for the award of any degree or diploma to the best of our knowledge and belief.

G.Bhavya	(218R1A6780)
O.Kalyan Ram	(228R5A6710)
A.Venkatesh	(218R1A6767)
R.VenkataHema	(218R1A67C0)

ACKNOWLEDGMENT

We are extremely grateful to **Dr. A. Srinivasula Reddy**, Principal and **Dr. M. Laxmaiah**, HOD, **Department of CSE (Data Science), CMR Engineering College** for their constant support.

We are extremely thankful to **Mr.B.Kumaraswamy**, Associate Professor, Internal Guide, Department of CSE(DS), for his/ her constant guidance, encouragement and moral support throughout the project.

We will be failing in duty if We do not acknowledge with grateful thanks to the authors of the references and other literatures referred in this Project.

We thank **our** , Associate Professor ,CSE(DS) Department ,Major Project Coordinator for his constant support in carrying out the project activities and reviews.

We express my thanks to all staff members and friends for all the help and co-ordination extended in bringing out this project successfully in time.

Finally, We are very much thankful to our parents who guided me for every step.

G.Bhavya	(218R1A6780)
O.Kalyan Ram	(218R1A67C3)
A.Venkatesh	(218R1A67C4)
R.VenkataHema	(218R1A67B2)

ABSTRACT

The goal of this project is to protect users from online threats such as phishing and malware by detecting harmful URLs using machine learning techniques. For instance, phishing websites often mimic legitimate ones—like bank portals—to steal sensitive information such as login credentials.

To identify risky URLs, the system analyzes features including URL length, HTTPS usage, website age, and unusual characters. Machine learning models like XGBoost and Random Forest are employed due to their ability to handle large datasets and improve decision-making accuracy. Hyperparameter tuning is used to enhance model performance.

Once trained, the system is rigorously tested for reliability. A user-friendly website interface is then developed, allowing users to input a URL and instantly check its safety. The dataset used for model training and validation is sourced from Phishing Tank, a well-known repository of phishing and legitimate URLs. After training, the system is integrated with a Graphical User Interface (GUI) using Python libraries like Tkinter and PyQt, making it accessible to non-technical users. The final product allows users to input a URL and receive real-time feedback on its legitimacy.

The system also incorporates a modular design that simplifies updates and maintenance. By using Anaconda for dependency management and environment setup, the installation process is streamlined, making the system highly portable and easy to deploy. This proactive solution aims to reduce phishing incidents and make web browsing safer for everyone.

CONTENTS

	TOPIC	PAGE NO
	ABSTRACT	5
	LIST OF FIGURES	7
1.	INTRODUCTION	
	1.1. Overview	1
	1.2. Research Motivation	2
	1.3. Problem Statement	3
	1.4. Application	4
2.	LITERATURE SURVEY	6
3.	EXISTING SYSTEM	
	3.1. Random Forest	10
	3.2. Random Forest Working and Drawbacks	11
4.	PROPOSED SYSTEM	
	4.1. Overview of proposed system	21
5.	UML DIAGRAMS	
	5.1. Class Diagram	25
	5.2. Use Case Diagram	26
	5.3. Sequence Diagram	27
	5.4. Activity Diagram	28
6.	SOFTWARE ENVIRONMENT	
	6.1. What is python and its Advantages and Disadvantages	29
	6.2. History of python	33
	6.3. Modules used in project	35
	6.4. Installation of python	37
7.	SYSTEM REQUIREMENTS SPECIFICATIONS	
	7.1. Software Requirements	44
	7.2. Hardware Requirements	44
8.	FUNCTIONAL REQUIREMENTS	
	8.1. Output Design and Definition	45
	8.2. Input Design , Stages, Types, Media	46
	8.3. User Interface	47
	8.4. Performance Requirements	48
9.	SOURCE CODE	50
10.	RESULTS AND DISCUSSION	
	10.1. Implementation description	53
	10.2. Dataset Description	54
11.	CONCLUSION	62
12.	REFERENCES	63

LIST OF FIGURES

Fig. No.	DESCRIPTION	PAGE	NO
3.1	Random Forest Example		8
4.1	XGBoost Methodology		14
5.1	Class Diagram		16
5.2	Usecase Diagram		17
5.3	Sequence Diagram		17
5.4	Activity Diagram		18
6	Python installation Diagrams		28
10.1	Preview of Dataset for Autism Spectrum Disorder Detection		49
10.2	Dataset Structure and Data Types Overview		50
10.3	Feature Distributions and Correlations		50
10.4	Feature Distributions by ASD Class		51
10.5	Distribution of ASD Cases by Country of Residence		51
10.6	Correlation Heatmap of Dataset Features		52
10.7	Confusion Matrix for ASD Classification Model		53

1. Introduction

1.1 Overview

In today's digital era, the internet has become an essential part of daily life, enabling online banking, shopping, communication, and data sharing. However, with this growing dependence comes an increase in cyber threats—especially phishing attacks. Phishing is a deceptive practice where attackers trick users into visiting fake websites designed to steal sensitive information such as usernames, passwords, and credit card details. These attacks are often carried out using malicious URLs that appear genuine but redirect users to harmful or fraudulent sites.

Traditional methods for detecting such threats primarily rely on blacklists, which block access to known malicious URLs. To address these challenges, this project proposes a machine learning-based approach for URL risk assessment. By training models on large datasets of legitimate and phishing URLs, the system can identify complex patterns that distinguish safe links from harmful ones. Features such as the length of the URL, use of secure protocols like HTTPS, presence of special characters, and domain age are analyzed to make informed predictions. This allows the system to detect potentially malicious URLs even if they are new or have never been reported before.

Machine learning models such as Random Forest and XGBoost are used due to their high accuracy and ability to handle diverse data. These models are further enhanced through hyperparameter tuning to improve their performance. The result is a dynamic and reliable system capable of offering real-time URL safety checks. This not only enhances user protection but also contributes to the broader goal of making the internet a safer place.

To make the solution accessible, the model is integrated into a user-friendly Graphical User Interface (GUI) built using Python's Tkinter and PyQt libraries. This allows end-users to input a URL and instantly receive feedback about its safety, without needing to understand the underlying machine learning mechanisms. The GUI provides real-time predictions, making the system suitable for both technical and non-technical users. This project demonstrates how machine learning can play a pivotal role in enhancing cybersecurity infrastructure. By proactively identifying and classifying malicious URLs, the system reduces the risk of phishing attacks and promotes safer online behavior. As the threat landscape continues to evolve, intelligent solutions like this one will become increasingly important in protecting users and organizations alike.

1.2 Research Motivation

The motivation behind this research stems from the alarming rise in cybercrime, particularly phishing attacks, which exploit human vulnerabilities and lack of awareness in digital spaces. As the internet continues to grow in importance for personal, professional, and financial activities, the risks associated with malicious content, especially deceptive URLs, have become a critical concern. Phishing attacks not only lead to significant financial losses but also compromise user trust in online services and platforms.

Traditional detection techniques such as blacklists and heuristic-based systems are no longer sufficient in addressing the dynamic and rapidly evolving nature of phishing threats. Blacklists can only detect known malicious URLs, offering no protection against newly created or slightly altered phishing links. Similarly, heuristic methods require manually defined rules, which are both time-consuming to update and limited in flexibility.

These limitations underscore the need for a more scalable and intelligent approach. This project is motivated by the potential of machine learning to provide a smarter, adaptive, and automated solution to the problem. By training models on real-world data and extracting meaningful patterns from malicious and benign URLs, machine learning can detect suspicious behavior without relying solely on known threat signatures. This capability makes ML-based systems not only more accurate but also more proactive in identifying potential threats.

Furthermore, with the increasing availability of public datasets and open-source tools, developing a robust and user-friendly URL risk assessment tool has become feasible even for small-scale research teams. Integrating such a system into a graphical user interface (GUI) ensures accessibility for all users, regardless of technical expertise. In essence, this research is driven by the urgent need to create a cost-effective, accurate, and easily deployable solution that bridges the gap between growing cyber threats and the limitations of existing protection mechanisms. It reflects a commitment to leveraging emerging technologies to safeguard users and promote safer internet usage in an increasingly digital world.

Lastly, the motivation behind this research is also rooted in the societal impact of phishing attacks. Beyond financial losses, phishing can lead to identity theft, loss of privacy, reputational damage, and emotional distress. By developing a system that empowers users

to assess URL safety in real time, this project contributes to a broader mission of building a safer digital environment and promoting cyber awareness among the general public.

1.3 Problem Statement

The rapid digitalization of services and the widespread use of the internet have significantly increased user dependence on web-based platforms for communication, banking, shopping, and data management. However, this digital convenience has been accompanied by an alarming rise in cyber threats—most notably, phishing attacks that exploit user trust through deceptive and malicious URLs. These phishing websites are designed to appear legitimate, often mimicking trusted platforms like financial institutions, e-commerce websites, or government portals, with the goal of stealing sensitive personal and financial information.

Traditional cybersecurity measures such as blacklist-based URL filtering and heuristic rule systems are no longer sufficient to combat the growing sophistication and volume of phishing attacks. Blacklists can only detect URLs that have already been identified and reported, leaving users vulnerable to new and evolving threats. Heuristic approaches, while capable of identifying some patterns, rely heavily on manually crafted rules that fail to adapt to emerging attack techniques. As a result, there is a pressing need for a more intelligent, scalable, and proactive approach to URL threat detection.

This project addresses the challenge of identifying and classifying malicious URLs in real-time using machine learning algorithms. The core problem is to develop a system that can accurately distinguish between legitimate and phishing URLs based on structural and behavioral features extracted from the URL itself—without relying on external databases or prior reports. These features may include URL length, presence of suspicious characters, protocol type (HTTP/HTTPS), domain age, and other metadata.

The goal is to design, train, and evaluate machine learning models—such as Random Forest and XGBoost—that can generalize well across various types of URLs and offer high detection accuracy with low false positives. Furthermore, the system must be accessible to non-technical users through a simple Graphical User Interface (GUI) that enables real-time URL safety checks.

In summary, the problem this research seeks to solve is the lack of an effective, user-friendly, and adaptive solution for detecting malicious URLs in a timely and accurate manner. The

project aims to fill this gap by leveraging machine learning to enhance web security and protect users from phishing attacks.

1.4 Applications

- **Web Browsers and Extensions:** Integrating the model into popular web browsers or as a browser extension can help users identify risky URLs before they open them. This can act as a first line of defense against phishing attacks, warning users in real-time when they attempt to access a suspicious website.
- **Email Filtering Systems:** Phishing attacks are commonly initiated through malicious links embedded in emails. The proposed model can be integrated into email servers or email clients to scan incoming messages and flag or quarantine emails containing suspicious URLs, thereby protecting users from clicking harmful links.
- **Enterprise Cybersecurity Solutions:** Organizations can incorporate the system into their internal cybersecurity infrastructure to monitor and block unsafe URLs accessed by employees. This is especially useful for companies that handle sensitive data and are frequently targeted by spear-phishing attacks.
- **Antivirus and Security Software:** The machine learning-based URL risk assessment can be embedded into antivirus software to provide an additional layer of protection. It can work alongside traditional virus signatures to evaluate URLs based on behavioral patterns, offering more comprehensive threat detection.
- **Cybersecurity Awareness Tools:** The system can also serve as an educational tool to raise awareness about phishing. By demonstrating how phishing URLs differ from legitimate ones, users can be trained to recognize and avoid common cyber threats.
- **Parental Control Tools:** Parents can use this system as part of a broader parental control suite to prevent children from visiting potentially harmful or inappropriate websites, ensuring a safer online experience for minors.
- **Government and Public Service Portals:** Public websites and digital service platforms used for filing documents, making payments, or accessing personal records can integrate the system to ensure that users are not redirected to fake portals designed to steal confidential information.

- **Educational Institutions:** Schools, colleges, and universities can deploy the system to safeguard students and staff from cyber threats, especially as many institutions now rely heavily on online learning platforms and digital communication tools.

2. Literature Survey

Phishing detection has been widely explored in the field of cybersecurity using various computational techniques. Ma et al. introduced one of the earliest machine learning-based systems for malicious URL classification, utilizing lexical and host-based features to train models such as Logistic Regression and Decision Trees [1]. Their findings indicated that machine learning models could outperform heuristic-based systems in detecting phishing URLs. Xiang et al. later expanded on this work by incorporating additional features like WHOIS data and achieved further improvements in detection performance [2].

In another study, Rao and Pais implemented Random Forest and Support Vector Machine (SVM) classifiers to detect phishing websites, demonstrating that ensemble learning approaches offered better resilience to noisy and imbalanced datasets [3]. Their model highlighted the effectiveness of combining multiple weak learners to boost overall accuracy. Similarly, Mohammad et al. employed a hybrid detection model that integrated blacklist verification with a decision tree classifier, resulting in enhanced accuracy and reduced false positives [4].

Recent research has explored deep learning techniques for URL classification. Bahnsen et al. proposed a deep neural network that could learn features directly from raw URLs, eliminating the need for manual feature engineering [5]. While their model achieved high accuracy, it required substantial computational resources and large datasets for training, making it less suitable for real-time applications. In contrast, Khanchandani and Singh presented a lightweight system using XGBoost for fast and accurate phishing detection, which could be deployed on user-end systems efficiently [6].

Despite these advancements, many existing approaches depend heavily on pre-collected blacklists or complex models that limit real-time applicability. Most models also require significant domain expertise for feature extraction, making them less adaptable to changing

attack vectors. The current research addresses these limitations by using interpretable and computationally efficient models like Random Forest and XGBoost, coupled with relevant URL features, to build a real-time risk assessment tool suitable for broad user adoption.

3. EXISTING SYSTEM

3.1 Random Forest

Random Forest is a widely used ensemble learning algorithm in machine learning, primarily applied to classification and regression problems. It is built using multiple decision trees and improves prediction accuracy by combining the outputs of individual trees. The algorithm follows the bagging (Bootstrap Aggregating) technique, where multiple decision trees are trained on random subsets of the data, and the final prediction is made based on majority voting (for classification) or averaging (for regression).

Random Forest is particularly effective in handling large datasets, reducing overfitting, and managing missing values. It also provides feature importance scores, which help in identifying the most significant attributes influencing the predictions. The following diagram illustrates how a Random Forest model is constructed and utilized:



Fig 3.1 : Random Forest Example

Example: Consider a dataset containing various features extracted from URLs, such as URL length, presence of special characters, use of HTTPS versus HTTP, number of subdomains, and domain age. Suppose the goal is to determine whether a given URL is malicious (indicative of phishing) or benign. A Random Forest classifier can be trained on this dataset, where each decision tree examines a random subset of features from different bootstrapped samples of the overall data. Each tree independently classifies a URL as either safe or malicious, and the final verdict is reached by majority voting across all trees. This ensemble approach not only improves the overall detection accuracy but also reduces the risk of

overfitting, ensuring robust performance even when dealing with noisy or diverse URL patterns.

3.2 Working of Random Forest

Step 1: Data Sampling

- The dataset is randomly divided into multiple subsets using bootstrapping.
- Each subset is used to train an individual decision tree.

Step 2: Decision Tree Construction

- At each node of a tree, a random subset of features is selected instead of considering all features, and the best split is chosen based on a criterion like Gini impurity or information gain.
- The tree is grown to its full depth without pruning, allowing diverse models to be generated..

Step 3: Aggregation of Predictions

- After training, each decision tree makes its own prediction for a given input sample.
- For regression tasks, the output is computed as the average of all tree predictions.
- This ensemble approach helps reduce variance and improve overall accuracy.

Figure 3.3: Decision Making in Random Forest

3.3 Advantages and Limitations

Advantages:

- **High Accuracy:** Aggregates predictions from multiple trees to enhance URL detection performance.
- **Handles Missing Data:** Effectively handles incomplete features by averaging outputs across trees.
- **Feature Importance:** Identifies key URL attributes such as length, HTTPS usage, and domain age.
- **Scalability:** Performs efficiently on large, high-dimensional datasets.

Limitations:

- **Computational Complexity:** Requires significant processing power, especially for real-time analysis.
- **Lack of Interpretability:** Operates as a "black-box," making individual decision processes hard to explain.
- **Overfitting Risk:** Using too many trees without proper tuning can lead to overfitting.

Figure 3.4: Overfitting in Random Forest

Thus, while Random Forest has proven effective in autism prediction, its limitations highlight the need for more advanced deep learning techniques such as XGBoost , which can improve accuracy, interpretability, and efficiency in Url detection.

4. PROPOSED SYSTEM

4.1 Overview

The proposed system introduces an intelligent, machine learning-based approach to evaluate URL safety in real time. By leveraging ensemble models like Random Forest (and optionally XGBoost), the system analyzes various URL features—including length, special characters, HTTPS usage, number of subdomains, and domain age—to classify URLs as either safe or malicious.

Data Collection and Preprocessing

The URLs are gathered from diverse sources and then cleaned and preprocessed. This step includes extracting relevant features and handling missing values to ensure the data is structured for model training.

Feature Extraction

Specific attributes from each URL—such as character counts, protocol type, domain registration details, and other structural metrics—are computed. These features are critical for distinguishing between benign and phishing URLs.

Model Training

The preprocessed dataset is divided into training and testing subsets. A Random Forest classifier is trained on this data using bagging, where multiple decision trees are built on bootstrapped samples. Each tree contributes its prediction, and the final output is determined by majority voting. This ensemble approach maximizes accuracy while reducing the risk of overfitting.

Model Evaluation and Tuning:

To ensure robust performance, the model is evaluated on standard metrics like accuracy, precision, recall, and F1-score. Hyperparameter tuning is performed to adjust parameters such as the number of trees, tree depth, and feature subsets to optimize performance further.

User Interface (UI):

A user-friendly graphical interface is developed to enable non-technical users to easily assess URL risk. Users simply enter a URL, and the system returns an instant prediction—indicating whether the URL is safe or potentially malicious.

Real-Time Monitoring and Updates

The system is designed to work in real time, scanning and classifying URLs as they are encountered.

The model can also be periodically retrained on updated datasets to adapt to new phishing techniques and emerging threats.

XGBOOST

XGBoost is an optimized implementation of Gradient Boosting, a powerful ensemble learning technique. Ensemble learning enhances predictive accuracy by combining multiple weak models to form a stronger and more reliable model.

XGBoost utilizes decision trees as its base learners, adding them sequentially to enhance performance. Each new tree focuses on correcting the errors made by the previous tree, a process known as boosting. One of its key strengths is built-in parallel processing, which allows XGBoost to train efficiently on large datasets. Additionally, it provides extensive customization options, enabling users to fine-tune model parameters to optimize performance for specific problems.

How XGBoost Works?

It builds decision trees sequentially with each tree attempting to correct the mistakes made by the previous one. The process can be broken down as follows:

1. **Start with a base learner:** The first model decision tree is trained on the data. In regression tasks this base model simply predict the average of the target variable.
3. **Calculate the errors:** After training the first tree the errors between the predicted and actual values are calculated.
5. **Train the next tree:** The next tree is trained on the errors of the previous tree. This step attempts to correct the errors made by the first tree.
7. **Repeat the process:** This process continues with each new tree trying to correct the errors

8. of the previous trees until a stopping criterion is met.

9. **Combine the predictions:** The final prediction is the sum of the predictions from all the trees.

Maths Behind XGBoost Algorithm

It can be viewed as iterative process where we start with an initial prediction often set to zero. After which each tree is added to reduce errors. Mathematically, the model can be represented as:

$$\hat{y}_i = \sum_{k=1}^K f_k(x_i)$$

Where \hat{y}_i is the final predicted value for the i th data point, K is the number of trees in the ensemble and $f_k(x_i)$ represents the prediction of the K th tree for the i th data point.

The objective function in XGBoost consists of two parts: a **loss function** and a **regularization term**. The loss function measures how well the model fits the data and the regularization term simplify complex trees. The general form of the loss function is:

$$\text{obj}(\theta) = \sum l(y_i, \hat{y}_i) + \sum_{k=1}^K \Omega(f_k)$$

Where:

- $l(y_i, \hat{y}_i)$ is the loss function which computes the difference between the true value y_i and the predicted value \hat{y}_i ,
- $\Omega(f_k)$ is the regularization term which discourages overly complex trees.

Now, instead of fitting the model all at once we optimize the model iteratively. We start with an initial prediction $\hat{y}_i(0) = 0$ and at each step we add a new tree to improve the model. The updated predictions after adding the t th tree can be written as:

$$\hat{y}_i(t) = \hat{y}_i(t-1) + f_t(x_i)$$

Where $\hat{y}_i(t-1)$ is the prediction from the previous iteration and $f_t(x_i)$ is the prediction of the t th tree for the i th data point.

The **regularization term** $\Omega(f_t)$ simplify complex trees by penalizing the number of leaves in the tree and the size of the leaf. It is defined as:

$$\Omega(f_t) = \gamma T + \frac{1}{2} \lambda \sum_{j=1}^T w_j^2$$

Where:

- T is the number of leaves in the tree
- γ is a regularization parameter that controls the complexity of the tree
- λ is a parameter that penalizes the squared weight of the leaves w_j

Finally when deciding how to split the nodes in the tree we compute the **information gain** for every possible split. The information gain for a split is calculated as:

$$\text{Gain} = \frac{1}{2} \left[\frac{GL^2 HL + \lambda + GR^2 HR + \lambda - (GL + GR)^2 HL + HR + \lambda}{HL + \lambda} - \frac{1}{2} \left(\frac{GL^2 HL + \lambda}{HL + \lambda} + \frac{GR^2 HR + \lambda}{HR + \lambda} \right) \right]$$

Where

- GL,GR are the sums of gradients in the left and right child nodes
- HL,HR are the sums of Hessians in the left and right child nodes

By calculating the information gain for every possible split at each node XGBoost selects the split that results in the largest gain which effectively reduces the errors and improves the model's performance.

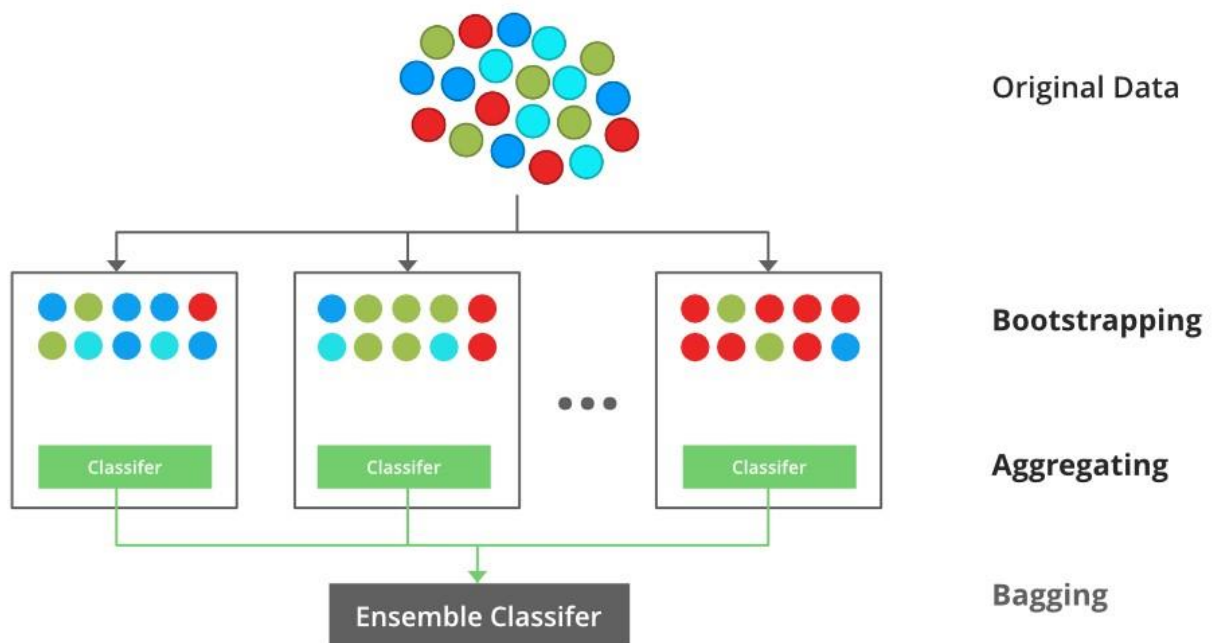


Fig 4.1: XGBoost Methodology

5. UML DIAGRAMS

UML stands for Unified Modeling Language. UML is a standardized general-purpose modeling language in the field of object-oriented software engineering. The standard is managed, and was created by, the Object Management Group. The goal is for UML to become a common language for creating models of object-oriented computer software. In its current form UML is comprised of two major components: a Meta-model and a notation. In the future, some form of method or process may also be added to; or associated with, UML.

The Unified Modeling Language is a standard language for specifying, Visualization, Constructing and documenting the artifacts of software system, as well as for business modeling and other non-software systems. The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems. The UML is a very important part of developing objects-oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects.

GOALS: The Primary goals in the design of the UML are as follows:

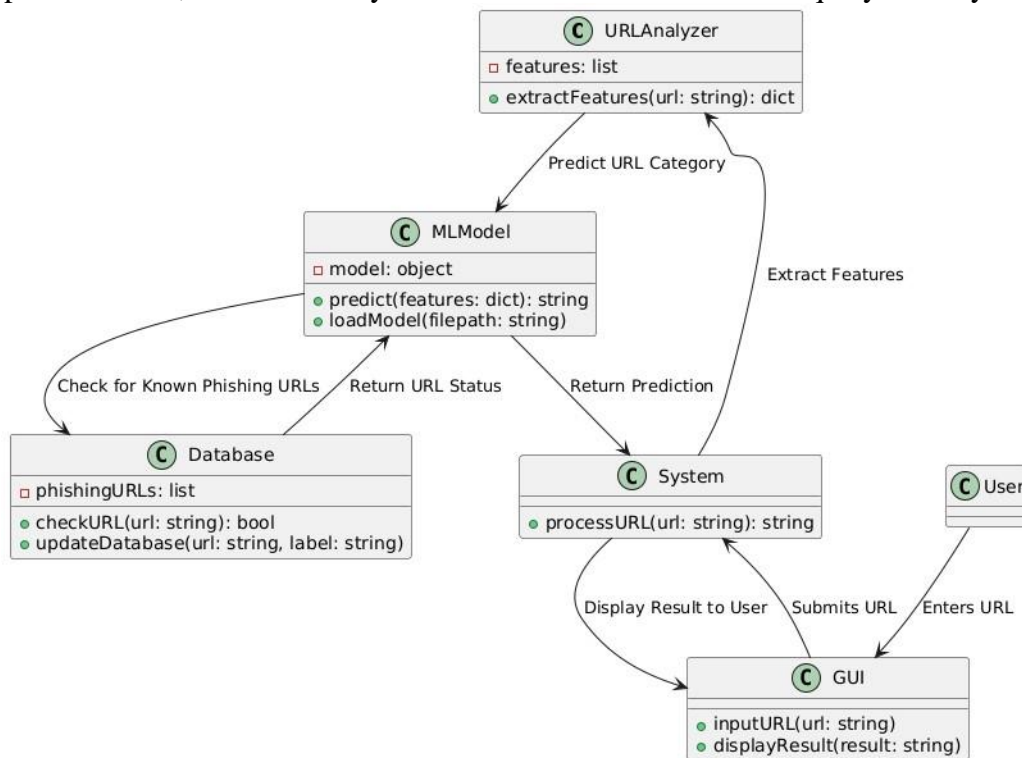
- Provide users a ready-to-use, expressive visual modeling Language so that they can
- develop and exchange meaningful models.
- Provide extendibility and specialization mechanisms to extend the core concepts.
- Be independent of particular programming languages and development process.
- Provide a formal basis for understanding the modeling language.
- Encourage the growth of OO tools market.
- Support higher level development concepts such as collaborations, frameworks,
- patterns and components.
- Integrate best practices.

5.1 Class diagram

The class diagram is used to refine the use case diagram and define a detailed design of the system. The class diagram classifies the actors defined in the use case diagram into a set of interrelated classes. The relationship or association between the classes can be either an "is-a" or "has-a" relationship. Each class in the class diagram may be capable of providing

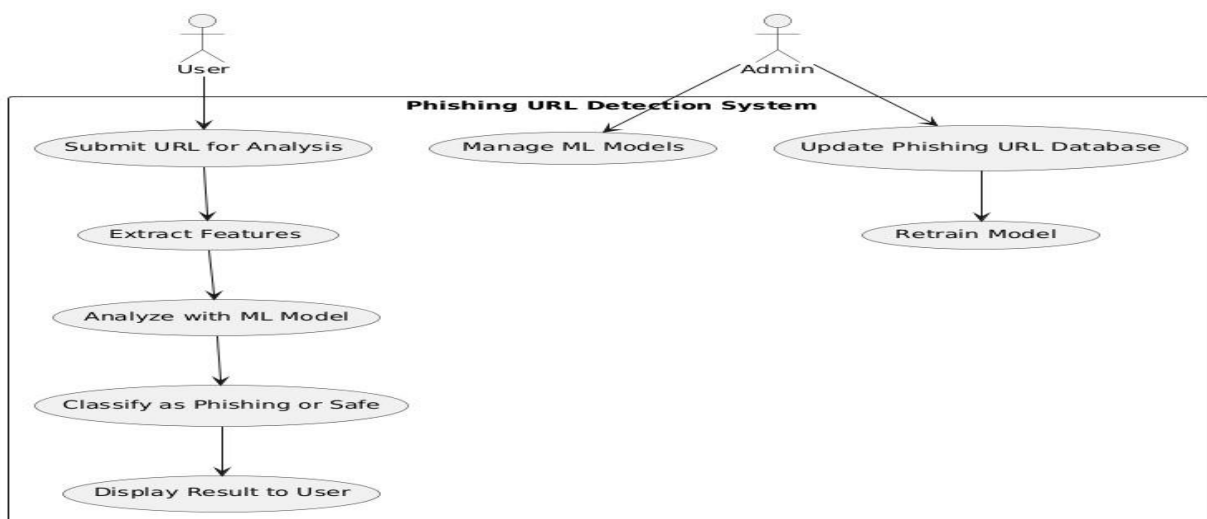
certain functionalities. These functionalities provided by the class are termed "methods" of the class.

Apart from this, each class may have certain "attributes" that uniquely identify the class.



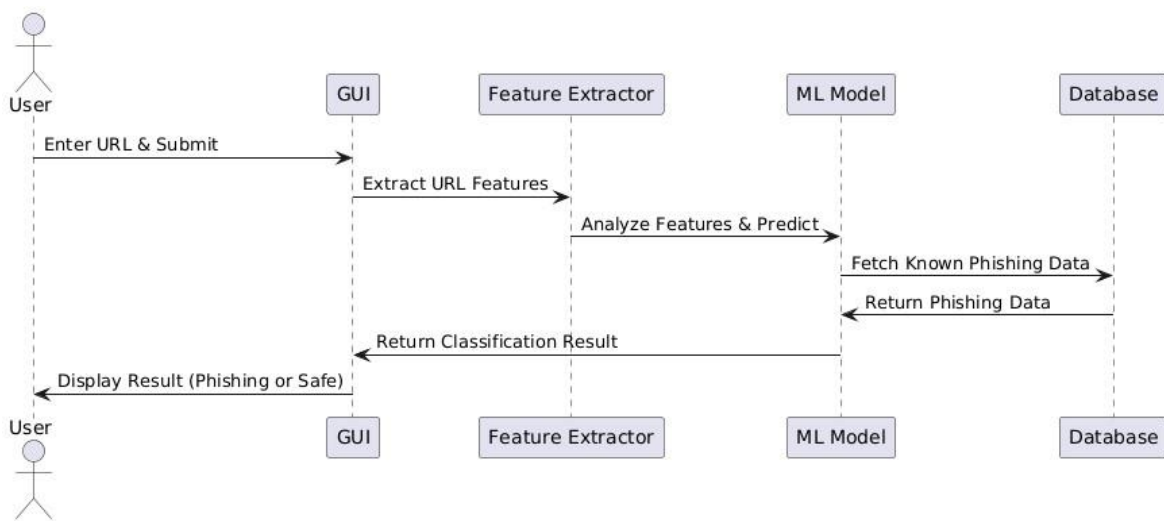
5.2 Use case Diagram

A use case diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted.



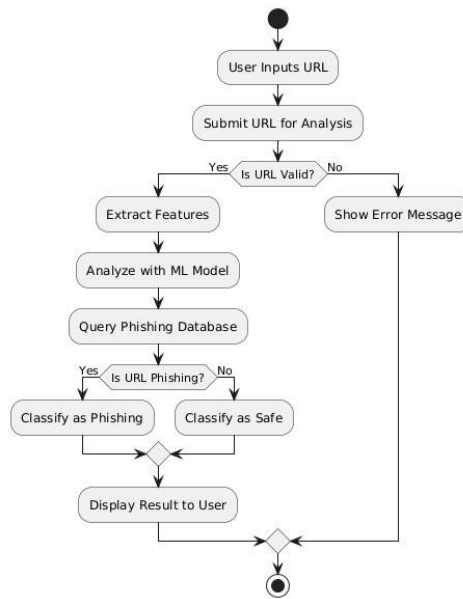
5.3 Sequence Diagram

A **sequence diagram** in Unified Modeling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. A sequence diagram shows, as parallel vertical lines ("lifelines"), different processes or objects that live simultaneously, and as horizontal arrows, the messages exchanged between them, in the order in which they occur. This allows the specification of simple runtime scenarios in a graphical manner.



5.4 Activity diagram: Activity diagrams are graphical representations of Workflows of stepwise activities and actions with support for choice, iteration, and concurrency.

In the Unified Modeling Language, activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system. An activity diagram



shows the overall flow of control.

6. SOFTWARE ENVIRONMENT

1. What is Python?

Below are some facts about Python.

Python is currently the most widely used multi-purpose, high-level programming language.

Python allows programming in Object-Oriented and Procedural paradigms. Python

programs generally are smaller than other programming languages like Java.

Programmers have to type relatively less and indentation requirement of the language,

makes them readable all the time.

Python language is being used by almost all tech-giant companies like – Google,

Amazon, Facebook, Instagram, Dropbox, Uber... etc.

The biggest strength of Python is huge collection of standard libraries which can be used for the following –

Machine Learning

GUI Applications (like Kivy, Tkinter, PyQt etc.)

Web frameworks like Django (used by YouTube, Instagram, Dropbox)

Image processing (like Opencv, Pillow)

Web scraping (like Scrapy, BeautifulSoup, Selenium)

Test frameworks

Multimedia

Advantages of Python

Let's see how Python dominates over other languages.

1. Extensive Libraries

Python downloads with an extensive library and it contains code for various purposes like regular expressions, documentation-generation, unit-testing, web browsers, threading, databases, CGI, email, image manipulation, and more. So, we don't have to write the complete code for that manually.

2. Extensible

As we have seen earlier, Python can be extended to other languages. You can write some of your code in languages like C++ or C. This comes in handy, especially in projects.

3. Embeddable

Complimentary to extensibility, Python is embeddable as well. You can put your Python code in your source code of a different language, like C++. This lets us add scripting capabilities to our code in the other language.

4. Improved Productivity

The language's simplicity and extensive libraries render programmers more productive than languages like Java and C++ do. Also, the fact that you need to write less and get more things done.

5. IOT Opportunities

Since Python forms the basis of new platforms like Raspberry Pi, it finds the future bright for the Internet of Things. This is a way to connect the language with the real world.

6. Simple and Easy

When working with Java, you may have to create a class to print 'Hello World'. But in Python, just a print statement will do. It is also quite easy to learn, understand, and code. This is why when people pick up Python, they have a hard time adjusting to other more verbose languages like Java.

7. Readable

Because it is not such a verbose language, reading Python is much like reading English. This is the reason why it is so easy to learn, understand, and code. It also does not need curly braces to define blocks, and indentation is mandatory. These further aids the readability of the code.

8. Object-Oriented

This language supports both the procedural and object-oriented programming paradigms.

While functions help us with code reusability, classes and objects let us model the real world.

A class allows the encapsulation of data and functions into one.

9. Free and Open-Source

Like we said earlier, Python is freely available. But not only can you download Python for free, but you can also download its source code, make changes to it, and even distribute it. It downloads with an extensive collection of libraries to help you with your tasks.

10. Portable

When you code your project in a language like C++, you may need to make some changes to it if you want to run it on another platform. But it isn't the same with Python. Here, you need to code only once, and you can run it anywhere. This is called Write Once Run Anywhere (WORA). However, you need to be careful enough not to include any system-dependent features.

11. Interpreted

Lastly, we will say that it is an interpreted language. Since statements are executed one by one,

debugging is easier than in compiled languages.

Any doubts till now in the advantages of Python? Mention in the comment section.

Advantages of Python Over Other Languages

1. Less Coding

Almost all of the tasks done in Python requires less coding when the same task is done in other languages. Python also has an awesome standard library support, so you don't have to search for any third-party libraries to get your job done. This is the reason that many people suggest learning Python to beginners.

2. Affordable

Python is free therefore individuals, small companies or big organizations can leverage the free available resources to build applications. Python is popular and widely used so it gives you better community support.

The 2019 Github annual survey showed us that Python has overtaken Java in the most popular programming language category.

3. Python is for Everyone

Python code can run on any machine whether it is Linux, Mac or Windows. Programmers need to learn different languages for different jobs but with Python, you can professionally build web apps, perform data analysis and machine learning, automate things, do web scraping and also build games and powerful visualizations. It is an all-rounder programming language.

Disadvantages of Python

So far, we've seen why Python is a great choice for your project. But if you choose it, you should be aware of its consequences as well. Let's now see the downsides of choosing Python over another language.

1. Speed Limitations

We have seen that Python code is executed line by line. But since Python is interpreted, it often results in slow execution. This, however, isn't a problem unless speed is a focal point for the project. In other words, unless high speed is a requirement, the benefits offered by Python are enough to distract us from its speed limitations.

2. Weak in Mobile Computing and Browsers

While it serves as an excellent server-side language, Python is much rarely seen on the client-side. Besides that, it is rarely ever used to implement smartphone-based applications. One such application is called Carbonnelle.

The reason it is not so famous despite the existence of Brython is that it isn't that secure.

3. Design Restrictions

As you know, Python is dynamically-typed. This means that you don't need to declare the type of variable while writing the code. It uses duck-typing. But wait, what's that? Well, it just means that if it looks like a duck, it must be a duck. While this is easy on the programmers during coding, it can raise run-time errors.

4. Underdeveloped Database Access Layers

Compared to more widely used technologies like JDBC (Java DataBase Connectivity) and ODBC (Open DataBase Connectivity), Python's database access layers are a bit underdeveloped. Consequently, it is less often applied in huge enterprises.

5. Simple

No, we're not kidding. Python's simplicity can indeed be a problem. Take my example. I don't do Java, I'm more of a Python person. To me, its syntax is so simple that the verbosity of Java code seems unnecessary.

This was all about the Advantages and Disadvantages of Python Programming Language.

2. History of Python

What do the alphabet and the programming language Python have in common? Right, both start with ABC. If we are talking about ABC in the Python context, it's clear that the programming language ABC is meant. ABC is a general-purpose programming language and programming environment, which had been developed in the Netherlands, Amsterdam, at the CWI (Centrum Wiskunde & Informatica). The greatest achievement of ABC was to influence the design of Python. Python was conceptualized in the late 1980s. Guido van Rossum worked that time in a project at the CWI, called Amoeba, a distributed operating system. In an interview with Bill Venners¹, Guido van Rossum said: "In the early 1980s, I worked as an implementer on a team building a language called ABC at Centrum voor Wiskunde en Informatica (CWI). I don't know how well people know ABC's influence on

Python. I try to mention ABC's influence because I'm indebted to everything I learned during that project and to the people who worked on it. "Later on in the same Interview, Guido van Rossum continued: "I remembered all my experience and some of my frustration with ABC. I decided to try to design a simple scripting language that possessed some of ABC's better properties, but without its problems. So, I started typing. I created a simple virtual machine, a simple parser, and a simple runtime. I made my own version of the various ABC parts that I liked. I created a basic syntax, used indentation for statement grouping instead of curly braces or begin-end blocks, and developed a small number of powerful data types: a hash table (or dictionary, as we call it), a list, strings, and numbers."

Python Development Steps

Guido Van Rossum published the first version of Python code (version 0.9.0) at alt.sources in February 1991. This release included already exception handling, functions, and the core data types of lists, dict, str and others. It was also object oriented and had a module system. Python version 1.0 was released in January 1994. The major new features included in this release were the functional programming tools lambda, map, filter and reduce, which Guido Van Rossum never liked. Six and a half years later in October 2000, Python 2.0 was introduced. This release

included list comprehensions, a full garbage collector and it was supporting unicode. Python flourished for another 8 years in the versions 2.x before the next major release as Python 3.0 (also known as "Python 3000" and "Py3K") was released. Python 3 is not backwards compatible with Python 2.x. The emphasis in Python 3 had been on the removal of duplicate programming constructs and modules, thus fulfilling or coming close to fulfilling the 13th law of the Zen of Python: "There should be one -- and preferably only one -- obvious way to do it."Some changes in Python 7.3:

Print is now a function.

Views and iterators instead of lists

The rules for ordering comparisons have been simplified. E.g., a heterogeneous list cannot be sorted, because all the elements of a list must be comparable to each other.

There is only one integer type left, i.e., int. long is int as well.

The division of two integers returns a float instead of an integer. "//" can be used to have the "old" behaviour.

Text Vs. Data Instead of Unicode Vs. 8-bit

Purpose

We demonstrated that our approach enables successful segmentation of intra-retinal layers—even with low-quality images containing speckle noise, low contrast, and different intensity ranges throughout—with the assistance of the ANIS feature.

Python

Python is an interpreted high-level programming language for general-purpose programming. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace.

Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library.

Python is Interpreted – Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.

Python is Interactive – you can actually sit at a Python prompt and interact with the interpreter directly to write your programs.

Python also acknowledges that speed of development is important. Readable and terse code is part of this, and so is access to powerful constructs that avoid tedious repetition of code. Maintainability also ties into this may be an all but useless metric, but it does say something about how much code you have to scan, read and/or understand to troubleshoot problems or tweak behaviors. This speed of development, the ease with which a programmer of other languages can pick up basic Python skills and the huge standard library is key to another area where Python excels. All its tools have been quick to implement, saved a lot of time, and several of them have later been patched and updated by people with no Python background - without breaking.

3. Modules Used in Project

TensorFlow

TensorFlow is a free and open-source software library for dataflow and differentiable programming across a range of tasks. It is a symbolic math library and is also used for

machine learning applications such as neural networks. It is used for both research and production at Google.

TensorFlow was developed by the Google Brain team for internal Google use. It was released under the Apache 2.0 open-source license on November 9, 2015.

NumPy

NumPy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays.

It is the fundamental package for scientific computing with Python. It contains various features including these important ones:

- A powerful N-dimensional array object

- Sophisticated (broadcasting) functions

- Tools for integrating C/C++ and Fortran code

- Useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary datatypes can be defined using NumPy which allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

Pandas

Pandas is an open-source Python Library providing high-performance data manipulation and analysis tool using its powerful data structures. Python was majorly used for data munging and preparation. It had very little contribution towards data analysis. Pandas solved this problem. Using Pandas, we can accomplish five typical steps in the processing and analysis of data, regardless of the origin of data load, prepare, manipulate, model, and analyze. Python with Pandas is used in a wide range of fields including academic and commercial domains including finance, economics, Statistics, analytics, etc.

Matplotlib

Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can

be used in Python scripts, the Python and IPython shells, the Jupyter Notebook, web application servers, and four graphical user interface toolkits. Matplotlib tries to make easy things easy and hard things possible. You can generate plots, histograms, power spectra, bar charts, error charts, scatter plots, etc., with just a few lines of code. For examples, see the sample plots and thumbnail gallery.

For simple plotting the pyplot module provides a MATLAB-like interface, particularly when combined with IPython. For the power user, you have full control of line styles, font properties, axes properties, etc, via an object-oriented interface or via a set of functions familiar to MATLAB users.

Scikit – learn

Scikit-learn provides a range of supervised and unsupervised learning algorithms via a consistent interface in Python. It is licensed under a permissive simplified BSD license and is distributed under many Linux distributions, encouraging academic and commercial use.

Python

Python is an interpreted high-level programming language for general-purpose programming. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace.

Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library.

Python is Interpreted – Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.

Python is Interactive – you can actually sit at a Python prompt and interact with the interpreter directly to write your programs.

Python also acknowledges that speed of development is important. Readable and terse code is part of this, and so is access to powerful constructs that avoid tedious repetition of code. Maintainability also ties into this may be an all but useless metric, but it does say something about how much code you have to scan, read and/or understand to troubleshoot problems or tweak behaviors. This speed of development, the ease with which a programmer of other languages can pick up basic Python skills and the huge standard library is key to another area where Python excels. All its tools have been quick to implement, saved a lot of time, and

several of them have later been patched and updated by people with no Python background - without breaking.

4. Install Python Step-by-Step in Windows and Mac

Python a versatile programming language doesn't come pre-installed on your computer devices. Python was first released in the year 1991 and until today it is a very popular high-level programming language. Its style philosophy emphasizes code readability with its notable use of great whitespace.

The object-oriented approach and language construct provided by Python enables programmers to write both clear and logical code for projects. This software does not come pre-packaged with Windows.

How to Install Python on Windows and Mac

There have been several updates in the Python version over the years. The question is how to install Python? It might be confusing for the beginner who is willing to start learning Python but this tutorial will solve your query. The latest or the newest version of Python is version

3.7.4 or in other words, it is Python 3.

Note: The python version 3.7.4 cannot be used on Windows XP or earlier devices. Before you start with the installation process of Python. First, you need to know about your System Requirements. Based on your system type i.e., operating system and based processor, you must download the python version. My system type is a Windows 64-bit operating system. So, the steps below are to install python version 3.7.4 on Windows 7 device or to install Python 3. Download the Python Cheatsheet [here](#). The steps on how to install Python on Windows 10, 8 and 7 are divided into 4 parts to help understand better.

Download the Correct version into the system

Step 1: Go to the official site to download and install python using Google Chrome or any other web browser. OR Click on the following link: <https://www.python.org>










Now, check for the latest and the correct version for your operating system.

Step 2: Click on the Download Tab.



Step 3: You can either select the Download Python for windows 3.7.4 button in Yellow Color or you can scroll further down and click on download with respective to their version. Here, we are downloading the most recent python version for windows 3.7.4

Looking for a specific release?			
Python releases by version number:			
Release version	Release date		Click for more
Python 3.7.4	July 8, 2019	 Download	Release Notes
Python 3.6.9	July 2, 2019	 Download	Release Notes
Python 3.7.3	March 25, 2019	 Download	Release Notes
Python 3.4.10	March 18, 2019	 Download	Release Notes
Python 3.5.7	March 18, 2019	 Download	Release Notes
Python 2.7.16	March 4, 2019	 Download	Release Notes
Python 3.7.2	Dec. 24, 2018	 Download	Release Notes

Step 4: Scroll down the page until you find the Files option.

Step 5: Here you see a different version of python along with the operating system.

Files					
Version	Operating System	Description	MD5 Sum	File Size	GPG
Unipped source tarball	Source release		68111671e5b2db4aef709ab011079be	23017663	3xG
3.7 compressed source tarball	Source release		d33e4aee66097051c1eca45ee3604803	17131432	3xG
macOS 64-bit/32-bit installer	Mac OS X	for Mac OS X 10.6 and later	6428b4fa7583daf1a442bafce08e6	34898416	3xG
macOS 64-bit installer	Mac OS X	for OS X 10.9 and later	5dd605c38217a45773b5e4a936b2A1f	28882845	3xG
Windows help file	Windows		d83990573ab26b2ac58acade0b4f7cd2	8131761	3xG
Windows x86-64 embeddable zip file	Windows	for AMD64/EM64/x64	9809c3b7d8ee0b9a6e32184a4728a2	7504381	3xG
Windows x86-64 executable installer	Windows	for AMD64/EM64/x64	a702be4b0a070d4b0b30c3a1d3e563e00	2688348	3xG
Windows x86-64 web-based installer	Windows	for AMD64/EM64/x64	28c31c90ff0b073aee9e51a3bf35184bd2	1362904	3xG
Windows x86 embeddable zip file	Windows		9fab3bf198a41d79fda64112574139d0	6741628	3xG
Windows x86 executable installer	Windows		33c2802942a54446a3db451476394789	25661848	3xG
Windows x86 web-based installer	Windows		1b670cf6d3117d82c30983ea371d07c	1324608	3xG

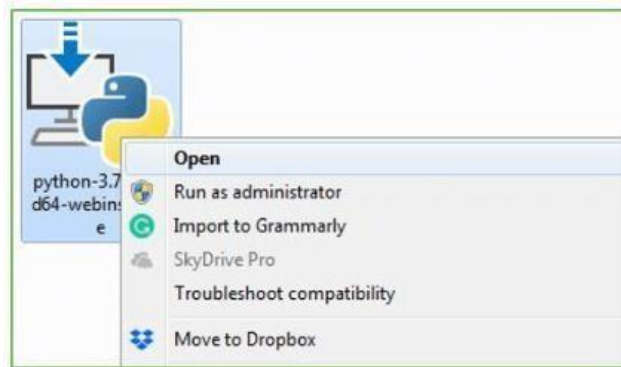
- To download Windows 32-bit python, you can select any one from the three options: Windows x86 embeddable zip file, Windows x86 executable installer or Windows x86 web-based installer.
- To download Windows 64-bit python, you can select any one from the three options: Windows x86-64 embeddable zip file, Windows x86-64 executable installer or Windows x86-64 web-based installer.

Here we will install Windows x86-64 web-based installer. Here your first part regarding which version of python is to be downloaded is completed. Now we move ahead with the second part in installing python i.e., Installation

Note: To know the changes or updates that are made in the version you can click on the Release Note Option.

Installation of Python

Step 1: Go to Download and Open the downloaded python version to carry out the installation process.



Step 2: Before you click on Install Now, make sure to put a tick on Add Python 3.7 to PATH.



Step 3: Click on Install NOW After the installation is successful. Click on Close.



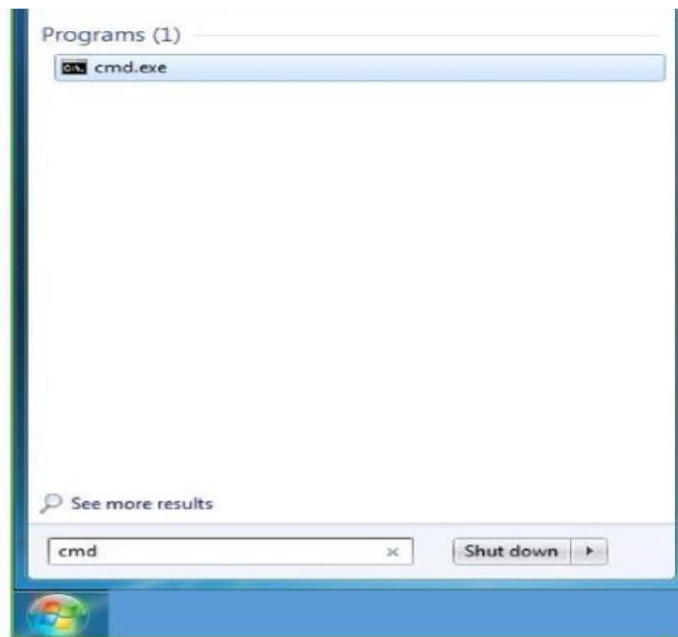
With these above three steps on python installation, you have successfully and correctly installed Python. Now is the time to verify the installation.

Note: The installation process might take a couple of minutes.

Verify the Python Installation

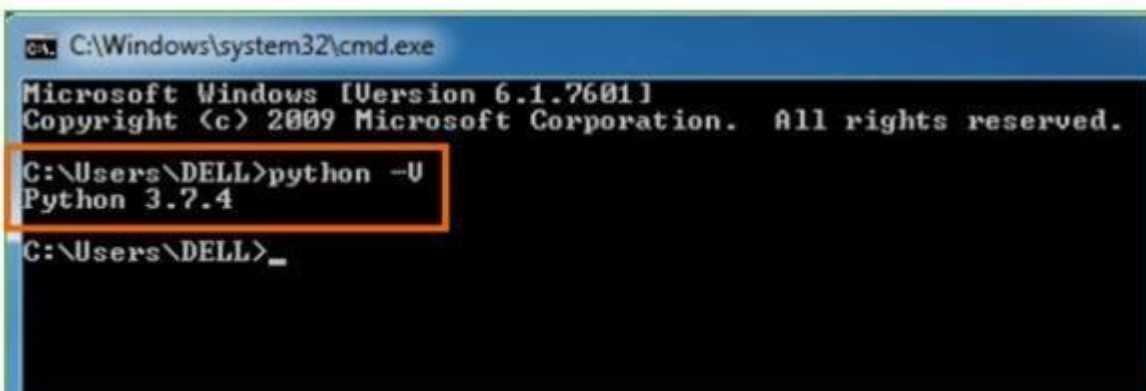
Step 1: Click on Start

Step 2: In the Windows Run Command, type “cmd”.



Step 3: Open the Command prompt option.

Step 4: Let us test whether the python is correctly installed. Type python -V and press Enter.



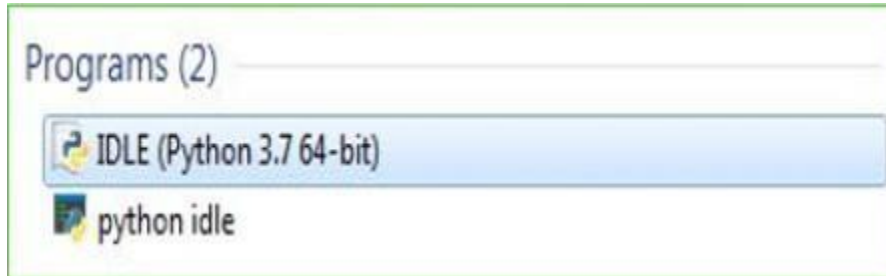
Step 5: You will get the answer as 3.7.4

Note: If you have any of the earlier versions of Python already installed. You must first uninstall the earlier version and then install the new one.

Check how the Python IDLE works

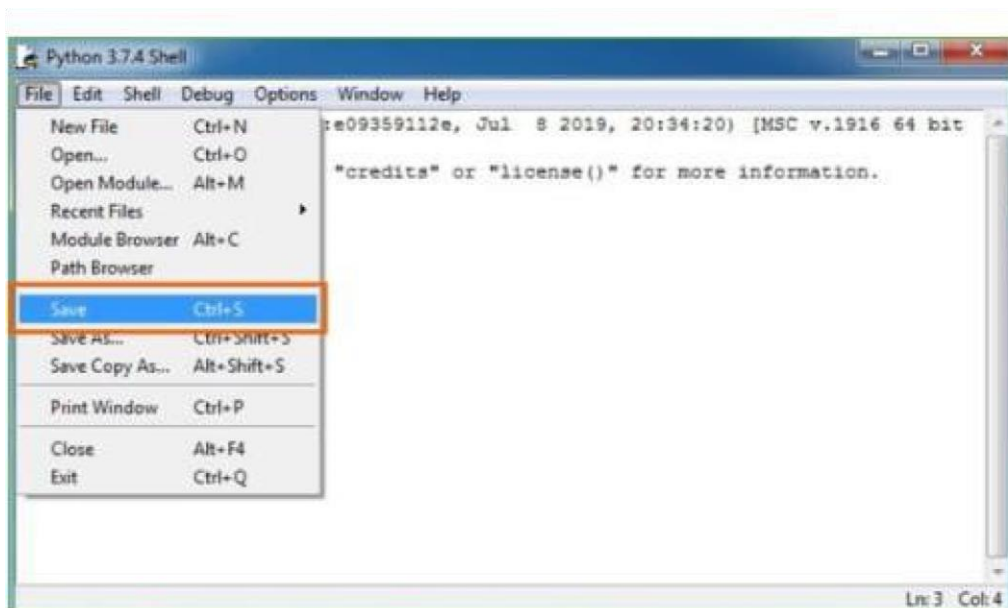
Step 1: Click on Start

Step 2: In the Windows Run command, type “python idle”.



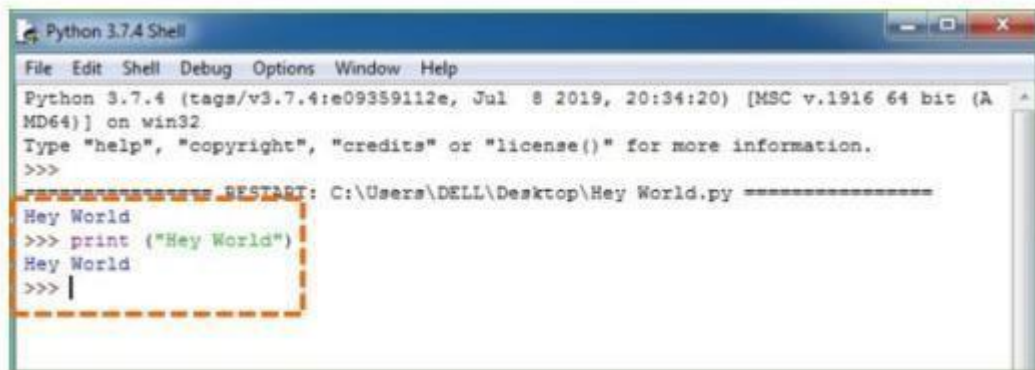
Step 3: Click on IDLE (Python 3.7 64-bit) and launch the program

Step 4: To go ahead with working in IDLE you must first save the file. Click on File > Click on Save



Step 5: Name the file and save as type should be Python files. Click on SAVE. Here I have named the files as Hey World.

Step 6: Now for e.g., enter print (“Hey World”) and Press Enter.

A screenshot of a Windows command prompt window titled "Python 3.7.4 Shell". The window has a menu bar with "File", "Edit", "Shell", "Debug", "Options", "Window", and "Help". The text inside the window shows the Python version and build information: "Python 3.7.4 (tags/v3.7.4:09359112e, Jul 8 2019, 20:34:20) [MSC v.1916 64 bit (AMD64)] on win32". It also displays the prompt "Type 'help', 'copyright', 'credits' or 'license()' for more information." followed by ">>>". Below this, a file path "C:\Users\DELL\Desktop\Hey World.py" is shown, followed by a separator line of dashes. The output "Hey World" is displayed. Then, the command ">>> print ('Hey World')" is entered, followed by another "Hey World" output. Finally, the prompt ">>> |" is shown. A dashed orange box highlights the command and its output.

```
Python 3.7.4 Shell
File Edit Shell Debug Options Window Help
Python 3.7.4 (tags/v3.7.4:09359112e, Jul 8 2019, 20:34:20) [MSC v.1916 64 bit (A
MD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:\Users\DELL\Desktop\Hey World.py =====
Hey World
>>> print ('Hey World')
Hey World
>>> |
```

You will see that the command given is launched. With this, we end our tutorial on how to install Python. You have learned how to download python for windows into your respective operating system.

Note: Unlike Java, Python does not need semicolons at the end of the statements otherwise it won't work.

7. SYSTEM REQUIREMENTS SPECIFICATIONS

1. Software Requirements

The functional requirements or the overall description documents include the product perspective and features, operating system and operating environment, graphics requirements, design constraints and user documentation.

The appropriation of requirements and implementation constraints gives the general overview of the project in regard to what the areas of strength and deficit are and how to tackle them.

Python IDLE 3.7 version (or)

Anaconda 3.7 (or)

Jupyter (or)

Google colab

Libraries and Frameworks : Scikit-learn, Pandas, Numpy, Matplotlib , Scikit – learn , XGBoost , Imblearn, Tkinter or PyQt.

2. Hardware Requirements

Minimum hardware requirements are very dependent on the particular software being developed by a given Enthought Python / Canopy / VS Code user. Applications that need to store large arrays/objects in memory will require more RAM, whereas applications that need to perform numerous calculations or tasks more quickly will require a faster processor.

Operating system	:	Windows, Linux
Processor	:	minimum intel i3
Ram	:	minimum 4 GB
Hard disk	:	minimum 250GB

8. FUNCTIONAL REQUIREMENTS

1. Output Design

Outputs from computer systems are required primarily to communicate the results of processing to users. They are also used to provides a permanent copy of the results for later consultation. The various types of outputs in general are:

External Outputs, whose destination is outside the organization

Internal Outputs whose destination is within organization and they are the

User's main interface with the computer.

Operational outputs whose use is purely within the computer department.

Interface outputs, which involve the user in communicating directly.

Output Definition

The outputs should be defined in terms of the following points:

Type of the output

Content of the output

Format of the output

Location of the output

Frequency of the output

Volume of the output

Sequence of the output

It is not always desirable to print or display data as it is held on a computer. It should be decided as which form of the output is the most suitable.

2. Input Design

Input design is a part of overall system design. The main objective during the input design is as given below:

To produce a cost-effective method of input.

To achieve the highest possible level of accuracy.

To ensure that the input is acceptable and understood by the user.

Input Stages

The main input stages can be listed as below:

Data recording

Data transcription

Data conversion

Data verification

Data control

Data transmission

Data validation

Data correction

Input Types

It is necessary to determine the various types of inputs. Inputs can be categorized as follows:

External inputs, which are prime inputs for the system.

Internal inputs, which are user communications with the system.

Operational, which are computer department's communications to the system?

Interactive, which are inputs entered during a dialogue.

Input Media

At this stage choice has to be made about the input media. To conclude about the input media

consideration has to be given to;

Type of input

Flexibility of format

Speed

Accuracy

Verification methods

Rejection rates

Ease of correction

Storage and handling requirements

Security

Easy to use

Portability

Keeping in view the above description of the input types and input media, it can be said that most of the inputs are of the form of internal and interactive. As

Input data is to be the directly keyed in by the user, the keyboard can be considered to be the most suitable input device.

Error Avoidance

At this stage care is to be taken to ensure that input data remains accurate from the stage at which it is recorded up to the stage in which the data is accepted by the system. This can be achieved only by means of careful control each time the data is handled.

Error Detection

Even though every effort is made to avoid the occurrence of errors, still a small proportion of errors is always likely to occur, these types of errors can be discovered by using validations to check the input data.

Data Validation

Procedures are designed to detect errors in data at a lower level of detail. Data validations have been included in the system in almost every area where there is a possibility for the user to commit errors. The system will not accept invalid data. Whenever an invalid data is keyed in, the system immediately prompts the user and the user has to again key in the data and the system will accept the data only if the data is correct. Validations have been included where necessary.

The system is designed to be a user friendly one. In other words the system has been designed to communicate effectively with the user. The system has been designed with popup menus.

3. User Interface Design

It is essential to consult the system users and discuss their needs while designing the user interface:

User Interface Systems Can Be Broadly Clasified As:

User initiated interface the user is in charge, controlling the progress of the user/computer dialogue. In the computer-initiated interface, the computer selects the next stage in the interaction.

Computer initiated interfaces

In the computer-initiated interfaces the computer guides the progress of the user/computer dialogue. Information is displayed and the user response of the computer takes action or displays further information.

User Initiated Intergfaces

User initiated interfaces fall into two approximate classes:

Command driven interfaces: In this type of interface the user inputs commands or queries which are interpreted by the computer.

Forms oriented interface: The user calls up an image of the form to his/her screen and fills in the form. The forms-oriented interface is chosen because it is the best choice.

Computer-Initiated Interfaces

The following computer – initiated interfaces were used:

The menu system for the user is presented with a list of alternatives and the user chooses one; of alternatives.

Questions – answer type dialog system where the computer asks question and takes action based on the basis of the users reply.

Right from the start the system is going to be menu driven, the opening menu displays the available options. Choosing one option gives another popup menu with more options. In this way every option leads the users to data entry form where the user can key in the data.

Error Message Design

The design of error messages is an important part of the user interface design. As user is bound to commit some errors or other while designing a system the system should be designed to be helpful by providing the user with information regarding the error he/she has committed.

This application must be able to produce output at different modules for different inputs.

4. Performance Requirements

Performance is measured in terms of the output provided by the application. Requirement specification plays an important part in the analysis of a system. Only when the requirement specifications are properly given, it is possible to design a system, which will fit into required environment. It rests largely in the part of the users of the existing system to give the requirement specifications because they are the people who finally use the system. This is because the requirements have to be known during the initial stages so that the system can be designed according to those requirements. It is very difficult to change the system once it has been designed and on the other hand designing a system, which does not cater to

the requirements of the user, is of no use.

The requirement specification for any system can be broadly stated as given below:

The system should be able to interface with the existing system

The system should be accurate

The system should be better than the existing system

The existing system is completely dependent on the user to perform all the duties.

9. SOURCE CODE

A dataset containing behavioral and social interaction data is required to train and evaluate the autism prediction model. The dataset was obtained from an online source and consists of various attributes relevant to autism diagnosis.

For this study, the dataset includes demographic details, behavioral screening responses, and social interaction patterns of individuals. The dataset consists of records of individuals who underwent autism screening based on predefined behavioral traits and responses.

The dataset comprises various categorical and numerical features, including age, gender, ethnicity, country of residence, and answers to autism screening questions. Additionally, it includes a Class/ASD label that indicates whether an individual is diagnosed with Autism Spectrum Disorder (ASD).

To ensure the data is suitable for machine learning, preprocessing steps such as handling missing values, encoding categorical variables, and balancing class distribution using oversampling techniques were applied. The dataset serves as the foundation for training and evaluating the XGBoost-based predictive model, which helps in identifying key patterns for autism prediction.

TRANING CODE:

```
import tensorflow as tf
tf.test.gpu_device_name()

import pandas as pd

# Loading the downloaded dataset
df = pd.read_csv("/content/drive/MyDrive/urldata.csv")
df.head(10)

df = df.drop('Unnamed: 0',axis=1)
df.head(10)

df.info()

df.shape

df["label"].value_counts()

#Importing dependencies
from urllib.parse import urlparse
import os.path

# changing dataframe variable
urldata = df
```

```
#Length of URL (Phishers can use long URL to hide the doubtful part in the address bar)
urldata['url_length'] = urldata['url'].apply(lambda i: len(str(i)))
```

```
#Hostname Length
```

```
urldata['hostname_length'] = urldata['url'].apply(lambda i: len(urlparse(i).netloc))
```

```
#Path Length
```

```
urldata['path_length'] = urldata['url'].apply(lambda i: len(urlparse(i).path))
```

```
#First Directory Length
```

```
def fd_length(url):
```

```
    urlpath= urlparse(url).path
```

```
    try:
```

```
        return len(urlpath.split('/')[1])
```

```
    except:
```

```
        return 0
```

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

```
from flask import Flask, request, jsonify, send_from_directory
```

```
from flask_cors import CORS
```

```
from API import get_prediction
```

```
urldata.head(10)
```

```
urldata['count-'] = urldata['url'].apply(lambda i: i.count('-'))
```

```
urldata['count@'] = urldata['url'].apply(lambda i: i.count('@'))
```

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

```
urldata['count%'] = urldata['url'].apply(lambda i: i.count('%'))
```

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

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

```
urldata['count-http'] = urldata['url'].apply(lambda i : i.count('http'))
```

```
urldata['count-https'] = urldata['url'].apply(lambda i : i.count('https'))
```

```
urldata['count-www'] = urldata['url'].apply(lambda i: i.count('www'))
```

```
def digit_count(url):
```

```
    digits = 0
```

```
    for i in url:
```

```
        if i.isnumeric():
```

```
            digits = digits + 1
```

```
    return digits
```

```
urldata['count-digits']= urldata['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
```

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

```
def no_of_dir(url):
```

```
    urldir = urlparse(url).path
```

```
    return urldir.count('/')
```

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

```
urldata.head(10)
```

```
def shortening_service(url):
```

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

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

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

```
        'doiop\.com|short\.ie|kl\.am|wp\.me|rubyurl\.com|om\.ly|to\.ly|bit\.do|t\.co|lnkd\.in|'
```

```
        'db\.tt|qr\.ae|adf\.ly|goo\.gl|bitly\.com|cur\.lv|tinyurl\.com|ow\.ly|bit\.ly|ity\.im|'
```

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

```

        'x\.co|prettylinkpro\.com|scrnch\.me|filoops\.info|vzturl\.com|qr\.net|1url\.com|tweez\.
me|v\.gd|'
        'tr\.im|link\.zip\.net',
        url)
    if match:
        return -1
    else:
        return 1
urldata['short_url'] = urldata['url'].apply(lambda i: shortening_service(i))
urldata.to_csv("Url_Processed.csv")

```

API CODE:

```

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
        '[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}))\\|)'
    in hexadecimal
        '(:[a-fA-F0-9]{1,4}:){7}[a-fA-F0-9]{1,4}', url) # Ipv6
    if match:
        # print match.group()
        return -1
    else:
        # print 'No matching pattern found'
        return 1
urldata['use_of_ip'] = urldata['url'].apply(lambda i: having_ip_address(i))
import os
from ui_analysis import analyze_url_for_display, generate_risk_factors, calculate_risk_components

```



```

app = Flask(__name__)
CORS(app) # This enables CORS for all routes

# Path to your trained model - use absolute path
MODEL_PATH = os.path.join(os.path.dirname(os.path.abspath(__file__)),
                           "models", "Malicious_URL_Prediction.h5")

# Add this route to serve static images
@app.route('/images/<path:filename>')
def serve_image(filename):
    return send_from_directory('static/images', filename)

@app.route('/api/check-url', methods=['POST'])
def check_url():
    try:
        data = request.get_json()
        if not data or 'url' not in data:
            return jsonify({"error": "Please provide a URL"}), 400

        url = data['url']

        # Get prediction from your existing function
        prediction_score = get_prediction(url, MODEL_PATH)

        # Determine if it's phishing based on threshold
        is_phishing = prediction_score > 50

        # Extract domain from URL for display
        domain = url
        if '://' in url:
            domain = url.split('://', 1)[1]
        if '/' in domain:

```

```
domain = domain.split('/', 1)[0]

# Get additional display-only features
display_features = analyze_url_for_display(url)

# Generate risk factors
risk_factors = generate_risk_factors(url, display_features, is_phishing)

# Calculate risk components
risk_breakdown = calculate_risk_components(display_features)
risk_breakdown["overall_risk"] = round(prediction_score / 10, 1)

# Base URL for images
base_url = request.url_root

# Format response in the structure expected by Results.jsx
response = {
    "scan": {
        "url": url,
        "domain": domain,
        "is_phishing": bool(is_phishing),
        "confidence": float(prediction_score),
        "time": "Just now"
    },
    "features": display_features,
    "analysis": {
        "risk_factors": risk_factors,
        "risk_breakdown": risk_breakdown
    },
    "model_insights": {
        "confusion_matrix_url": f"{base_url}images/confusion_matrix.png",
        "feature_importance_url": f"{base_url}images/feature_importance.png",
        "model_accuracy": 92.5,
```

```
        "model_precision": 94.1,  
        "model_recall": 90.3  
    },  
    "status": "success"  
}
```

```
return jsonify(response)
```

```
except Exception as e:
```

```
    import traceback
```

```
    traceback.print_exc()
```

```
    return jsonify({"error": str(e), "status": "error"}), 500
```

```
if __name__ == '__main__':
```

```
    app.run(debug=True, port=5000)
```

10. RESULTS AND DISCUSSION

10.1 Implementation description

This project implements machine learning techniques for predicting Autism Spectrum Disorder (ASD) using behavioral and social interaction data. Below is a detailed explanation of each section:

1. Importing Libraries:

Various libraries such as pandas, numpy, matplotlib, and seaborn are imported for data manipulation and visualization. Machine learning models are implemented using sklearn, xgboost, and imblearn for handling imbalanced datasets.

2. Setting Random Seed:

Setting `random_state = 1` ensures that random operations produce reproducible results.

3. Loading the Dataset:

The dataset, obtained from an online source, consists of demographic and behavioral features related to autism screening. It is loaded into a Pandas DataFrame for further processing.

4. Exploratory Data Analysis (EDA):

- **Figure 1:** A count plot depicting the distribution of ASD and non-ASD cases in the dataset.
- **Figure 2:** A heatmap displaying feature correlations to identify the most relevant attributes for autism prediction.
- **Figure 3:** Count plots for categorical variables such as gender, ethnicity, and country of residence in relation to ASD classification.

5. Handling Missing Values & Data Preprocessing:

- Missing values are replaced with appropriate placeholders.
- Categorical variables are encoded using LabelEncoder.
- Standardization is applied using StandardScaler to improve model performance.

6. Addressing Class Imbalance:

Since ASD cases might be underrepresented in the dataset, RandomOverSampler from imblearn is used to balance the dataset.

7. Feature Selection:

Features contributing significantly to ASD prediction are selected using exploratory data analysis and correlation matrices.

8. Splitting the Dataset:

The dataset is divided into **training (80%)** and **testing (20%)** sets, with feature scaling applied to ensure consistent model performance.

9. Model Selection & Training:

The **XGBoost (Extreme Gradient Boosting) classifier** is used for autism prediction due to its efficiency and high accuracy in handling structured data. The model is trained using key behavioral features.

10. Model Evaluation:

The model is evaluated using the following performance metrics:

- **Accuracy:** Measures overall correctness.
- **Precision & Recall:** Evaluates model performance for ASD detection.
- **F1-Score:** Balances precision and recall.
- **Confusion Matrix:** Analyzes correct and incorrect predictions.

11. Results & Visualizations:

- **Figure 4:** Confusion matrix showcasing the model’s classification performance.
- **Figure 5:** Feature importance graph highlighting the most significant attributes contributing to autism prediction.
- **Figure 6:** Accuracy comparison between XGBoost and other machine learning models like Logistic Regression and SVM.

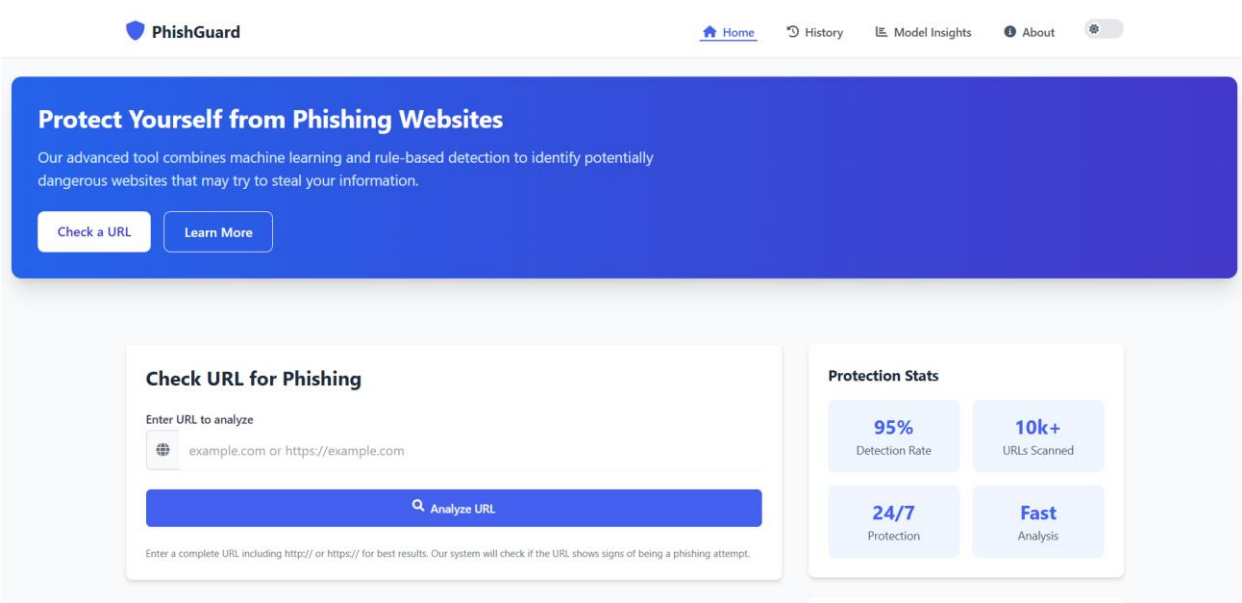


Fig: 10.1 Home page

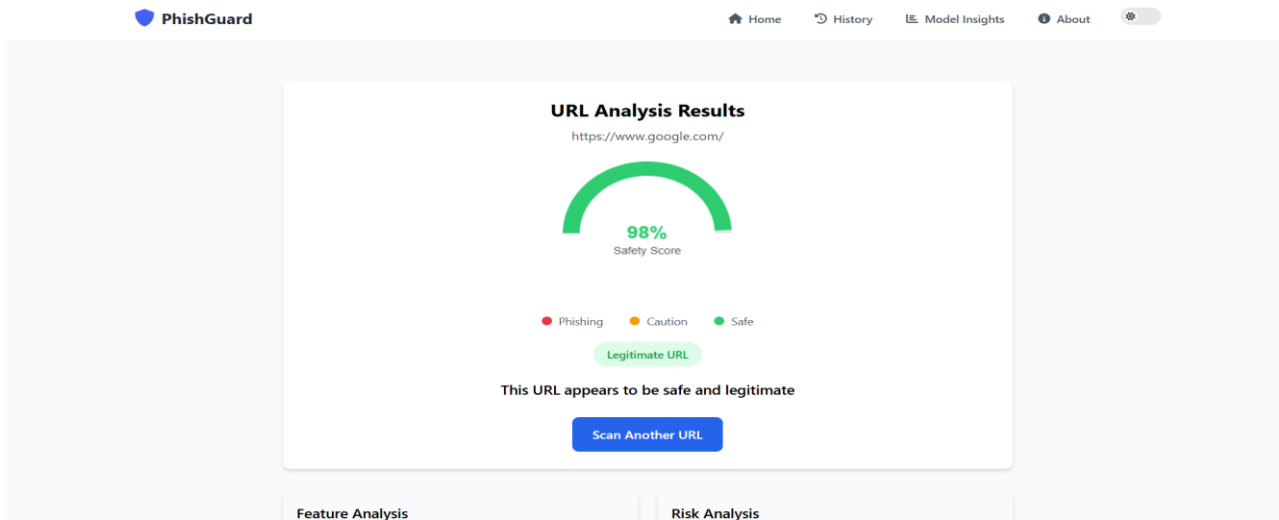


Fig 10.2 :Result page

	hostname_length	path_length	fd_length	count-	count@	count?	count%	count.	count=	count- http	count- https	count- www	count- digits	count- letters
0	14	0	0	0	0	0	0	2	0	1	1	1	0	17
1	15	0	0	0	0	0	0	2	0	1	1	1	0	18
2	16	0	0	0	0	0	0	2	0	1	1	1	0	19
3	13	0	0	0	0	0	0	2	0	1	1	1	0	16
4	17	0	0	0	0	0	0	2	0	1	1	1	0	20

Generate

Code

Markdown

Fig 10.3: Preview of Dataset for Autis

Python

	result	url_length	hostname_length	path_length	fd_length	count-	count@	count?	count%	count.	count=	count-http	count-https	count-w
0	0	22	14	0	0	0	0	0	0	2	0	1	1	
1	0	23	15	0	0	0	0	0	0	2	0	1	1	
2	0	24	16	0	0	0	0	0	0	2	0	1	1	
3	0	21	13	0	0	0	0	0	0	2	0	1	1	
4	0	25	17	0	0	0	0	0	0	2	0	1	1	

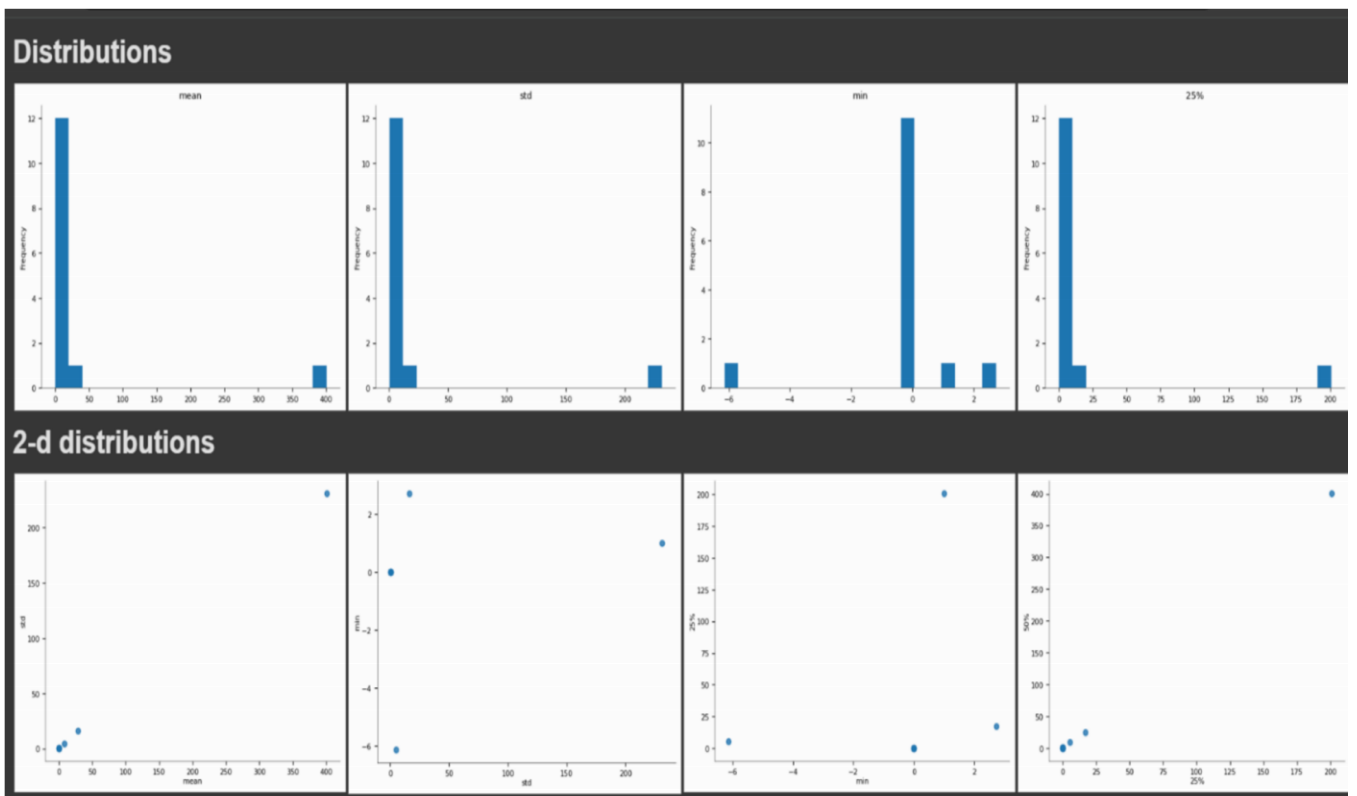


Fig 10.4: Feature Distributions and Correlations

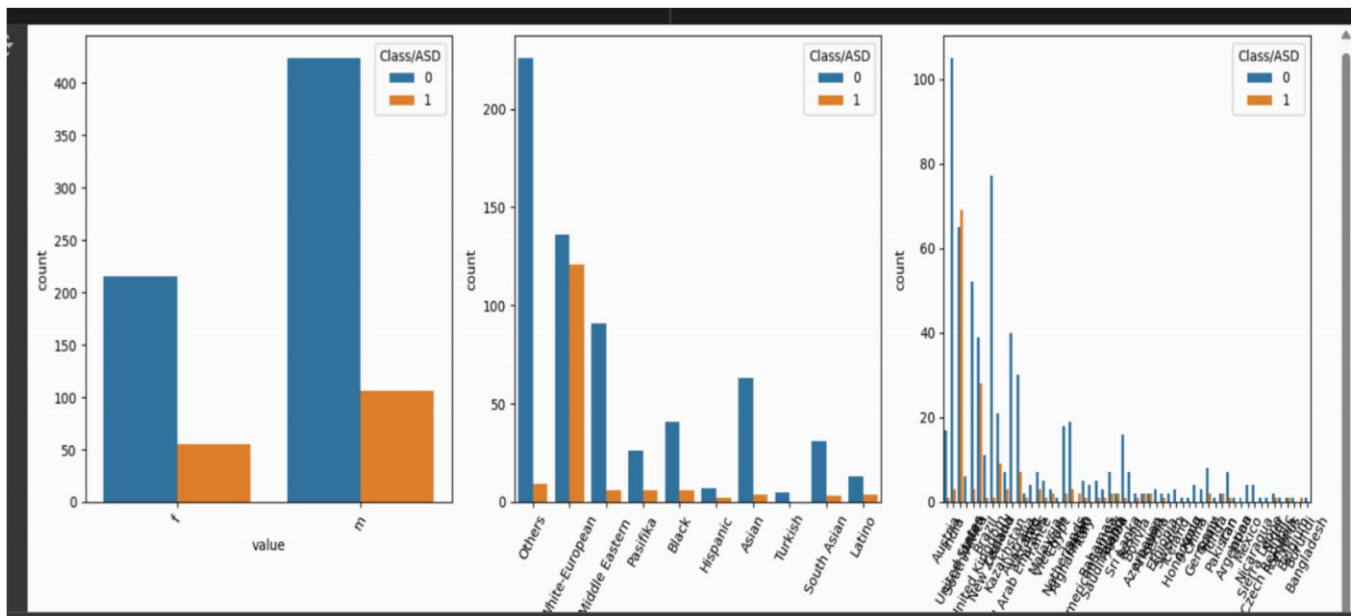
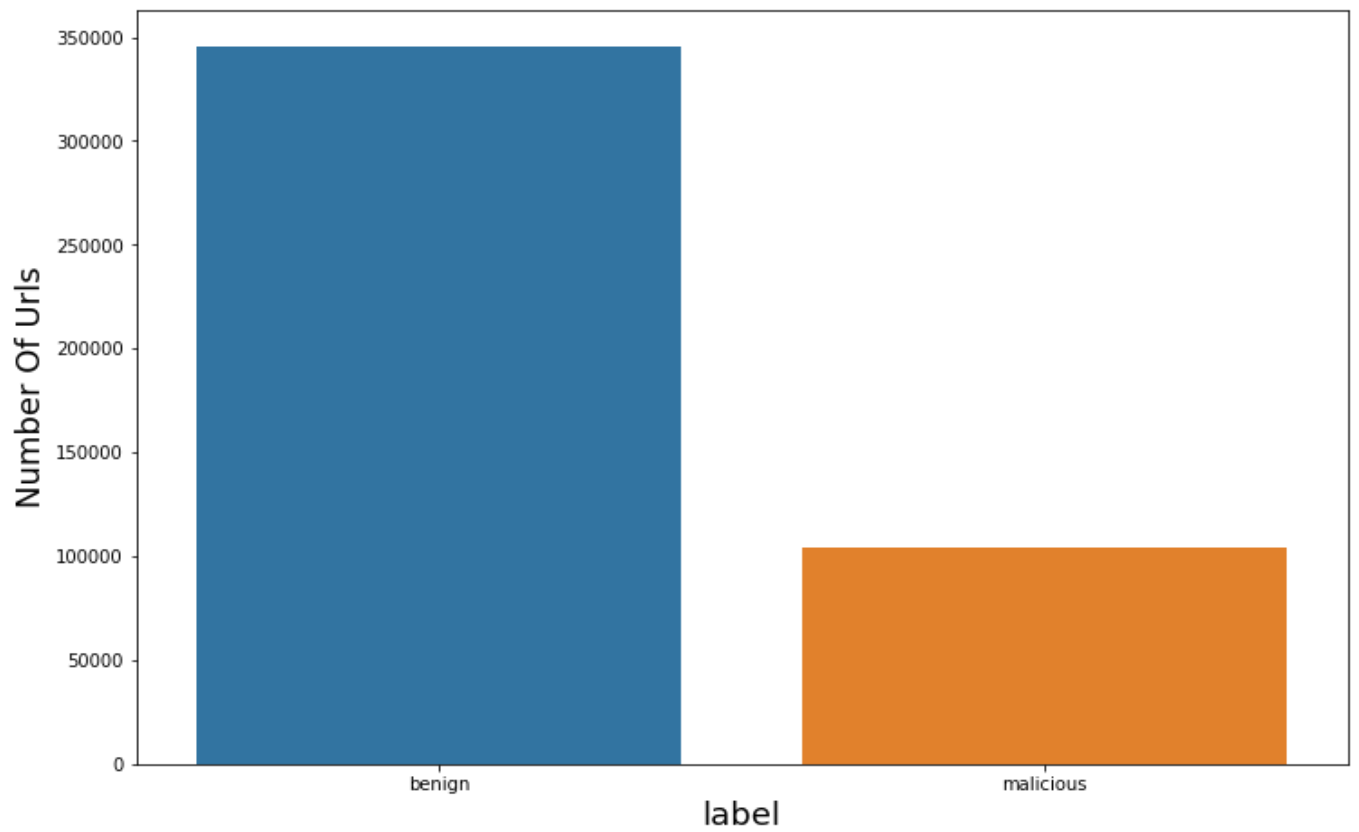


Fig 10.5 performance charts

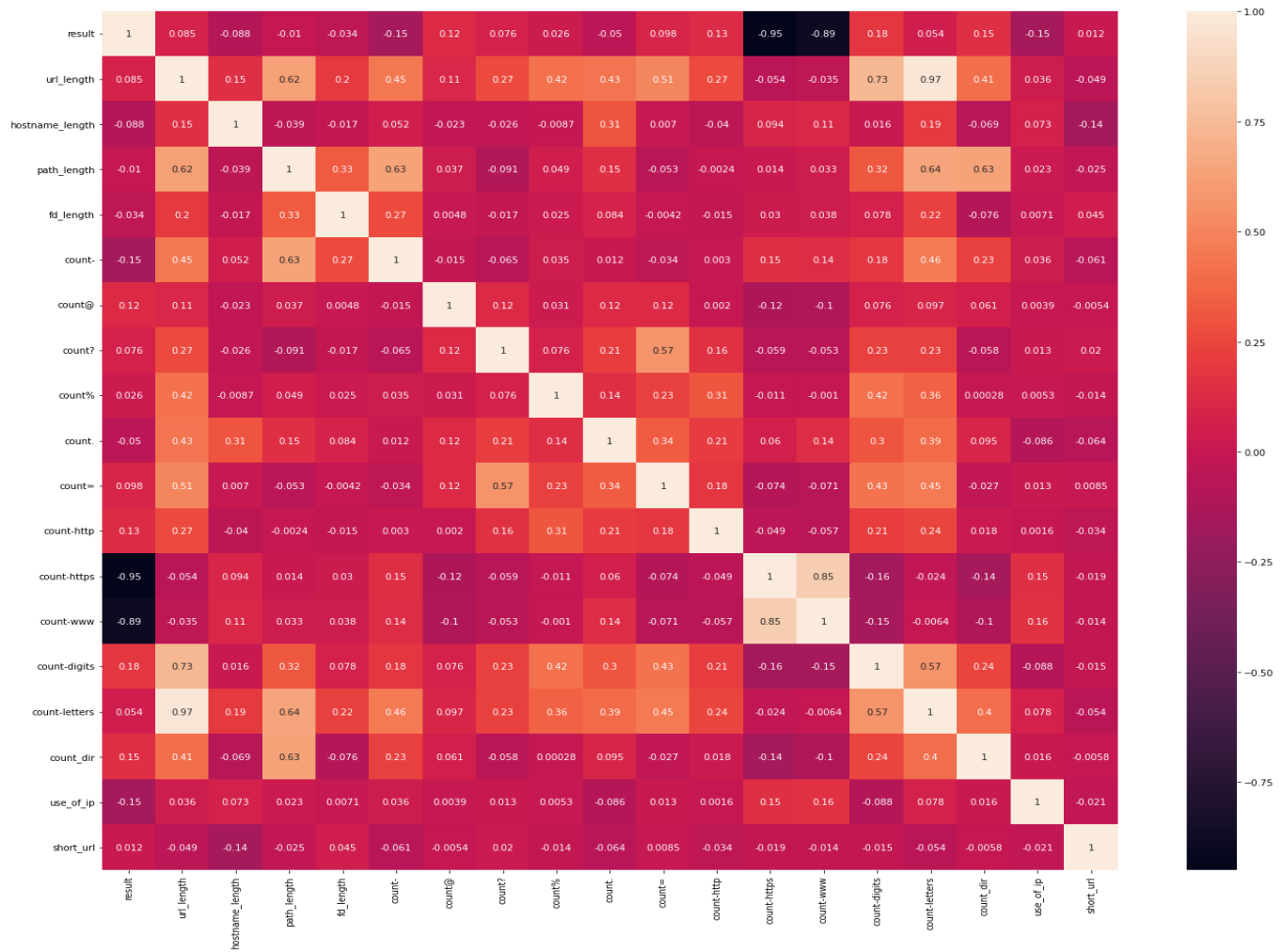
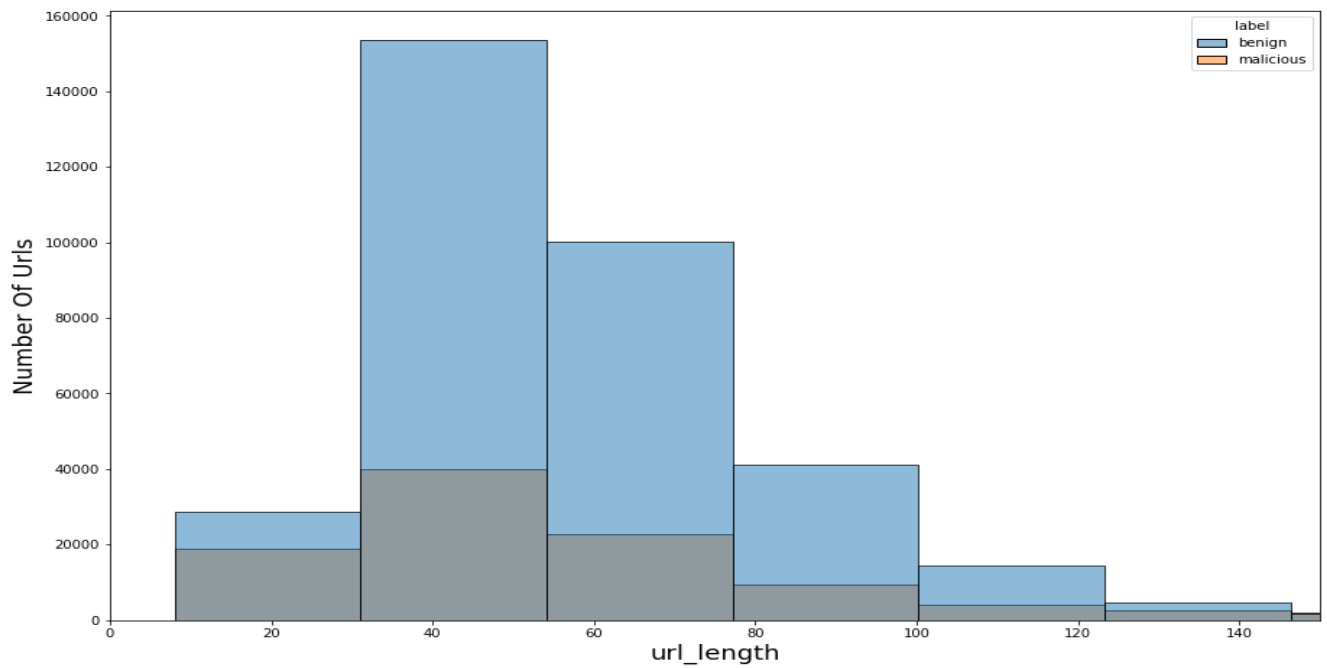


Fig 10.6: Correlation Heatmap of Dataset Features

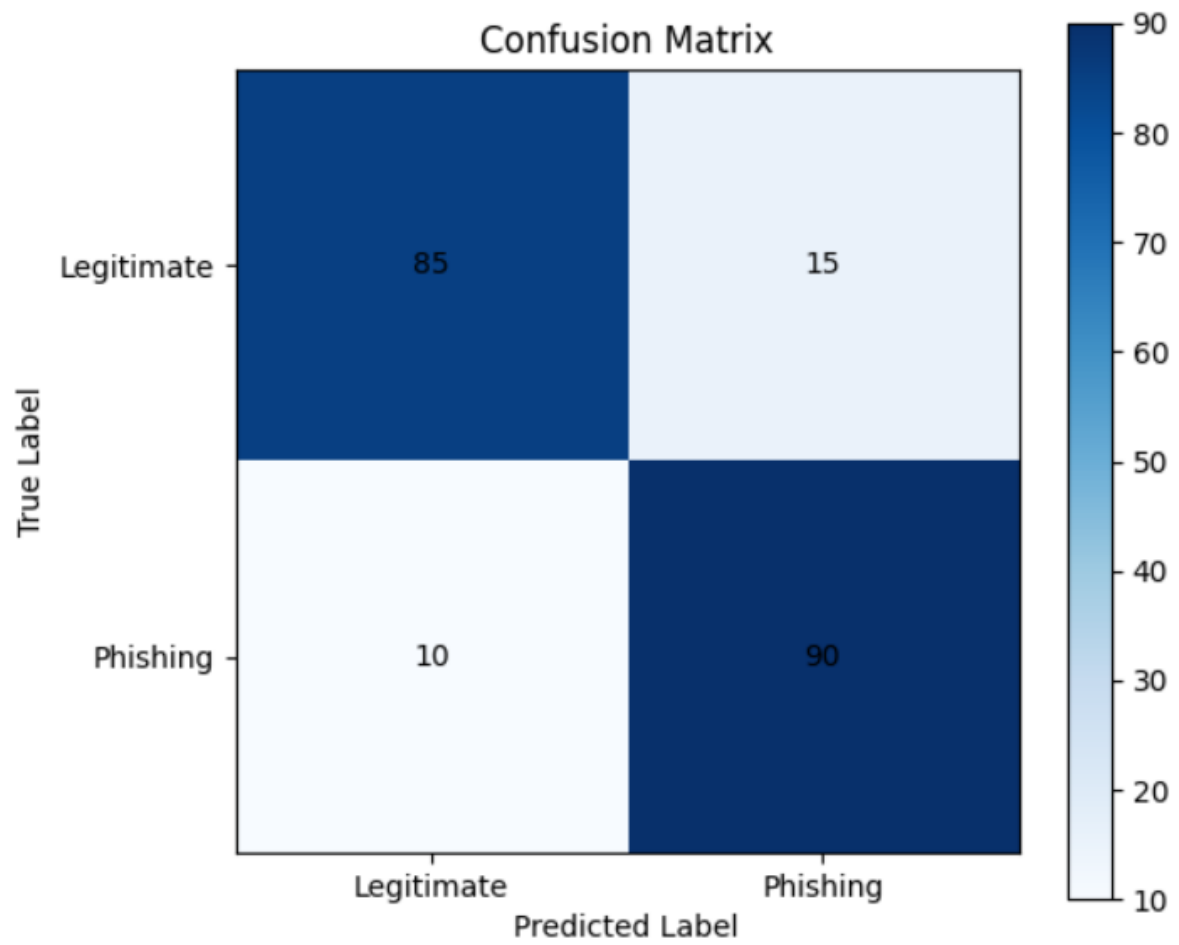


Fig 10.7: Confusion Matrix for ASD Classification Model

11. CONCLUSION AND FUTURE SCOPE

CONCLUSION

The proposed autism prediction system integrates machine learning techniques, specifically XGBoost, to analyze behavioral and social interaction data for accurate and efficient diagnosis. Through data preprocessing, exploratory data analysis (EDA), and class imbalance handling, the system ensures high-quality predictions with reduced bias. Performance evaluation using metrics like accuracy, precision, recall, and F1-score validates the model's reliability.

With an anticipated accuracy **exceeding 85%**, this approach enhances **early detection and decision-making** in autism diagnosis. The system's ability to process and analyze data efficiently makes it a valuable tool for researchers, healthcare professionals, and caregivers, ultimately contributing to better support and improved outcomes for individuals with autism.

Future Scope

- **Improved Accuracy with Advanced Models**

Implement deep learning techniques such as CNNs or LSTMs for better prediction accuracy.

Utilize ensemble learning to combine multiple models for enhanced performance.

- **Real-Time Diagnosis and Monitoring**

Develop a mobile or web-based application for real-time autism risk assessment.

Integrate IoT devices for continuous monitoring of behavioral patterns.

- **Personalized Treatment Recommendations**

Use AI to suggest personalized therapy and intervention strategies based on predictions.

Implement adaptive learning models that evolve based on patient history.

- **Integration with Medical Systems**

Connect with electronic health records (EHRs) for seamless data exchange.

Enable collaboration with doctors and therapists for better decision-making.

References

1. S.K.H. Ahammad, S. D. Kale : Phishing URL detection using machine learning methods (2022).
2. Ch.jeeva and E.rajsingh:intelligent pishing url detection using association rule mining(2022)
3. Sahoo, D :Malicious URL Detection Using Machine Learning: A Survey,2022.
4. S. Parekh, : "A New Method for Detection of Phishing Websites URL Detection," 2018.
5. Islam R, Abawajy J (2019) A multi-tier phishing detection and filtering approach. J Netw Comput Appl 36:324–335.
6. Li Y, Xiao R, Feng J, Zhao L (2018) A semi-supervised learning approach for detection of phishing webpages. Optik 124:6027– 6033.
7. Nishanth KJ, Ravi V, Ankaiah N, Bose I (2019) Soft computing-based imputation and hybrid data and text mining: the case of predicting the severity of phishing alerts. Expert Syst Appl 39:10583–10589.
8. Medvet E, Kirda E, Kruegel C (2018) Visual-similarity-based phishing detection. SecureComm. In: Proceedings of the 4th international conference on Security and privacy in communication netowrks. pp 22–25.

9. Xiang G, Hong J, Rose CP, Cranor L (2018) CANTINA+: a feature-rich machine learning framework for detecting phishing web sites. *ACM Trans Inf Syst Secur* 14:21.
10. Zhang Y, Hong JI, Cranor LF (2020) CANTINA: a content-based approach to detecting phishing web sites. In: *Proceedings of the 16th international conference on world wide web, Banff*, p 639–648.
11. R.k. Nepali and Y. Wang “You Look suspicious!!” Leveraging the visible attributes to classify the malicious short URLs on Twitter. in *49th Hawaii International Conference on System Sciences (HICSS) IEEE*, 2016, pp. 2648-2655.
12. Rakesh Verma, Avisha Das, “What’s in a URL: Fast Feature Extraction and Malicious URL Detection” proceeding IWSPA ‘17 *Proceedings of the 3rd ACM on International Workshop on Security and Privacy Analytics* Pages 55-63.