Phishing Website Detection

A major project report submitted in partial fulfilment of the requirement for the award of a degree of

Bachelor of Technology

in

Computer Science & Engineering / Information Technology

Submitted by

Naman Sharma (201362)

Swastik Jha (201368)

Under the guidance & supervision of

Dr. Nishant Sharma

Department of Computer Science & Engineering and Information Technology

Jaypee University of Information Technology, Waknaghat, Solan - 173234 (India)

CERTIFICATE

This is to certify that the work which is being presented in the project report titled **“Phishing Website Detection”** in partial fulfilment of the requirements for the award of the degree of **Bachelor of Technology** in **Computer Science & Engineering / Information Technology** submitted to the Department of Computer Science And Engineering, Jaypee University of Information Technology, Waknaghat is an authentic record of work carried out by **Naman Sharma(201362)** and **Swastik Jha(201368)** during the period from August 2023 to May 2024 under the supervision of **Dr. Nishant Sharma,**(Assistant Professor(SG), Department of Computer Science and Engineering, Jaypee University of Information Technology).

Naman Sharma	Swastik Jha
Roll No.: 201362	Roll No.: 201368

The above statement made is correct to the best of our knowledge.

Supervisor Name:
Dr. Nishant Sharma
Designation: Assistant Professor (SG)
Department: CSE/IT

DECLARATION

We at this moment declare that the work presented in this report entitled **'Phishing Website Detection'** in partial fulfilment of the requirements for the award of the degree of **Bachelor of Technology** in **Computer Science & Engineering / Information Technology** submitted in the Department of Computer Science & Engineering and Information Technology**,** Jaypee University of Information Technology, Waknaghat is an authentic record of our work carried out over a period from August 2023 to May 2024 under the supervision of **Dr. Nishant Sharma** (Assistant Professor(SG), Department of Computer Science & Engineering and Information Technology).
The matter embodied in the report has not been submitted for any other degree or diploma award.

Naman Sharma	Swastik Jha
Roll No.: 201362	Roll No.: 201368

This is to certify that the above statement made by the candidate is true to the best of my knowledge.

Supervisor Name:
Dr. Nishant Sharma
Designation: Assistant Professor (SG)
Department: CSE/IT

ACKNOWLEDGEMENT

Firstly, I express my heartiest thanks and gratefulness to almighty God for His divine blessing to make it possible to complete the project work successfully.

We are grateful and wish out profound indebtedness to Supervisor **Dr Nishant Sharma, Assistant Professor (SG)**, Department of CSE Jaypee University of Information Technology, Waknaghat. Deep Knowledge & keen interest of our supervisor in the field of "**Information Security**" to carry out this project. His endless patience, scholarly guidance, continual encouragement, constant and energetic supervision, constructive criticism, valuable advice, reading many inferior drafts and correcting them at all stages have made it possible to complete this project.

We would also generously welcome each one of those individuals who have helped me straightforwardly or in a roundabout way in making this project a win. In this unique situation, we want to thank the various staff individuals, both educating and non-instructing, who have developed their convenient help and facilitated our undertaking.

Finally, we must acknowledge with due respect the constant support and patience of our parents.

Naman Sharma (201362)

Swastik Jha (201368)

TABLE OF CONTENT

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Online phishing is one of the most common attacks on the modern internet. The goal of phishing website uniform resource locators is to steal personal data including login credentials and credit card numbers. As technology keeps growing, phishing strategies began to develop rapidly. The aim of phishing site URLs is to collect the private information like the user's identity, passwords and online money related exchanges. Phishers use the sites which are visibly and semantically like those of authentic websites. Since the majority of the clients go online to get to the administrations given by the government and money related organizations, there has been a vital increment in phishing threats and attacks since some years. There has been a vital increment in phishing threats and attacks since some years.

As technology is growing, phishing methods have started to progress rapidly. It can be avoided by making use of anti-phishing techniques such as Deep CDR, Endpoint Compliance to detect phishing. Machine learning and deep learning is an authoritative tool that can be used to aim against phishing assaults. The machine learning approaches to detect phishing websites have been proposed earlier and have been implemented. The central aim of this project is to implement the system with high efficiency, accuracy and cost effectively.

Chapter 01: INTRODUCTION

In the current digital era, security experts are increasingly concerned about phishing. This is due to the fact that it is comparatively simple for attackers to produce phony websites that closely mimic authentic ones. Although experts are able to recognize these fraudulent websites, many users are not, leaving them open to phishing scams. The mainobjective of cybercriminals frequently involves pilfering confidential data, like login credentials for bank accounts. The following is a condensed description of how hackers execute phishing attacks: Usually, they send spam emails stating that your password for the university network is about to expire. The email provides a link to update your password, but clicking on it redirects you to a hacker-controlled server where they can steal your online data. In our project, we aim to predict whether a website is a phishing site or a legitimate one. We've collected the dataset from Kaggle [1]. URLs with no malicious detection are labeled as '-1' (benign), while those with detections are labeled as '1' (malicious).
We're exploring various machine learning algorithms to analyze the characteristics of these URLs. By understanding the features that indicate phishing, we hope to improve detection strategies. Phishing attacks often succeed due to a lack of consumer awareness, exploiting weaknesses in users.

One common technique is updating blacklisted URLs and IPs in antivirus databases, known as the "blacklist" method. However, attackers constantly find ways to bypass blacklists through tactics like URL obfuscation, fast- flux (rapidly changing proxies), and algorithmically generating new URLs. To lure people in, phishers send mass emails, prompting users to visit a spoofed website. These sites often trick users into running software or downloading files. Malicious URLs can be detected using machine learning techniques, providing a more efficient solution compared to traditional methods. For better protection against phishing, it's essential for individuals to understand how these websites appear in their browsers. High-end companies can employ machine learning and deep neural network algorithms to proactively blacklist or detect phishing websites early.Machine learning helps catch phishing websites by learning from examples of both real and fake sites. It looks for signs like strange web addresses or tricky content that are common in phishing. Once it learns what to look for, it can quickly decide if a new website seems like a phishing threat or not. This smart system works faster than older ways of finding phishing sites, making it better at keeping users safe from online scams. Essentially, machine learning acts like a smart detective that's always learning and staying one step ahead of the tricks that hackers use to create fake websites

Machine Learning in addition has proven to be the best approach to deal with such important aspects of Machine learning serves as a transformative paradigm in constructing real-life models by enabling computational systems to autonomously learn and derive insights from data, subsequently facilitating informed decision-making. This methodology obviates the need for explicit programming, allowing models to discern intricate patterns and extrapolate predictions.

1.1 Problem Statement

Online phishing is one of the most common attacks on the modern internet. Phishers use the sites which are visibly and semantically like those of authentic websites.

The problem is derived after making a thorough observation and study about the method of classification of phishing websites that makes use of machine learning techniques. We must design a system that should allow us to:

- Accurately and efficiently classify the websites into legitimate or phishing.
- Time consumed for detection should be less and should be cost effective.

1.2 Objective

- Phishing websites' most popular social engineering approach imitates reliable URLs and web pages.
- The study aims to train machine learning models and deep neural nets on a dataset designed to detect phishing websites.
- The website's benign and phishing URLs are gathered to create a dataset, from which necessary URLs and content-based features are retrieved
- Every model's efficiency is measured and contrasted.
- A phishing internet site most common social engineering approach that mimics trustful URLs andweb pages.

1.3 Methodology

In this project, our primary focus has been on leveraging machine learning techniques and deep learning techniques to assess and classify links as either "good" or "bad." The core of our approach lies in the utilization of various algorithms for training and prediction purposes. To implement this, we have employed the Flask framework, a powerful Python web framework, and seamlessly integrated multiple libraries catering to diverse functionalities. Our choice of algorithms encompasses a comprehensive set, including Random Forest, Decision Tree, Support Vector Machine (SVM), K-Nearest Neigbhor, Naive Bayes, Gradient Boost and Multi-layer Perceptron. These algorithms serve as the backbone of our predictive model each contributing unique strengths to the overall accuracy of link classification.

The implementation involves a two-step process: training the algorithms on a labeled dataset to learn the patterns associated with good and bad links, and subsequently, utilizing trained models for real-time prediction of new links. By employing this combination of machine learning algorithms and performance evaluation metrics, we aim to provide a robust and accurate solution for link classification, ensuring the reliability and efficiency of our system in distinguishing between good and bad links.

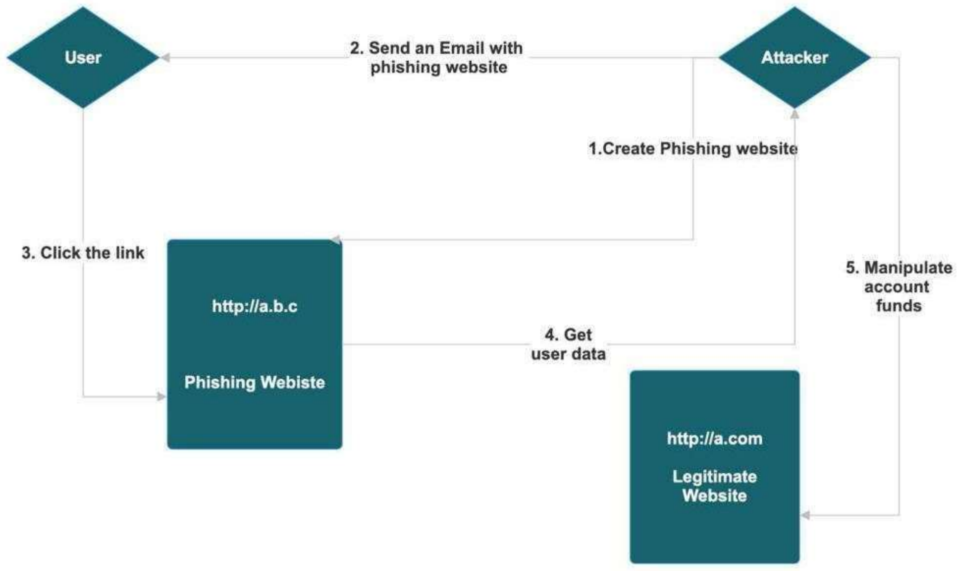


Figure 1.1 Flow of Methodology

1.4 Organization

CHAPTER 1 - INTRODUCTION: This chapter serves as a project and launching pad to provide a comprehensive background for secure cloud backup and recovery. It includes an introduction to the project, defines the problem, outlines the objectives, reflects the purpose and motivation of the project and decides the organization of the projectpreview report.

CHAPTER 2 - LITERATURE SURVEY: This chapter focuses on extensive research and integrates information from a variety of reputable sources, including standard books, journals, websites, and technical publications. The pertinent literature is printed, prior research on the subject is highlighted, and vital traits in cloud storage, backup, healing, and encryption are mentioned.

CHAPTER 3 - SYSTEM DEVELOPMENT: This chapter discusses the technical additives in the order of requirements and evaluation, venture planning, and architecture. It covers the statistics education andimplementation system and introduces the most vital parts which include code samples, algorithms, equipment and strategies. The fundamental difficulties of the development procedure and the way to remedy them are also examined.

CHAPTER 4 – PERFORMANCE ANALYSIS: This chapter discusses the trying out technique, emphasizing the devices and strategies used to evaluate the platform and its performance, emphasizing the platform and ensuring reliability. It presents a thorough evaluation of the system with the aid of supplying a summary of take a look at instances and their outcomes.

CHAPTER5 - CONCLUSIONS: This insightful chapter provides an overview of the project and key findings, limitations and industry contributions. It follows with recommendations for future research pathways, paving the way for more investigation and innovation by providing guidelines for fine-tuning testing procedures, investigating additional performance indicators, and addressing outstanding problems.

Chapter 02: LITERATURE SURVEY

- Chunlin Liu, Bo Lang: Finding effective type for malicious URL detection [3]

Chunlin et al. presented a technique in 2018 that primarily concentrates on particular frequency aspects. Their method yields more accurate results in the classification of dangerous URLs by combining machine learning techniques with statistical analysis of URLs. By contrasting their suggested algorithm with six other machine learning algorithms, they were able to determine how effective it was. The precision of 99.7% was demonstrated in the results, and the false positive rate was kept far below 0.4%. This suggests that their approach to correctly detecting and categorizing dangerous URLs is quite dependable.

- Machine Learning to Combat Phishing Scams, FadiThabtah :

Fadi Thabtah and colleagues used real phishing datasets and a range of variables to compare multiple machine learning approaches in their trials. The goal was to illustrate the benefits and drawbacks of machine learning predictive models as well as show how well they actually work to combat phishing scams. The findings showed that covering technique models work better as anti-phishing defenses. Muhemmet Baykara and colleagues also presented an Anti-Phishing Simulator application. This program advises on how to spot phishing emails and sheds light on the difficulties associated with phishing detection. The database is updated with spam emails through the use of a Bayesian algorithm. According to the study, the best approach is to use the email text as a keyword for effective word processing solely.

- Client-side detection of phishing websites, Ankit Kumar Jain, B. B. Gupta [4]:

Regarding the client-side detection of phishing websites by the application of machine learning: Gupta et al. and colleagues presented a novel anti-phishing strategy in Springer Science+Business Media, LLC, a division of Springer Nature 2017 that focuses exclusively on extracting features from the client side. The study's overall accuracy in identifying phishing websites was an amazing 99.09%. The method's limitation—that it can only identify websites written in HTML—is acknowledged by the authors. This method is ineffective for identifying websites that use non-HTML code.

- A Prior-based transfer learning techniques for the Phishing Detection, Yang Xin , Dan Li, Yangxi Ou [5]:

Specifically, they utilize a logistic regression as the basis of a concern-based transferable learning method in our statistical machine learning classification for phishing website identification based on specific URL attributes. We have presented many models, each suited to a different phishing domain, due to the variable distribution of features in these domains. When collecting sufficient data for a new area is not feasible to repair the detection model and use transfer learning, our suggested workaround is to use a URL-based method for phishing detection. We support the use of a transfer learning technique to solve potential flaws in trait detection and build a more reliable and powerful model.

- Feature Extraction Method, Ahmad Abunadi, Oluwatobi Akanb [6]:

Feature Extraction Method: An Approach to Phishing Detection To make the solution accessible, they developed a user-friendly Flask web application in 2013. The program allows users to enter URLs and have the trained XGBoost model classify them in real time. This user-friendly tool allows individuals and groups to assess URL properties and identify potential security threats.

- Effective detection of phishing URLs based on machine learning, Sahingoz, O.K., Buber, E., Demir, O. and Diri, B., 2019 [7].

Effective detection of phishing URLs based on machine learning. 345–357 in Expert Systems with Applications, 117. It's critical to identify phishing websites in order to shield users from fraud. Individuals frequently fall victim to these frauds because they are ignorant of internet addresses, are unable to determine which websites are reliable, are too busy to double-check, or may not be able to view the entire address owing to deceptive tactics. To increase the accuracy of this identification process, they used a variety of techniques, including random forests and decision trees.

Chapter 03: SYSTEM DEVELOPMENT

3.1 Analysis of the Algorithms:

As supervised machine learning is the foundation of our entire project. A subset of artificial intelligence and machine learning is supervised learning. The way it operates is that we feed a model data and a label, and after the model is trained, it finds patterns in the data, links the labels to those patterns, and generates new predictions. Supervised learning has an enormous range of possible applications. A few of them may be used for spam detection or spam detection. This is how we determine whether an email is spam or not: if it's spam, it will be automatically deposited in a spam folder; otherwise, it will be deposited in your inbox. Classifying objects is another, among many others. Two types of supervised machine learning exist:

- **Classification:**

Not only does it assist in finding items that we can look up using keywords, but it also finds our inventions that are extremely similar to ours. A section that is divided into topic areas known as classes and subclasses. Predictive modeling is a sort of problem that involves estimating the mapping characteristic from input variables to discrete output variables. Take an email spam detector, for example. The main objective of a classification algorithm, which is specially designed to forecast the result for a given collection of data, is to choose the category for the dataset. An algorithm that implements the type using a dataset is called a classifier. [8]

- **Regression:**

It is a supervised learning technique that makes it possible to determine the correlation between variables and predict the continuous output variable based only on the most basic or significant predictor variables. Prediction, forecasting, time collection modeling, and determining the causal-impact relationship between variables are among its specific uses. Regression involves creating a graph with the variables that best fits the provided data points; by using this plot, the machine learning version of the data can be predicted. [9]

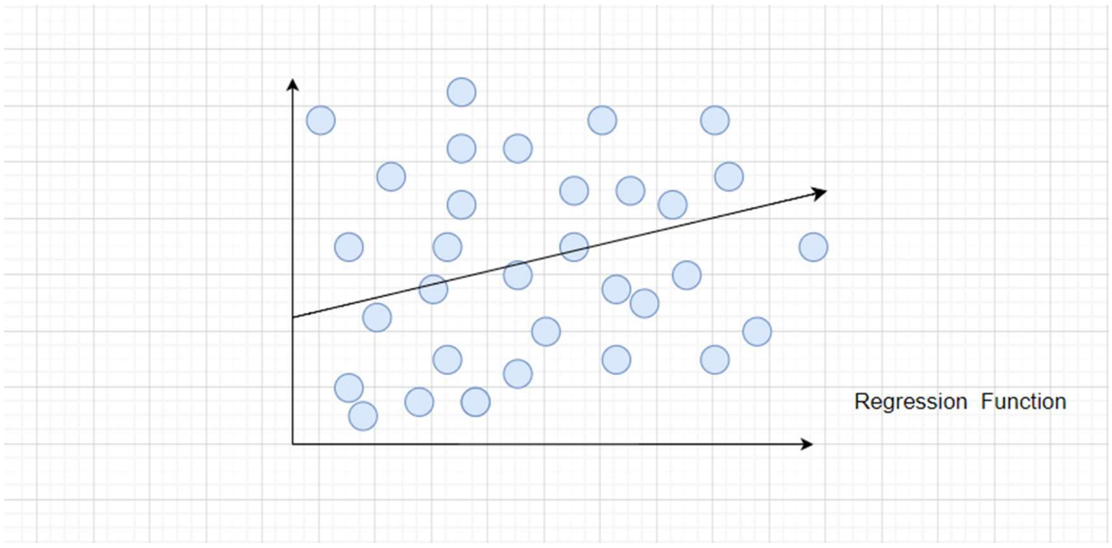


Figure 3.1 Regression

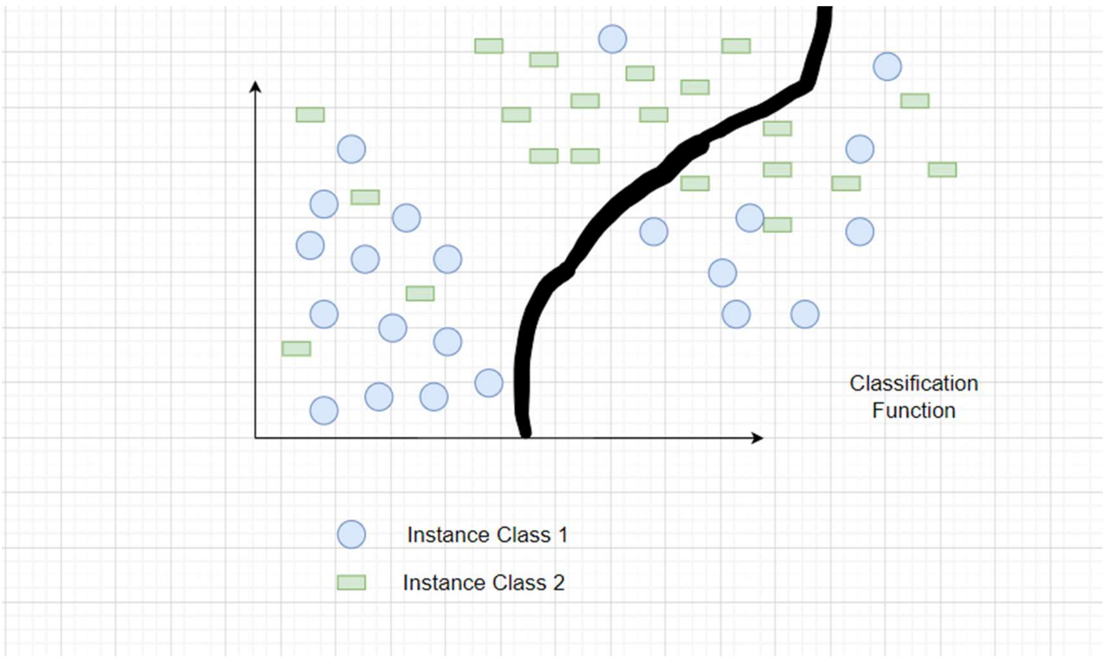


Figure 3.2 Classification

We used the Python framework Flask in our project and imported numerous libraries for various uses. Eight algorithms have been selected: Random Forest, Decision Tree, SVM, Gradient Boost, Naive Bayes, K-Nearest Neighbor, Logistic Regression and Multi-layer Perceptron. Thus, this contains a variety of URL types, including phishing, benign, and spam URLs. We have taken dataset from Kaggle [1] which consists of 11,054 rows and 32 columns.

We examine various machine learning algorithms for analyzing the characteristic in order to gain a thorough understanding of how the URLs that propagate phishing are constructed. For this project, the supervised machine learning techniques and deep learning techniques that we have selected are:

3.1.1 Random Forest:

Many random decision trees make up a random forest. The trees contain two different kinds of randomness. Every tree is initially constructed using a random sample of the initial data. Secondly, to produce the best split, a subset of features is randomly chosen at each node in the tree. More trees will prevent the model's trees from being overly fitted.

The majority of the data will remain accurate while the missing values are handled by the random forest classifier.[10]

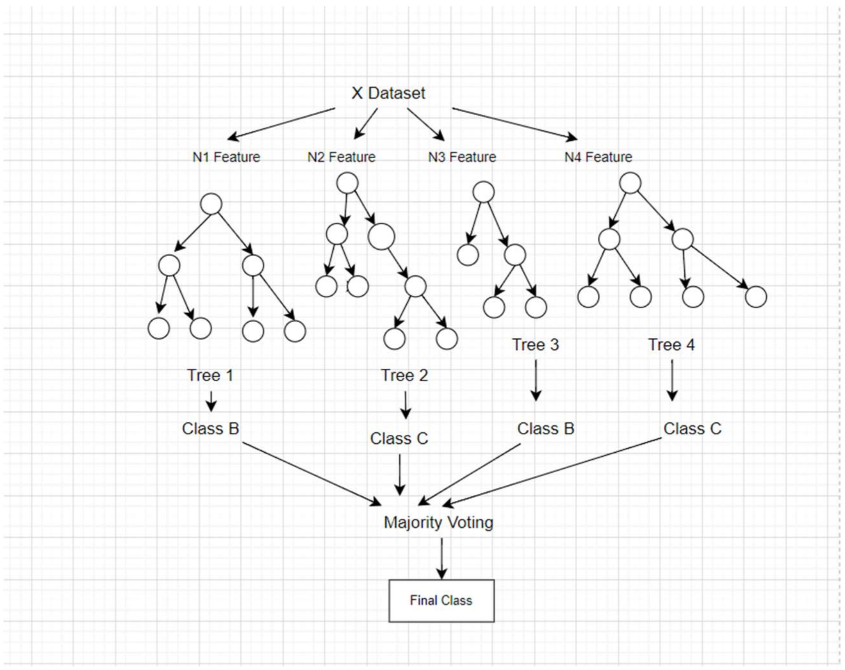


Figure 3.3 Random Forest

3.1.2 Decision Tree:

A random forest is made up of several random decision trees. There are two distinct types of randomness in thetrees. Each tree is first built with a random subset of the original data. Second, at each node in the tree, a subset of features is randomly selected in order to generate the best split. The model's trees won't be unduly fitted with more trees. While the random forest classifier fills in the missing values, the majority of the data will continue to be accurate.[10]

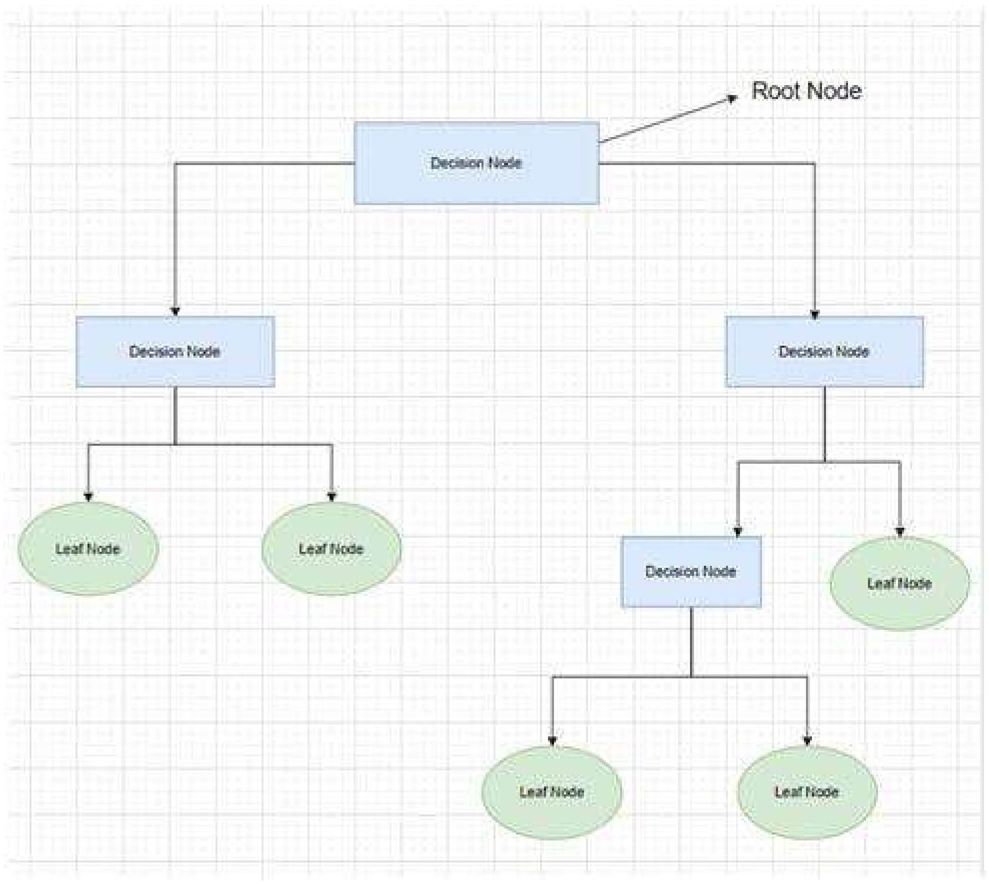


Figure 3.4 Decision Tree

3.1.3 SVM (Support Vector Machine):

Support vector machines (SVMs) are supervised machine learning algorithms that are versatile and strong, and they are utilized for both regression and classification. However, they are typically applied to classification issues. In essence, a hyper-plane in multidimensional space, the Support Vector Machine (SVM) model represents different classes. To reduce the error, the SVM will create the hyper-plane repeatedly. SVM seeks to determine a maximum marginal hyper-plane (MMH) by classifying the datasets. [12]

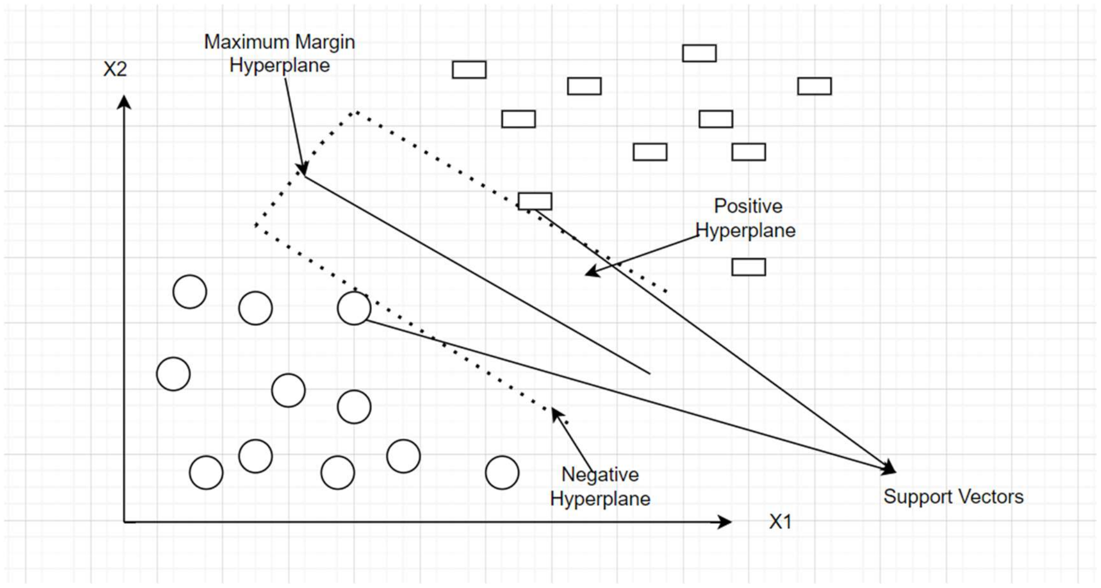


Fig 3.5 Support Vector Machine

3.1.4 Gradient Boost :

Under the Gradient Boosting framework, XGBoost is an open-source software library that implements optimized distributed gradient boosting machine learning algorithms. Gaining a thorough understanding of the machine learning principles and techniques that supervised machine learning, decision trees, ensemble learning, and gradient boosting are based on is essential to comprehending XGBoost. XGBoost is the ideal fusion of hardware and software capabilities created to optimize current boosting methods with maximum speed and accuracy. It is, in general, this algorithm's feasibility, accuracy, and efficiency. It contains algorithms for both tree learning and linear model solving. Therefore, its ability to perform parallel computation on a single machine is what gives it its speed.[13]

3.1.5 Flask

A lightweight Python web framework called Flask was created to make creating web apps and APIs easier. The WSGI (Web Server Gateway Interface) standard is adhered to, and it offers a simple and uncomplicated method for developing websites. Fundamentally, Flask helps developers create web applications by providing necessary tools and conventions, freeing them up to concentrate on writing application logic instead of managing the intricate details of handling HTTP requests and responses. Theframework is a well-liked option for projects ranging from small prototypes to larger-scale applications because of its simplicity, adaptability, and ease of use. Because Flask has a modular architecture, developers can easily add more features by integrating different extensions for things like form handling, database interaction, and authentication.[14].

3.1.6 Naive Bayes

"Naive Bayes classifiers" refers to the class of classification algorithms that use Bayes' Theorem as its foundation. Rather than being a single algorithm, it is actually a family of algorithms built on the same principle: every pair of characteristics that is being classed stands alone. It is based on the Bayes theorem. Based on the probability of a previous occurrence, the Bayes Theorem determines the likelihood of a current event. Naive Bayes classifiers are straightforward probabilistic frameworks in statistics that apply the Bayes theorem. The likelihood of a hypothesis given the available information and some prior knowledge forms the basis of this theorem. Given that each feature in the input data is assumed to be independent of every other feature, the naïve Bayes classifier often assumes that false in real-world situations. Despite this oversimplifying assumption of an extra incident that has already happened, the naive Bayes classifier is still widely utilized due to its effectiveness and high performance in several real-world applications. [16]

3.1.7 Multilayer Perceptron

An MLP comprises an input layer, one or more hidden layers, and a layer for output. It is a specific type of feed forward artificial neural network. All of the layers are completely interconnected. An essential idea in deep learning and neural networks will be covered in this article: the Multiple-layer Perceptron Neural Network. The only direction in which the Multilayer Perceptron (MLP) Neural Network may operate is forward. Every node is completely linked to the network. Every node only transmits its value in a forward way to the subsequent node. Backward propagation is an algorithm used by the MLP neural network to improve training model accuracy. [17]

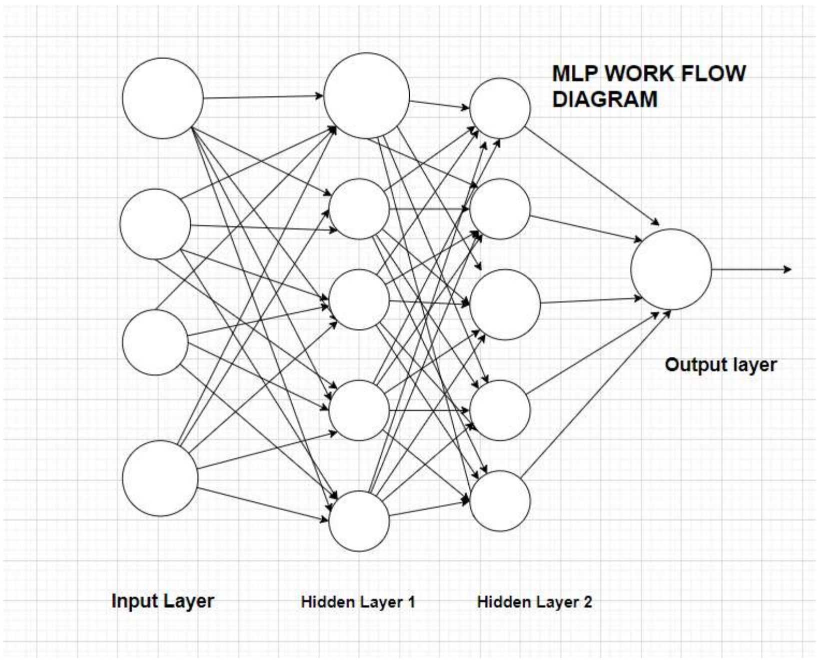


Figure 3.6 Multilayer Perceptron

3.1.8 Logistic Regression

Logistic regression (LR) is a supervised machine learning technique used to assess the likelihood of an event, a result, or an observation to resolve binary classification problems. The model's output is usually binary, meaning it can be either true or false. Logistic regression is a technique used to sort data according to the degree of correlation between one or more independent variables. It is frequently used in predictive modeling, in which an event's mathematical likelihood of falling into a particular category or not is determined by the model. In binary classification problems, where the outcome variable indicates one of the two categories (0 or 1), logistic regression is frequently utilized. The sigmoid function, a logistic function, is used to map predictions in logistic regression.In addition, if the estimated probability generated by the sigmoid function over a predefined threshold on the graph, the model predicts that the instance belongs to that class. The model states that if the computed probability is less than the preset threshold, the instance does not belong in the class. [18]

3.1.9 K-nearest neighbor

A supervised, non-parametric learning technique is the k-nearest neighbors algorithm (KNN). A data point's classification or grouping is predicted according to how close it is to nearby points. KNN is a flexible technique that is frequently used in machine learning for a range of regression and classification applications. KNN is an acronym for K-Nearest Neighbor. It's an algorithm for supervised machine learning. Problem assertions involving regression and classification can both be resolved by the algorithm. The sign "K" indicates the number of closest neighbors to a new unknown variable that has to be classed or forecasted. KNN determines the separation between every point in the vicinity of the unknown data and eliminates those that have the shortest distances to it. [19]

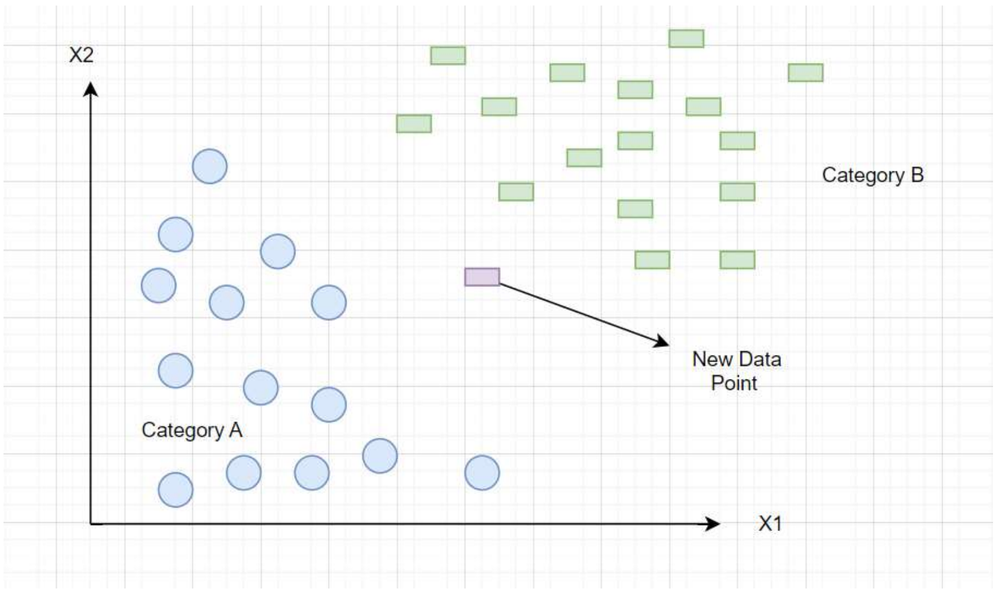


Figure 3.7 K-Nearest Neighbor

3.1.10 Docker

The freely accessible Docker platform makes it possible to use containerization to automate application deployment, scaling, and management. A lightweight substitute for classic virtualization is termed containerization, in which programs are packed with their dependencies and operate in separate contexts known as containers.

Regardless of the physical host system, containers offer apps a consistent and repeatable environment to function in. The libraries, binaries, and configuration files needed for the program to function properly are all contained in each container. Applications may operate reliably in a variety of settings, from development to production, thanks to this separation. This suggests that the program can function reliably in a variety of settings. Docker's speed is a result of its utilization of containerization technologies. By isolating the program and its dependencies from the host operating system, containerization enables effective resource management. This indicates that the program can function reliably in a variety of settings. Docker borrows all the resources from the base operating system. [20]

3.2 Design:

In our project, we focused on using machine learning techniques for link classification (good or bad). We implemented this using Flask, a Python framework, and incorporated algorithms like Random Forest, Decision Tree, KNN, Naive Bayes, Logistic Regression, Multilayer Perceptron and Gradient Boost. The classification report, featuring metrics such as recall and precision, evaluates the models' performance. Our approach ensures accurate and reliable link classification by leveraging diverse algorithms and thorough performance analysis. Alternatively, the collection of phishing URLs is simple because of the open source supplier known as Phish Tank. This supplier offers a set of phishing URLs that are updated hourly and come in a few different formats, including CSV and JSON. The dataset is immediately loaded into a Data Frame upon download.

We have used a comprehensive feature extraction process was undertaken to analyze the dataset. The features were categorized into distinct groups: Address Bar Based Features, Domain Based Features, and HTML and JavaScript Based Features. Our dataset includes multiple features designed to evaluate the features of URLs for possible phishing activity. These features include the use of an IP address, the length of the URL, the presence of shortened URLs, symbols like '@' in the URL, double slashes in redirection, prefixes or suffixes in the URL, the number of subdomains, the use of HTTPS, the duration since domain registration, the presence of a favicon, the use of non-standard ports, the presence of HTTPS in the domain part of the URL, request and anchor URLs, links in script tags, type of server form handler, the presence of email addresses, indication of abnormal URLs, website forwarding, customization of the status bar, right-click disable feature, usage of popup windows and iframe redirection, and the age of the domain The information includes the domain name system recording state, the anticipated traffic to the website, the page rank, the Google indexing status, the number of links referring to the page, the availability of a statistics report, and the label designating whether or not the URL is considered phishing. This dataset offers an extensive range of features for phishing threat analysis and URL classification. The combination of these three feature sets resulted in a comprehensive dataset comprising 32 features, providing a multifacetedperspective for our machine learning models to discern and classify URLs effectively based on their diverseattributes and characteristics.

The goal of this methodical feature combination and selection process was to improve our model's robustness and accuracy in identifying phishing and genuine URLs. We used machine learning algorithms to classify URLs after finishing feature extraction and merging datasets, and in the end, we discovered that Gradient Boost produced the best accuracy out of all the algorithms we looked at. The trained Gradient Boost model was then stored for later use. We developed a Flask application with two HTML pages, index.html and result.html, to create an easy-to-use user interface.

Users can enter URLs for classification using the index.html page, which also acts as an input interface. The results of the classification are shown on the result.html page. The Flask application, defined in the app.py file,was executed on our local system, providing a practical platform for users to interact with the trained Gradient Boostmodel in real-world scenarios. This cohesive solution, integrating the robust Gradient Boost model with the Flask web application, offers a user-friendly tool for users to assess and categorize URLs based on the machine learning model's predictions. [14]

Eight different algorithms were thoroughly compared, with an emphasis on how accurately they could classify URLs. Random Forest, Decision Tree, Multi-layer Perceptron, KNN, Logistic Regression, Support Vector Machine (SVM) and Gradient Boost, were the algorithms that were being examined. Following the training and assessment of every model, we discovered that Gradient Boost performed better than the others, obtaining an astounding accuracy score of 97.4%. The comparison process involved training each algorithm on labeled datasets, assessing their performance, and selecting the model with the highest accuracy for further deployment. Gradient Boost, with its ensemble learning approach and gradient boosting techniques, demonstrated superior predictive power in distinguishing between different types of URLs, showcasing its effectiveness in our specific use case.

The choice of Gradient Boost as the optimal model underscores its ability to handle complex relationships within the data, providing a robust solution for URL classification in our project. This high accuracy score serves as a testament to the efficacy of Gradient Boost in addressing the challenges posed by diverse URL characteristics and categories. The Dockerfile describes the processes required to containerize our application within the Ubuntu EC2 environment. Beginning with a base Ubuntu image, we describe the installation of critical dependencies, such as Python packages and any libraries needed for our detection techniques. Next, we put our application code into the container and set up the environment to guarantee a smooth execution. In addition, we may expose appropriate communication ports and declare any runtime instructions required to begin our program within the container. This Dockerfile encapsulates our phishing detection system into a portable, self-contained entity, making deployment easier and assuring consistent performance across several settings. Overall, by using Docker and creating an efficient Dockerfile specific to our project's needs, we improve the agility, scalability, and robustness of our phishing detection system in the Ubuntu EC2 environment.

3.3 Flow Chart

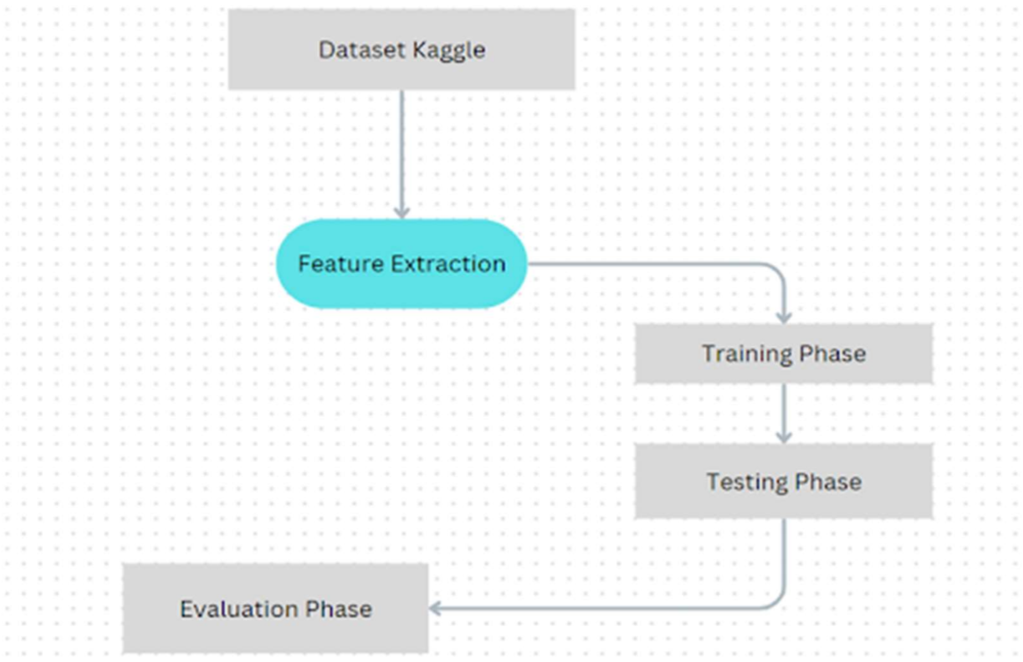
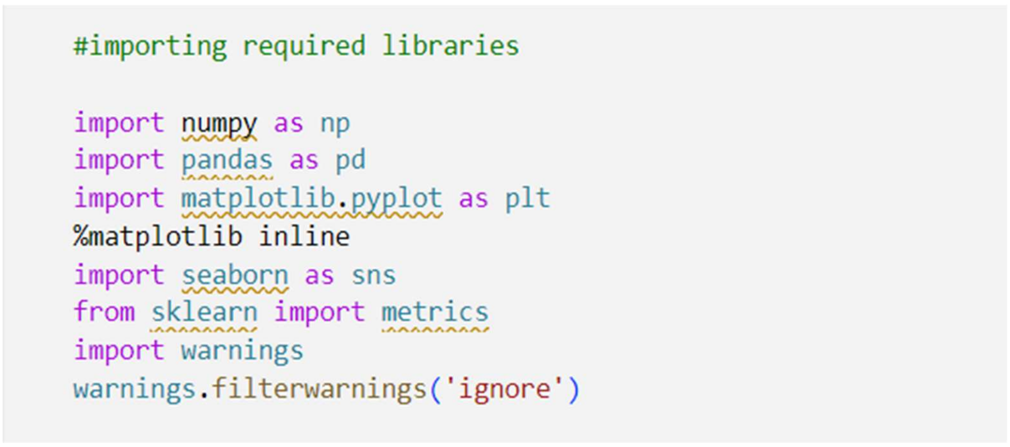


Figure 3.3.1 Flow Chart

We have predicted whether phishing websites are good or bad URLs in this project. We searched online and discovered some datasets before running any code. We developed a small piece of code to extract the feature after gathering the phishing and estimated websites from open source platforms. Due to the fact that there are various kinds of URLs, including malicious, phishing, spam, and benign URLs. We decided to take the dataset from Kaggle. The datasets have been reviewed, pre-processed, and separated into training and test sets. We employed machine learning algorithms and deep learning algorithm to categorize URLs, ultimately finding that Gradient Boost yielded the highest accuracy among the algorithms considered. Subsequently, we saved the trained Gradient Boost model for future use. To create an intuitive user interface, we developed a Flask application with two HTML pages: index.html and result.html. Flask application, defined in the app.py file, was executed on our local system, providing a practical platform for users to interact with the trained model. Further we have created a Dockerfile which offers us several benefits such as consistency, portability, isolation and scalability. Docker helps in ensuring that application behaves consistently across different environments and can be easily deployed in productions.

We have carried out a number of training implementations for this project, and we have predicted the legitimacy of phishing websites, some of the supervised algorithms with deep learning algorithms have been used.

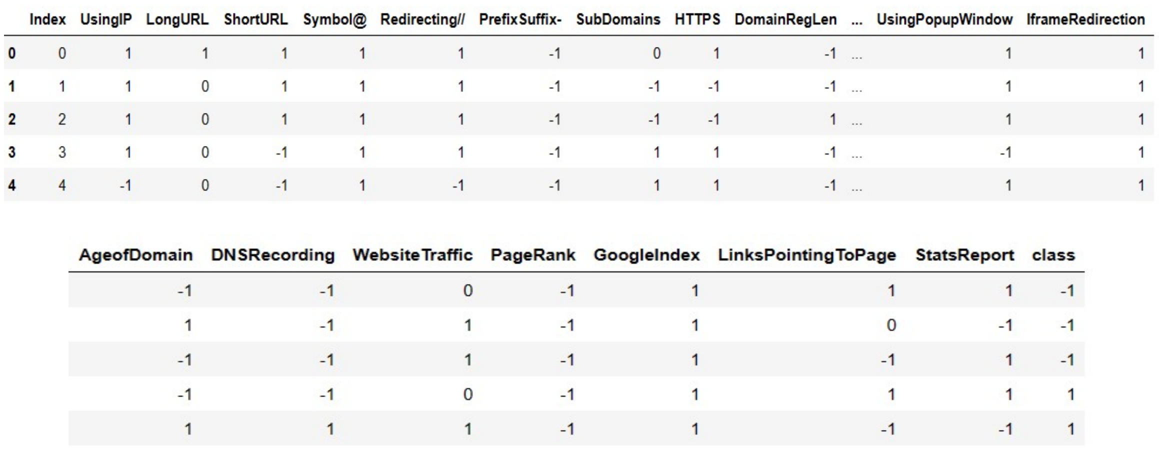
- Firstly, we imported some libraries such as pandas, numpy , matplotlib, Decision Tree Classifier andmany more.



```
#importing required libraries

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from sklearn import metrics
import warnings
warnings.filterwarnings('ignore')
```

Figure 3.3.2 Libraries

- We have taken the dataset from Kaggle which consists of 32 features.



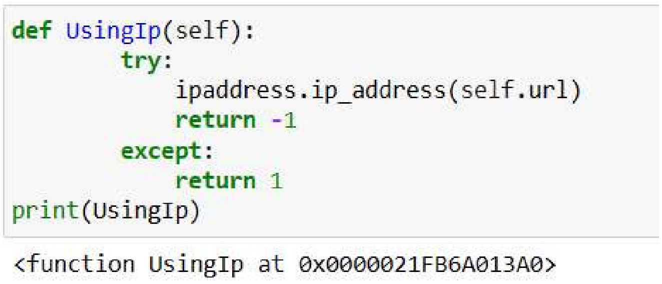
	Index	UsingIP	LongURL	ShortURL	Symbol@	Redirecting//	PrefixSuffix-	SubDomains	HTTPS	DomainRegLen	...	UsingPopupWindow	IframeRedirection
0	0	1	1	1	1	1	-1	0	1	-1	...	1	1
1	1	1	0	1	1	1	-1	-1	-1	-1	...	1	1
2	2	1	0	1	1	1	-1	-1	-1	1	...	1	1
3	3	1	0	-1	1	1	-1	1	1	-1	...	-1	1
4	4	-1	0	-1	1	-1	-1	1	1	-1	...	1	1

AgeofDomain	DNSRecording	WebsiteTraffic	PageRank	GoogleIndex	LinksPointingToPage	StatsReport	class
-1	-1	0	-1	1	1	1	-1
1	-1	1	-1	1	0	-1	-1
-1	-1	1	-1	1	-1	1	-1
-1	-1	0	-1	1	1	1	1
1	1	1	-1	1	-1	-1	1

Figure 3.3.3 Dataset Features

- Firstly we have performed feature extraction.

  - Using IP

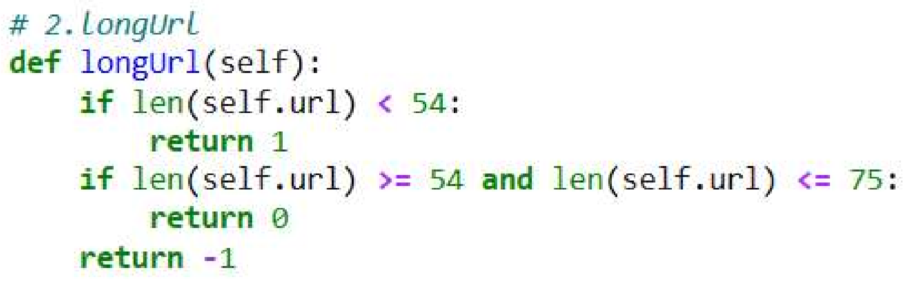


```
def UsingIp(self):
        try:
            ipaddress.ip_address(self.url)
            return -1
        except:
            return 1
print(UsingIp)
```

```
<function UsingIp at 0x0000021FB6A013A0>
```

Figure 3.3.4 IP Address

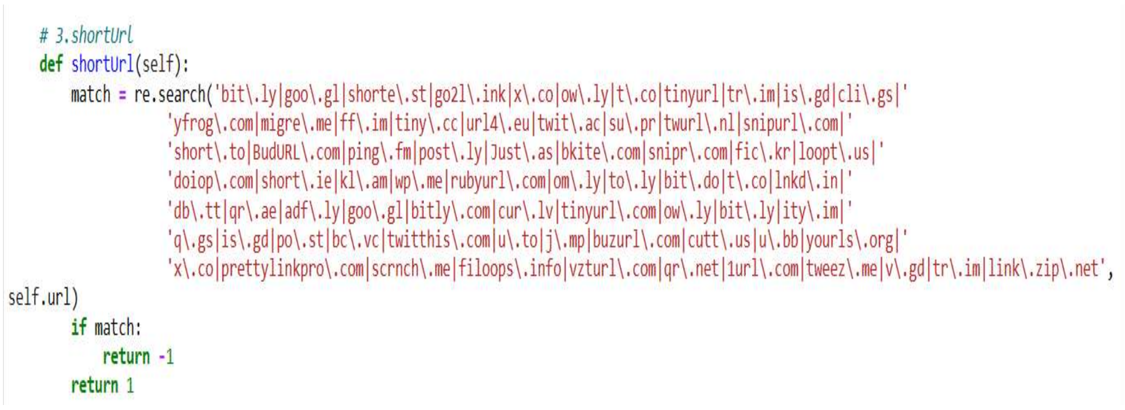
  - Long URL



```
# 2.longUrl
def longUrl(self):
    if len(self.url) < 54:
        return 1
    if len(self.url) >= 54 and len(self.url) <= 75:
        return 0
    return -1
```

Figure 3.3.5 Long URL Extraction

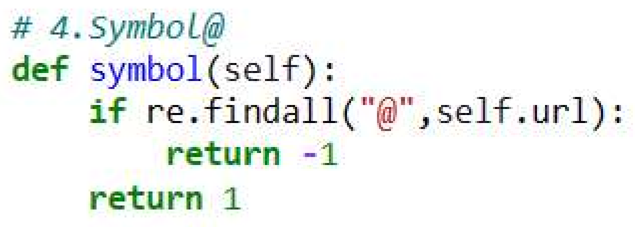
  - Short URL



```
    # 3.shortUrl
    def shortUrl(self):
        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',
self.url)
        if match:
            return -1
        return 1
```

Figure 3.3.6 Short URL Extraction

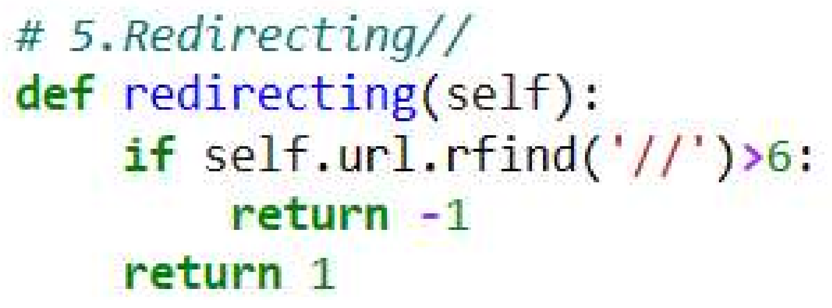
- Symbol@



```
# 4.Symbol@
def symbol(self):
    if re.findall("@",self.url):
        return -1
    return 1
```

Figure 3.3.7 @Symbol Extraction

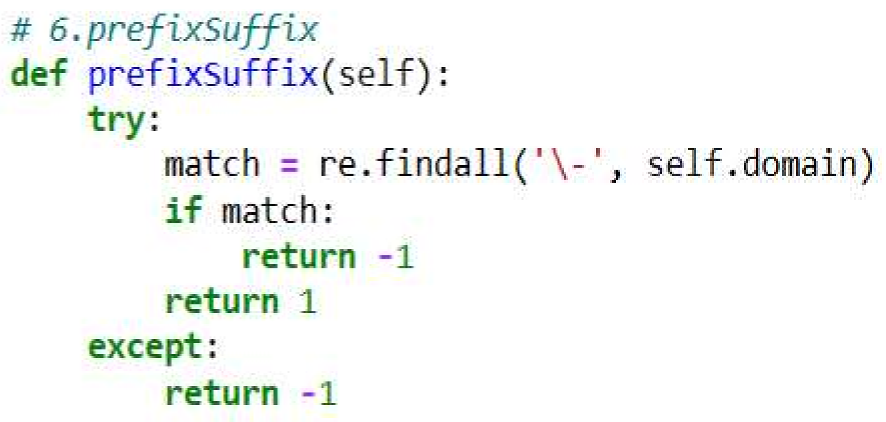
- Redirection “//” in URL



```
# 5.Redirecting//
def redirecting(self):
    if self.url.rfind('//')>6:
        return -1
    return 1
```

Figure 3.3.8 Redirection of (//) Symbol

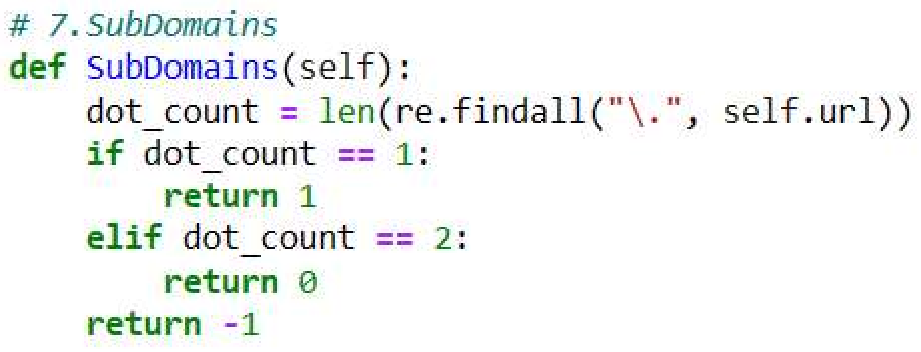
- Prefix Suffix



```
# 6.prefixSuffix
def prefixSuffix(self):
    try:
        match = re.findall('\-', self.domain)
        if match:
            return -1
        return 1
    except:
        return -1
```

Figure 3.3.9 Prefix and Suffix Extraction

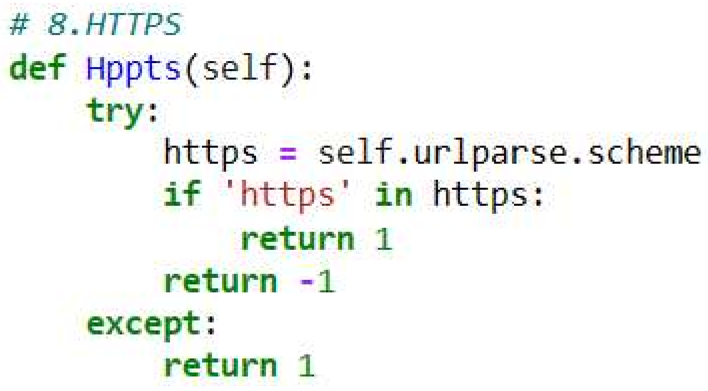
- Sub Domains



```
# 7.SubDomains
def SubDomains(self):
    dot_count = len(re.findall("\.", self.url))
    if dot_count == 1:
        return 1
    elif dot_count == 2:
        return 0
    return -1
```

Figure 3.3.10 Sub Domains

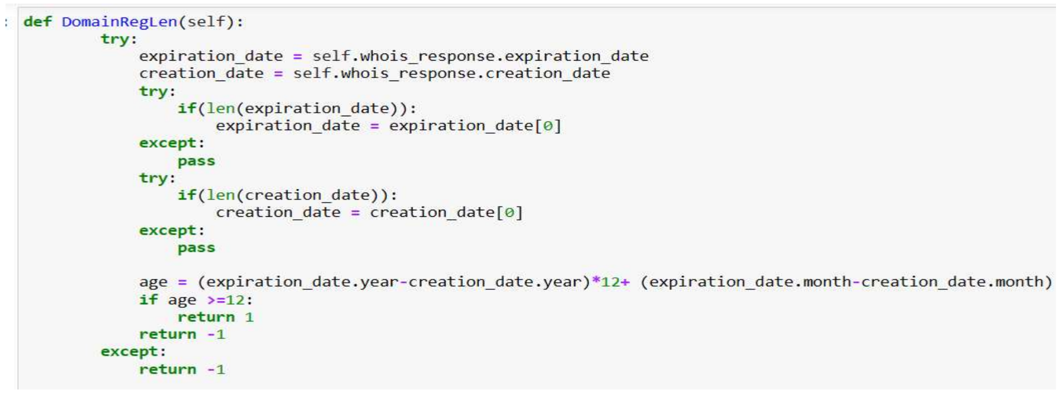
- https



```
# 8.HTTPS
def Hppts(self):
    try:
        https = self.urlparse.scheme
        if 'https' in https:
            return 1
        return -1
    except:
        return 1
```

Figure 3.3.11 Extraction of https

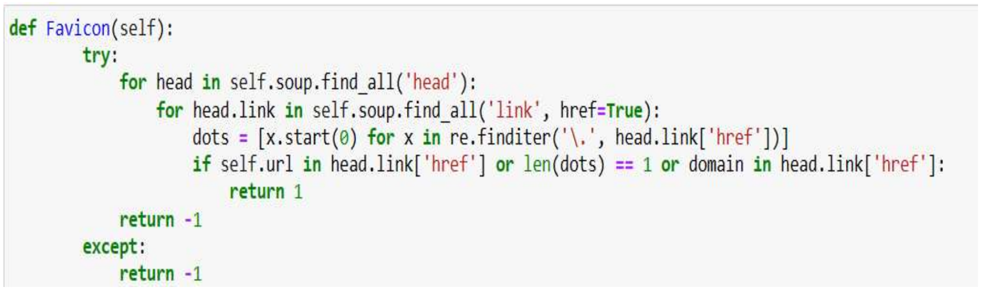
- Domain Length



```
def DomainRegLen(self):
        try:
            expiration_date = self.whois_response.expiration_date
            creation_date = self.whois_response.creation_date
            try:
                if(len(expiration_date)):
                    expiration_date = expiration_date[0]
            except:
                pass
            try:
                if(len(creation_date)):
                    creation_date = creation_date[0]
            except:
                pass

            age = (expiration_date.year-creation_date.year)*12+ (expiration_date.month-creation_date.month)
            if age >=12:
                return 1
            return -1
        except:
            return -1
```

Figure 3.3.12 Domain Length

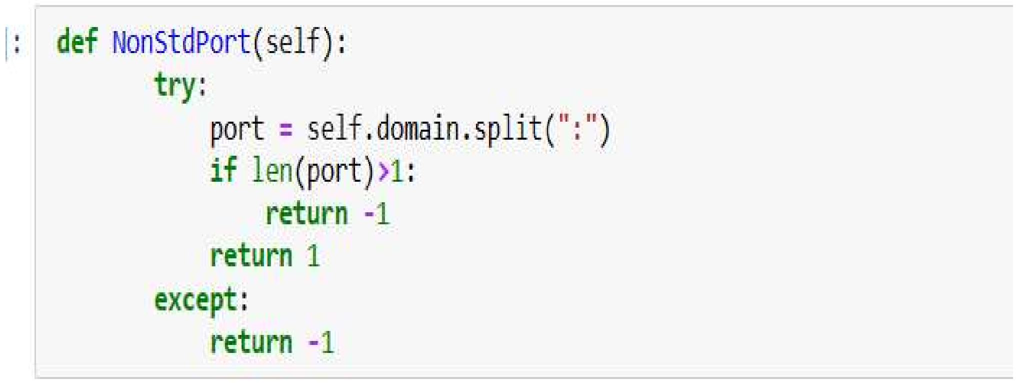
- Favicon



```
def Favicon(self):
        try:
            for head in self.soup.find_all('head'):
                for head.link in self.soup.find_all('link', href=True):
                    dots = [x.start(0) for x in re.finditer('\.', head.link['href'])]
                    if self.url in head.link['href'] or len(dots) == 1 or domain in head.link['href']:
                        return 1
            return -1
        except:
            return -1
```

Figure 3.3.13 Favicon

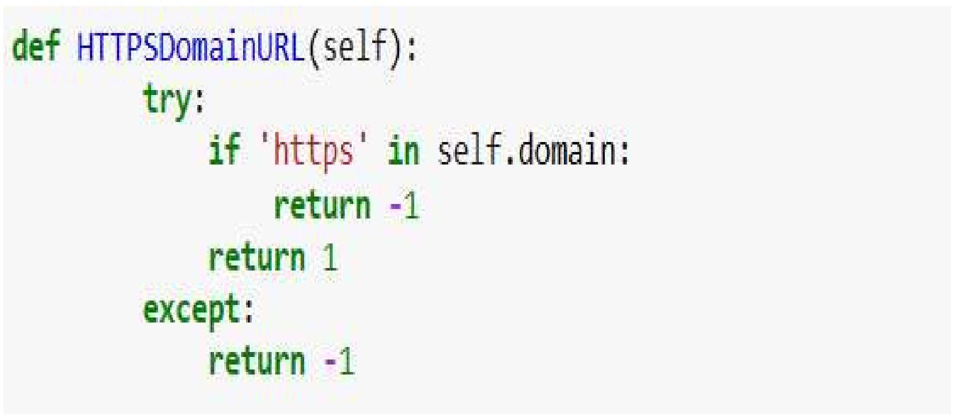
- NontdPort



```
def NonStdPort(self):
        try:
            port = self.domain.split(":")
            if len(port)>1:
                return -1
            return 1
        except:
            return -1
```

Figure 3.3.14 Std Port Extraction

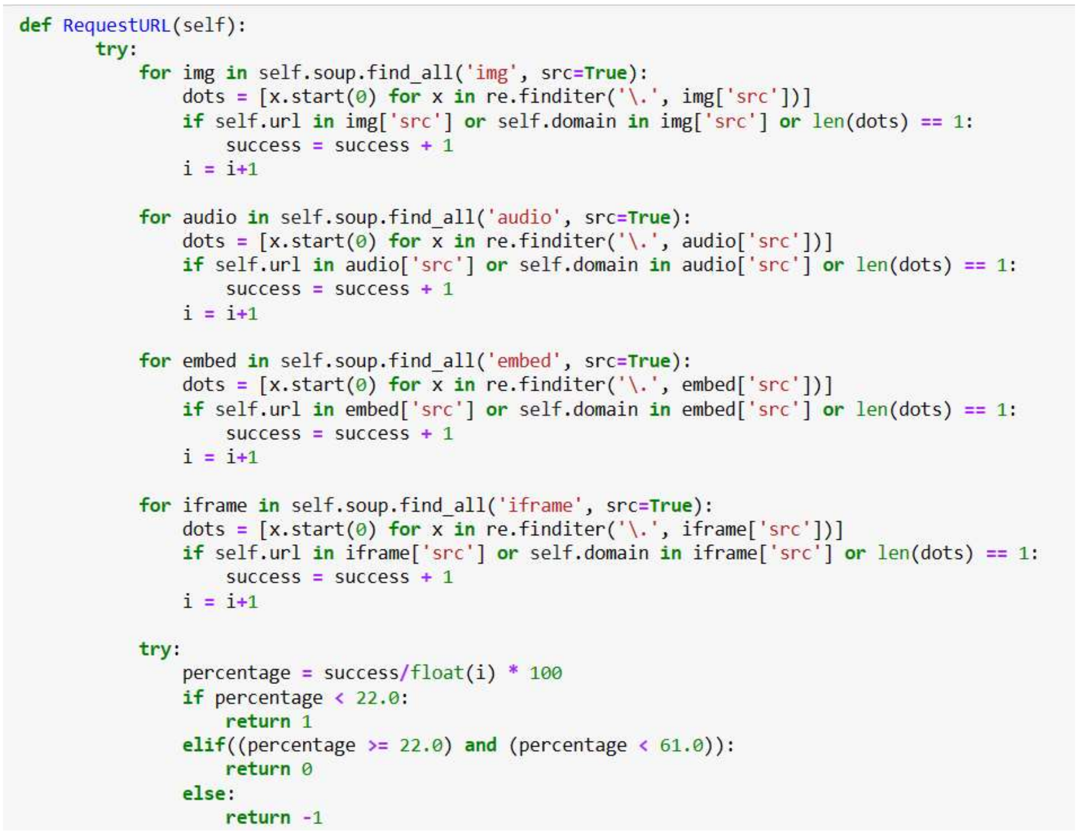
- HTTPS Domain URL



```
def HTTPSDomainURL(self):
        try:
            if 'https' in self.domain:
                return -1
            return 1
        except:
            return -1
```

Figure 3.3.15 Https Domain URL Extraction

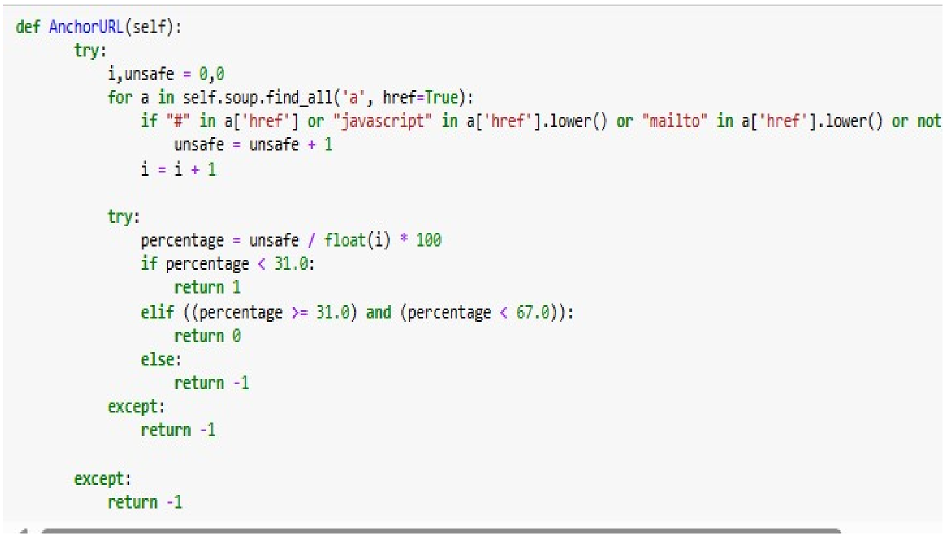
- Request URL



```python
def RequestURL(self):
       try:
           for img in self.soup.find_all('img', src=True):
               dots = [x.start(0) for x in re.finditer('\.', img['src'])]
               if self.url in img['src'] or self.domain in img['src'] or len(dots) == 1:
                   success = success + 1
               i = i+1

           for audio in self.soup.find_all('audio', src=True):
               dots = [x.start(0) for x in re.finditer('\.', audio['src'])]
               if self.url in audio['src'] or self.domain in audio['src'] or len(dots) == 1:
                   success = success + 1
               i = i+1

           for embed in self.soup.find_all('embed', src=True):
               dots = [x.start(0) for x in re.finditer('\.', embed['src'])]
               if self.url in embed['src'] or self.domain in embed['src'] or len(dots) == 1:
                   success = success + 1
               i = i+1

           for iframe in self.soup.find_all('iframe', src=True):
               dots = [x.start(0) for x in re.finditer('\.', iframe['src'])]
               if self.url in iframe['src'] or self.domain in iframe['src'] or len(dots) == 1:
                   success = success + 1
               i = i+1

           try:
               percentage = success/float(i) * 100
               if percentage < 22.0:
                   return 1
               elif((percentage >= 22.0) and (percentage < 61.0)):
                   return 0
               else:
                   return -1
```

Figure 3.3.16 Request URL

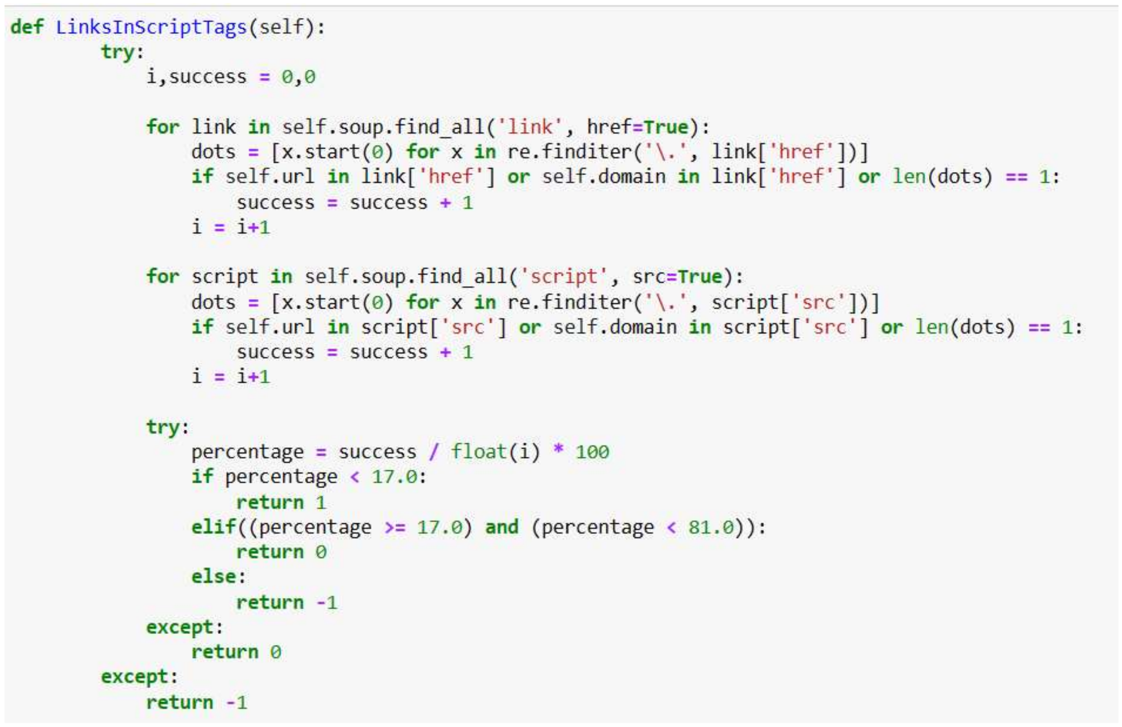
- Anchor URL



```python
def AnchorURL(self):
       try:
           i,unsafe = 0,0
           for a in self.soup.find_all('a', href=True):
               if "#" in a['href'] or "javascript" in a['href'].lower() or "mailto" in a['href'].lower() or not
                   unsafe = unsafe + 1
               i = i + 1

           try:
               percentage = unsafe / float(i) * 100
               if percentage < 31.0:
                   return 1
               elif ((percentage >= 31.0) and (percentage < 67.0)):
                   return 0
               else:
                   return -1
           except:
               return -1

       except:
           return -1
```

Figure 3.3.17 Anchor URL Extraction

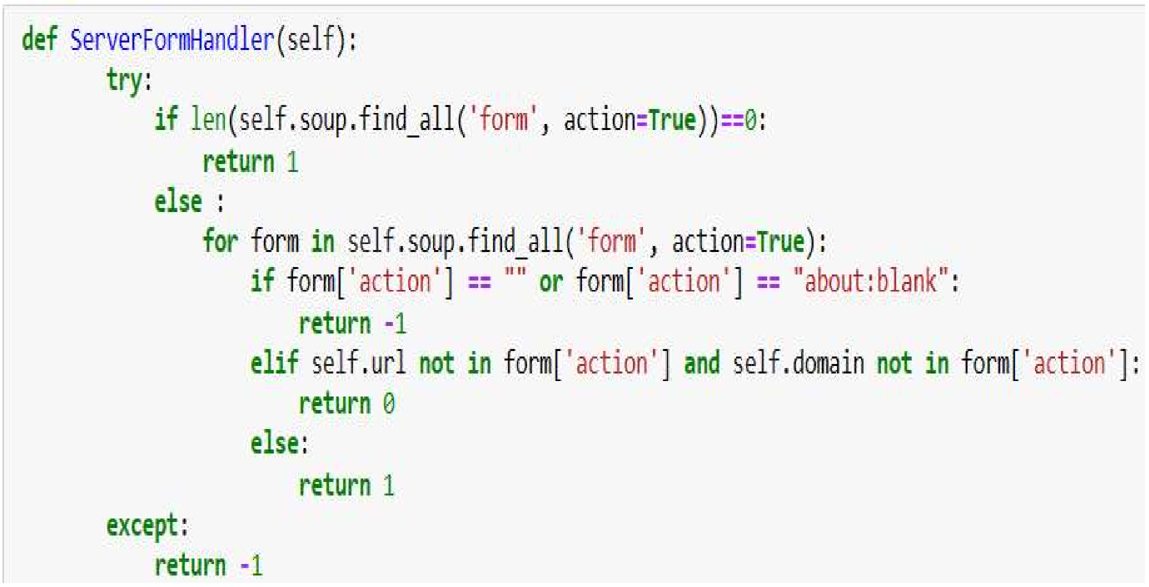
- Script Tags



```
def LinksInScriptTags(self):
        try:
            i,success = 0,0

            for link in self.soup.find_all('link', href=True):
                dots = [x.start(0) for x in re.finditer('\.', link['href'])]
                if self.url in link['href'] or self.domain in link['href'] or len(dots) == 1:
                    success = success + 1
                i = i+1

            for script in self.soup.find_all('script', src=True):
                dots = [x.start(0) for x in re.finditer('\.', script['src'])]
                if self.url in script['src'] or self.domain in script['src'] or len(dots) == 1:
                    success = success + 1
                i = i+1

            try:
                percentage = success / float(i) * 100
                if percentage < 17.0:
                    return 1
                elif((percentage >= 17.0) and (percentage < 81.0)):
                    return 0
                else:
                    return -1
            except:
                return 0
        except:
            return -1
```

Figure 3.3.18 Script Link Tags

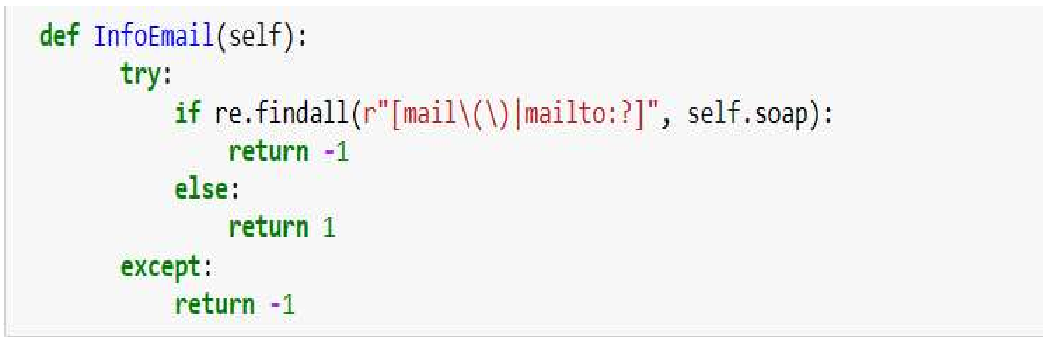
- Server Handler



```
def ServerFormHandler(self):
        try:
            if len(self.soup.find_all('form', action=True))==0:
                return 1
            else :
                for form in self.soup.find_all('form', action=True):
                    if form['action'] == "" or form['action'] == "about:blank":
                        return -1
                    elif self.url not in form['action'] and self.domain not in form['action']:
                        return 0
                    else:
                        return 1
        except:
            return -1
```

Figure 3.3.19 Server Handler Captioning

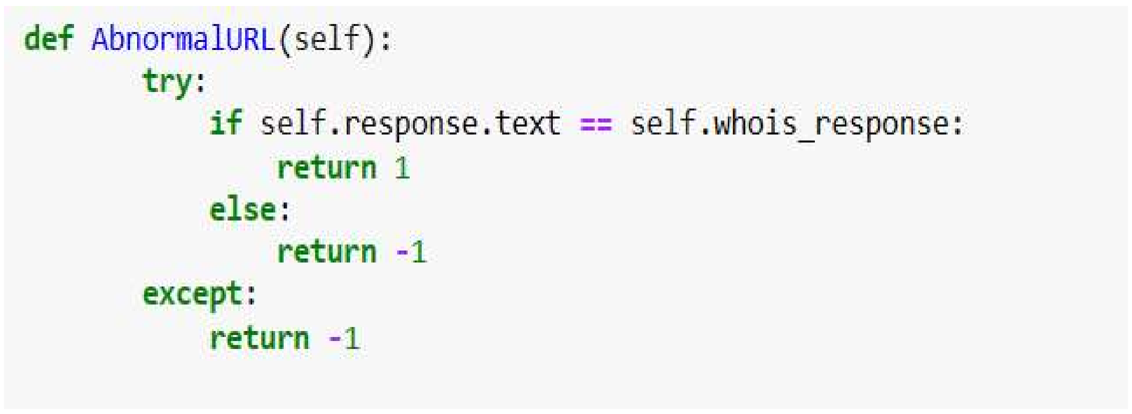
- Email Info



```
def InfoEmail(self):
        try:
            if re.findall(r"[mail\(\)|mailto:?]", self.soap):
                return -1
            else:
                return 1
        except:
            return -1
```

Figure 3.3.20 Email Information

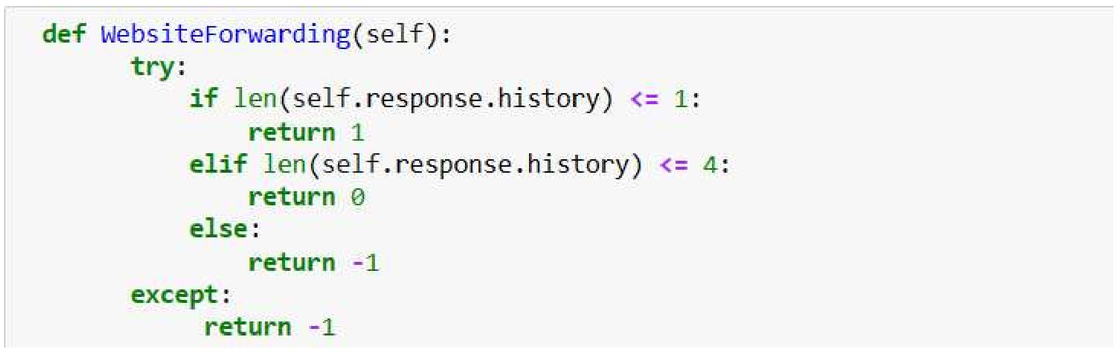
- Abnormal URL



```
def AbnormalURL(self):
        try:
            if self.response.text == self.whois_response:
                return 1
            else:
                return -1
        except:
            return -1
```

Figure 3.3.21 Abnormal URL Extraction

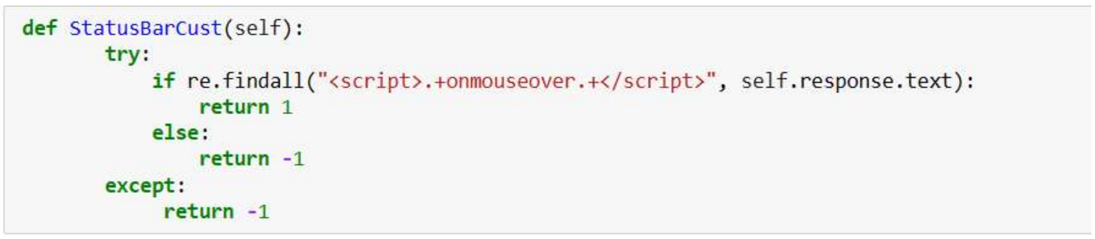
- Website Forwarding



```
def WebsiteForwarding(self):
        try:
            if len(self.response.history) <= 1:
                return 1
            elif len(self.response.history) <= 4:
                return 0
            else:
                return -1
        except:
             return -1
```

Figure 3.3.22 Website Forwarding Response

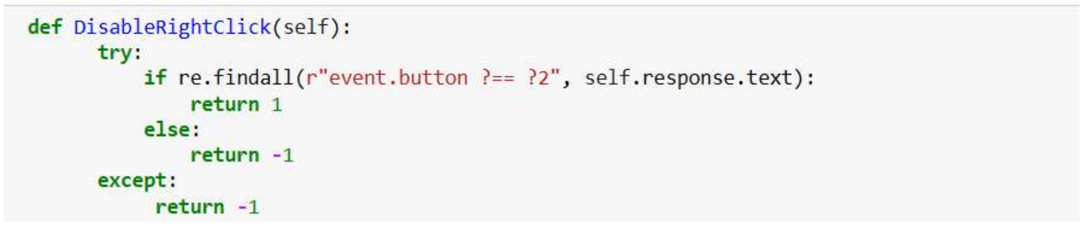
- Status



```
def StatusBarCust(self):
        try:
            if re.findall("<script>.+onmouseover.+</script>", self.response.text):
                return 1
            else:
                return -1
        except:
             return -1
```

Figure 3.3.23 Status bar finder

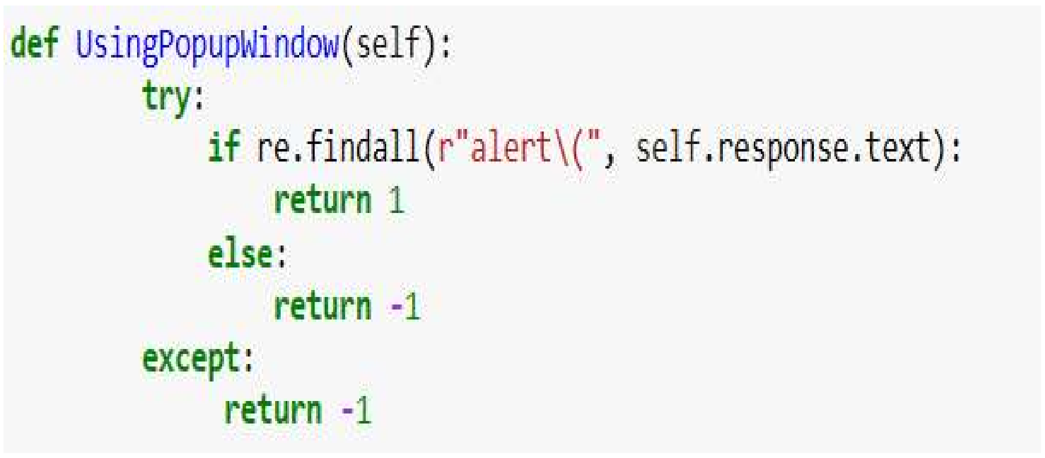
- Disable Right Click



```
def DisableRightClick(self):
        try:
            if re.findall(r"event.button ?== ?2", self.response.text):
                return 1
            else:
                return -1
        except:
             return -1
```

Figure 3.3.24 Disable Right Click

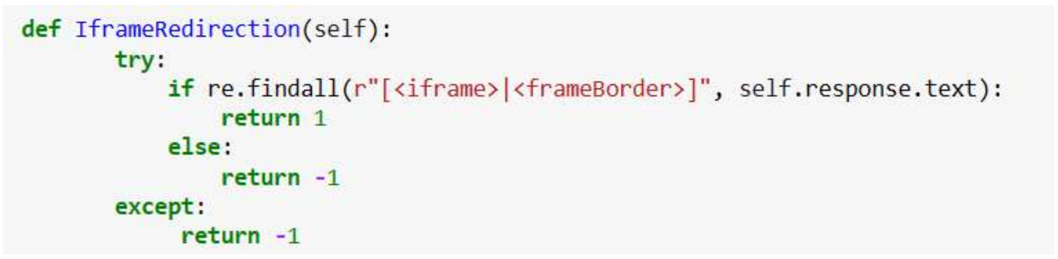
- Using Popup Window



```
def UsingPopupWindow(self):
        try:
            if re.findall(r"alert\(", self.response.text):
                return 1
            else:
                return -1
        except:
             return -1
```

Figure 3.3.25 Popping Window

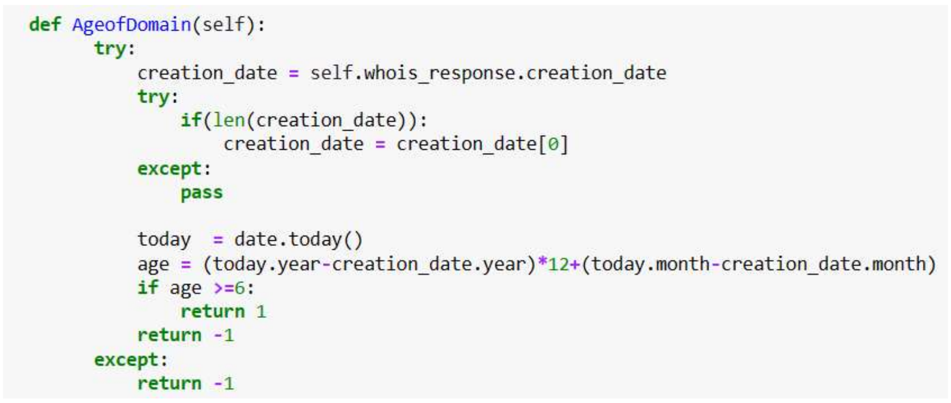
- Iframe Redirection



```python
def IframeRedirection(self):
        try:
            if re.findall(r"[<iframe>|<frameBorder>]", self.response.text):
                return 1
            else:
                return -1
        except:
             return -1
```

Figure 3.3.26 Redirection of iframe

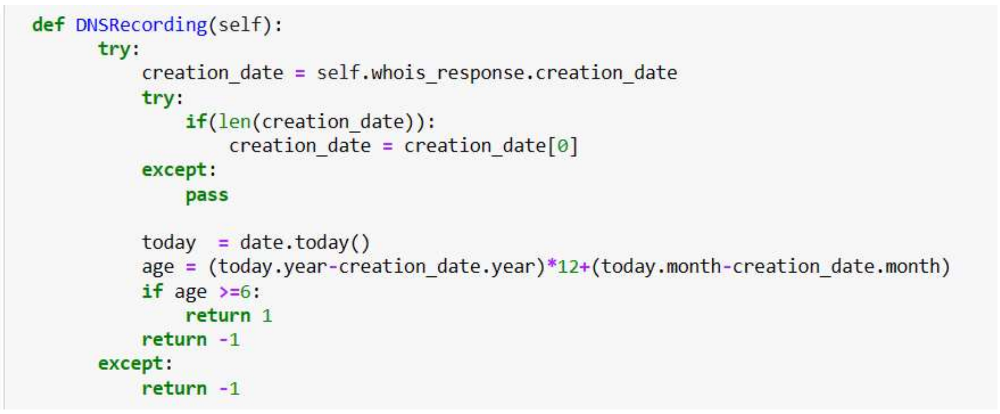
- Age of Domain



```python
def AgeofDomain(self):
        try:
            creation_date = self.whois_response.creation_date
            try:
                if(len(creation_date)):
                    creation_date = creation_date[0]
            except:
                pass

            today  = date.today()
            age = (today.year-creation_date.year)*12+(today.month-creation_date.month)
            if age >=6:
                return 1
            return -1
        except:
            return -1
```

Figure 3.3.27 Age Domain Handler

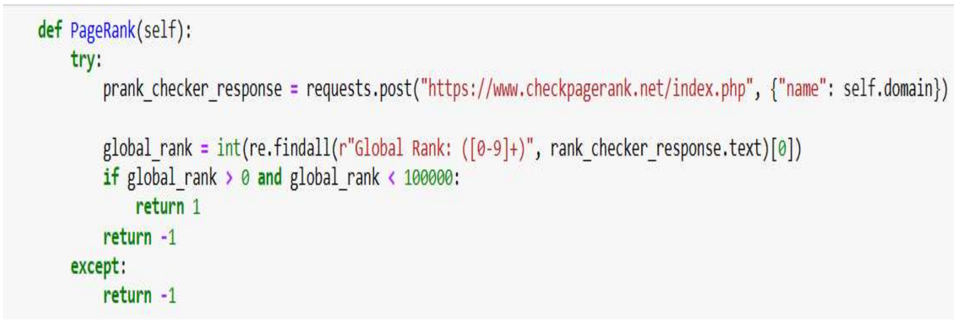
- DNS Recording



```python
def DNSRecording(self):
        try:
            creation_date = self.whois_response.creation_date
            try:
                if(len(creation_date)):
                    creation_date = creation_date[0]
            except:
                pass

            today  = date.today()
            age = (today.year-creation_date.year)*12+(today.month-creation_date.month)
            if age >=6:
                return 1
            return -1
        except:
            return -1
```

Figure 3.3.28 DNS Recording Handler

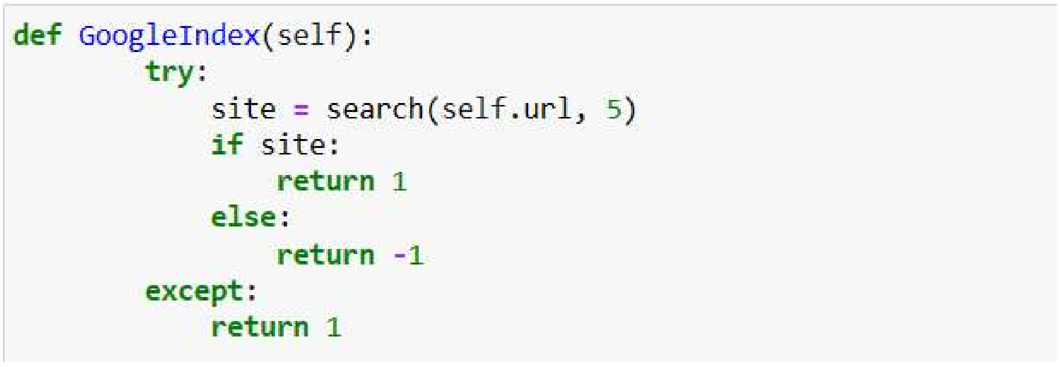
- Page Rank



```python
def PageRank(self):
    try:
        prank_checker_response = requests.post("https://www.checkpagerank.net/index.php", {"name": self.domain})

        global_rank = int(re.findall(r"Global Rank: ([0-9]+)", rank_checker_response.text)[0])
        if global_rank > 0 and global_rank < 100000:
            return 1
        return -1
    except:
        return -1
```

Figure 3.3.29 Page Rank Response Checker

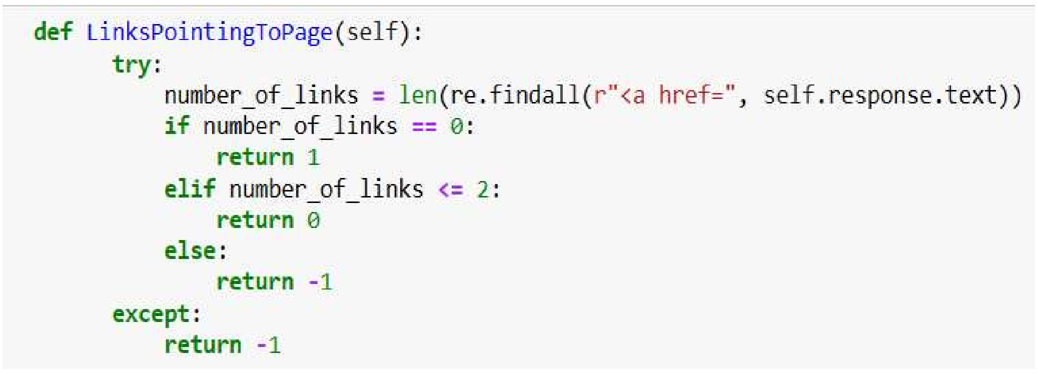
- Google Index



```python
def GoogleIndex(self):
        try:
            site = search(self.url, 5)
            if site:
                return 1
            else:
                return -1
        except:
            return 1
```

Figure 3.3.30 Google Index Handler

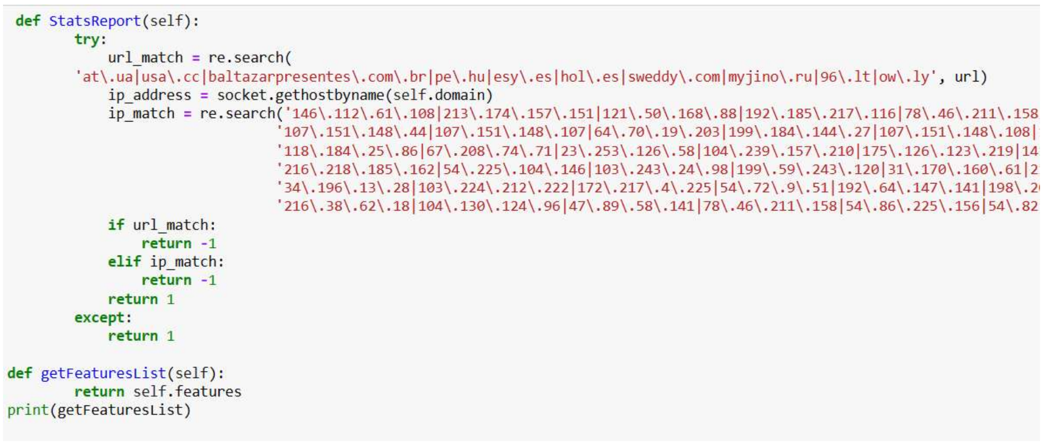
- Links Pointing To Page



```python
def LinksPointingToPage(self):
        try:
            number_of_links = len(re.findall(r"<a href=", self.response.text))
            if number_of_links == 0:
                return 1
            elif number_of_links <= 2:
                return 0
            else:
                return -1
        except:
            return -1
```

Figure 3.3.31 Number of Links Pointing

- Stats Report



```
def StatsReport(self):
        try:
            url_match = re.search(
        'at\.ua|usa\.cc|baltazarpresentes\.com\.br|pe\.hu|esy\.es|hol\.es|sweddy\.com|myjino\.ru|96\.lt|ow\.ly', url)
            ip_address = socket.gethostbyname(self.domain)
            ip_match = re.search('146\.112\.61\.108|213\.174\.157\.151|121\.50\.168\.88|192\.185\.217\.116|78\.46\.211\.158
                                 '107\.151\.148\.44|107\.151\.148\.107|64\.70\.19\.203|199\.184\.144\.27|107\.151\.148\.108|
                                 '118\.184\.25\.86|67\.208\.74\.71|23\.253\.126\.58|104\.239\.157\.210|175\.126\.123\.219|14
                                 '216\.218\.185\.162|54\.225\.104\.146|103\.243\.24\.98|199\.59\.243\.120|31\.170\.160\.61|2
                                 '34\.196\.13\.28|103\.224\.212\.222|172\.217\.4\.225|54\.72\.9\.51|192\.64\.147\.141|198\.2
                                 '216\.38\.62\.18|104\.130\.124\.96|47\.89\.58\.141|78\.46\.211\.158|54\.86\.225\.156|54\.82
            if url_match:
                return -1
            elif ip_match:
                return -1
            return 1
        except:
            return 1

def getFeaturesList(self):
        return self.features
print(getFeaturesList)
```

Figure 3.3.32 Matching Stats Report

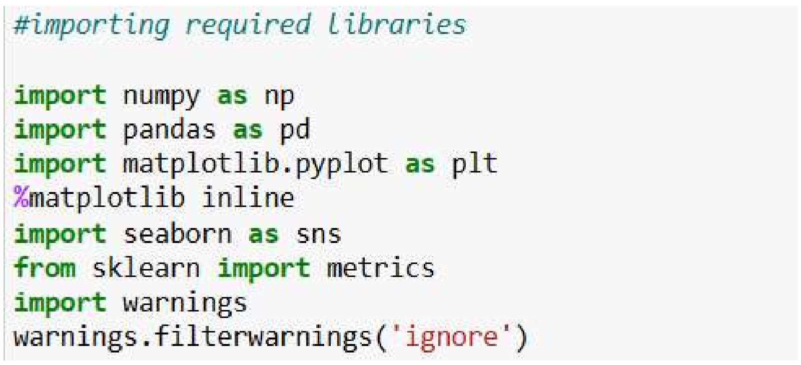
3.4 Development of the Phishing Model

In the development of our URL classification model, we conducted a thorough exploration of machine learning algorithms and deep learning algorithms, with a focus on Random Forest, Decision Tree, Support Vector Machine (SVM), Gradient Boost, Logistic Regression, Naive Bayes, K-nearest Neighbor and multilayer Perceptron. Through rigorous evaluation, Gradient Boost Classifier emerged as the most effective, achieving a notable accuracy score of 97.4%. The feature extraction phase involved deriving 30 relevant features from address bar, domain-based, and HTML/JavaScript-based attributes. This diverse dataset, encompassing both legitimate and phishing URLs, facilitated robust model training. Moving forward, the model's future development can include refining its capabilities with advanced machine learning techniques and continuous dataset updates to adapt to emerging online threats. The Flask web application (app.py) can be enhanced by incorporating features such as a more interactive user interface, detailed result insights, and real-time threat updates. Additionally, deploying the application on cloud services like AWS and using Docker can broaden its accessibility, allowing for global user interaction. Docker helps in ensuring that application behaves consistently across different environments and can be easily deployed in productions. Further improvements may include user authentication, features for managing and tracking URL classifications, and heightened security measures.

Chapter 04: PERFORMANCE ANALYSIS

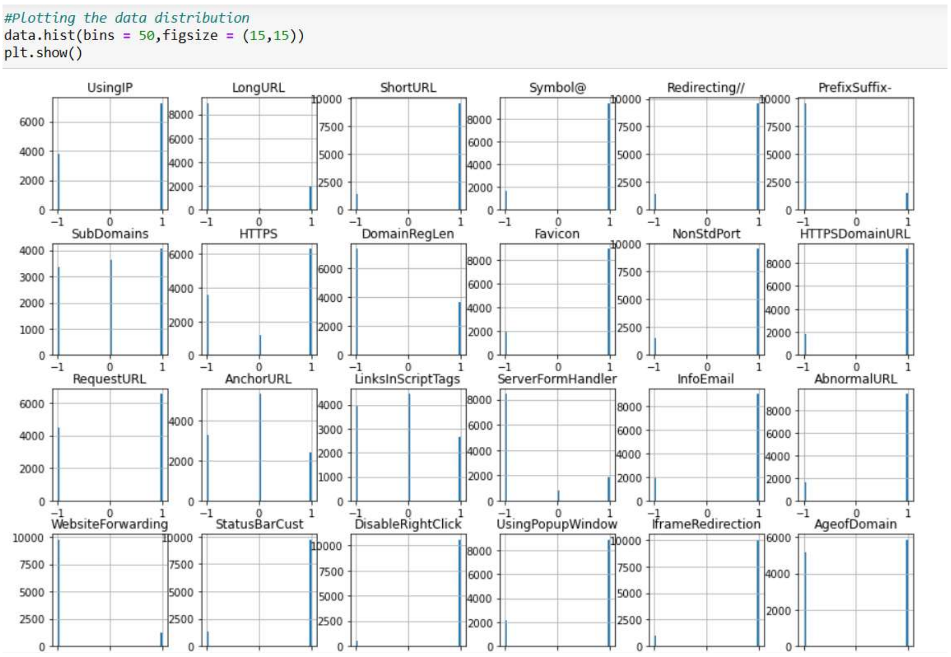
We have carried out a number of training implementations for this project, and we have predicted thelegitimacy of phishing websites. We have used some of the supervised algorithms.

- To start, we imported a number of libraries, including matplotlib, pandas, numpy, random forestclassifier, decision tree classifier, and many more.
- We have examined and pre-process the dataset and have plot the data distribution.



```
#importing required libraries

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from sklearn import metrics
import warnings
warnings.filterwarnings('ignore')
```

Figure 4.1.1 Importing Libraries



```
#Plotting the data distribution
data.hist(bins = 50,figsize = (15,15))
plt.show()
```

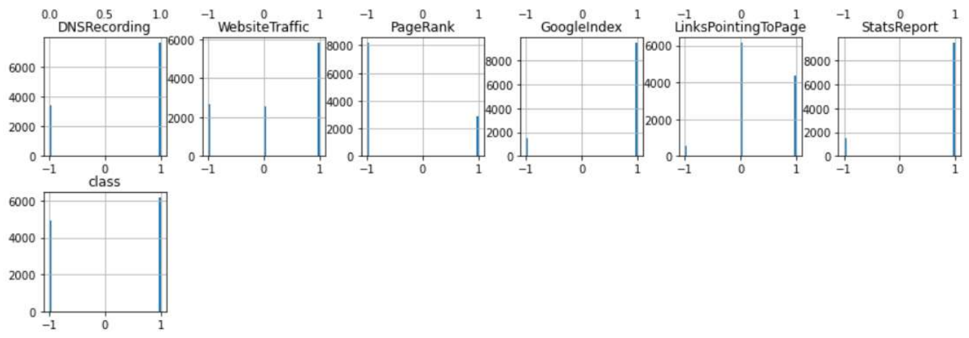
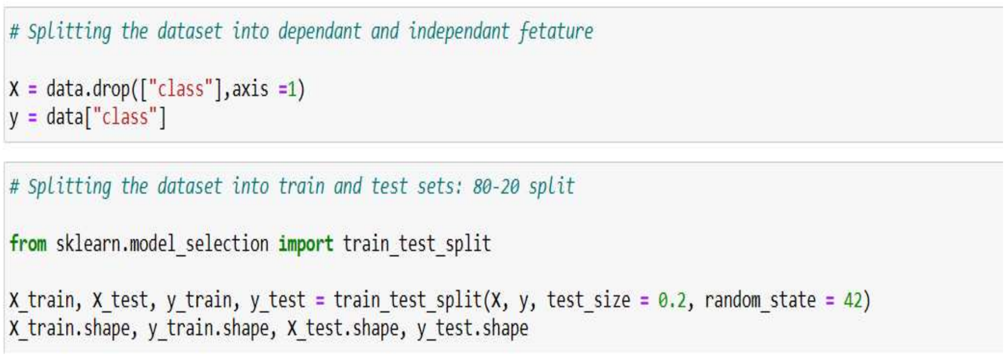


Figure 4.1.2 Data Distribution

- After that we have split the dataset into test and train.

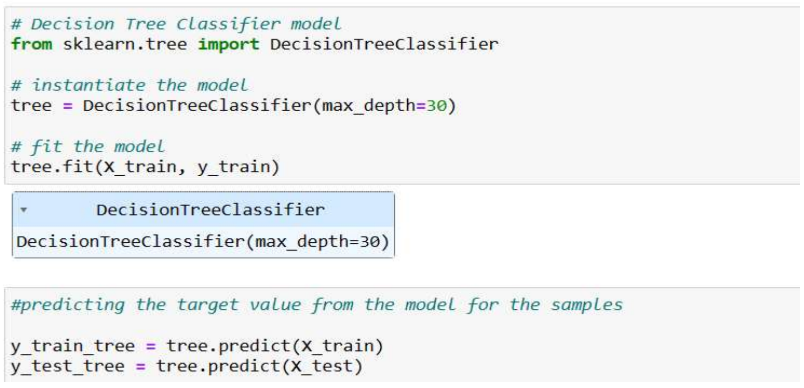


```
# Splitting the dataset into dependant and independant fetature

X = data.drop(["class"],axis =1)
y = data["class"]
```

```
# Splitting the dataset into train and test sets: 80-20 split

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 42)
X_train.shape, y_train.shape, X_test.shape, y_test.shape
```

Figure 4.1.3 Splitting Dataset

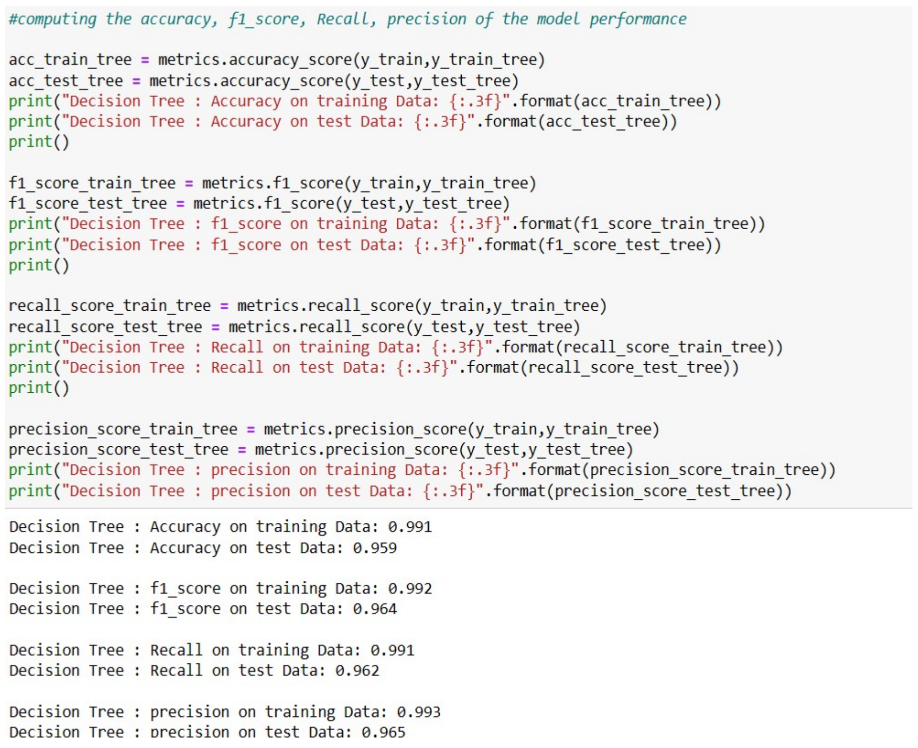
- Now we have use Decision Tree Classifier, checking the feature importance in the model.



```
# Decision Tree Classifier model
from sklearn.tree import DecisionTreeClassifier

# instantiate the model
tree = DecisionTreeClassifier(max_depth=30)

# fit the model
tree.fit(X_train, y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(max_depth=30)
```

```
#predicting the target value from the model for the samples

y_train_tree = tree.predict(X_train)
y_test_tree = tree.predict(X_test)
```



```
#computing the accuracy, f1_score, Recall, precision of the model performance

acc_train_tree = metrics.accuracy_score(y_train,y_train_tree)
acc_test_tree = metrics.accuracy_score(y_test,y_test_tree)
print("Decision Tree : Accuracy on training Data: {:.3f}".format(acc_train_tree))
print("Decision Tree : Accuracy on test Data: {:.3f}".format(acc_test_tree))
print()

f1_score_train_tree = metrics.f1_score(y_train,y_train_tree)
f1_score_test_tree = metrics.f1_score(y_test,y_test_tree)
print("Decision Tree : f1_score on training Data: {:.3f}".format(f1_score_train_tree))
print("Decision Tree : f1_score on test Data: {:.3f}".format(f1_score_test_tree))
print()

recall_score_train_tree = metrics.recall_score(y_train,y_train_tree)
recall_score_test_tree = metrics.recall_score(y_test,y_test_tree)
print("Decision Tree : Recall on training Data: {:.3f}".format(recall_score_train_tree))
print("Decision Tree : Recall on test Data: {:.3f}".format(recall_score_test_tree))
print()

precision_score_train_tree = metrics.precision_score(y_train,y_train_tree)
precision_score_test_tree = metrics.precision_score(y_test,y_test_tree)
print("Decision Tree : precision on training Data: {:.3f}".format(precision_score_train_tree))
print("Decision Tree : precision on test Data: {:.3f}".format(precision_score_test_tree))
```

```
Decision Tree : Accuracy on training Data: 0.991
Decision Tree : Accuracy on test Data: 0.959

Decision Tree : f1_score on training Data: 0.992
Decision Tree : f1_score on test Data: 0.964

Decision Tree : Recall on training Data: 0.991
Decision Tree : Recall on test Data: 0.962

Decision Tree : precision on training Data: 0.993
Decision Tree : precision on test Data: 0.965
```

Figure 4.1.4 Decision Tree Accuracy

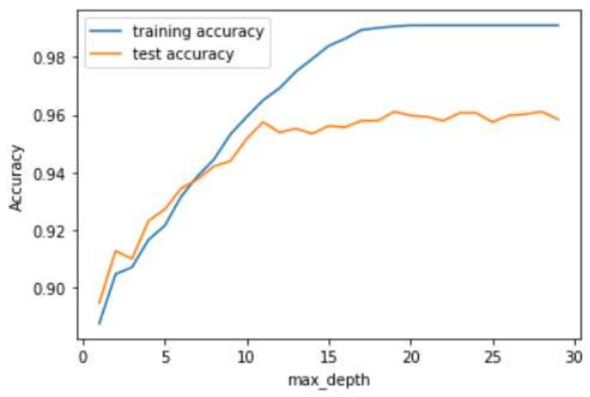
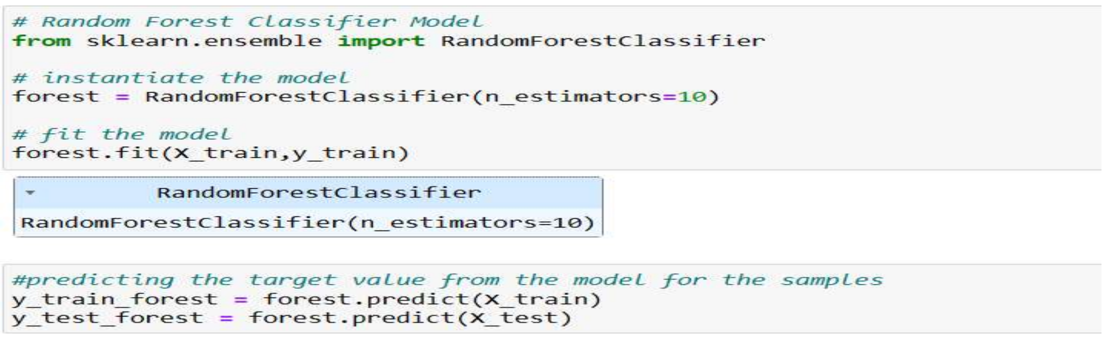


Figure 4.1.5 Accuracy vs Depth

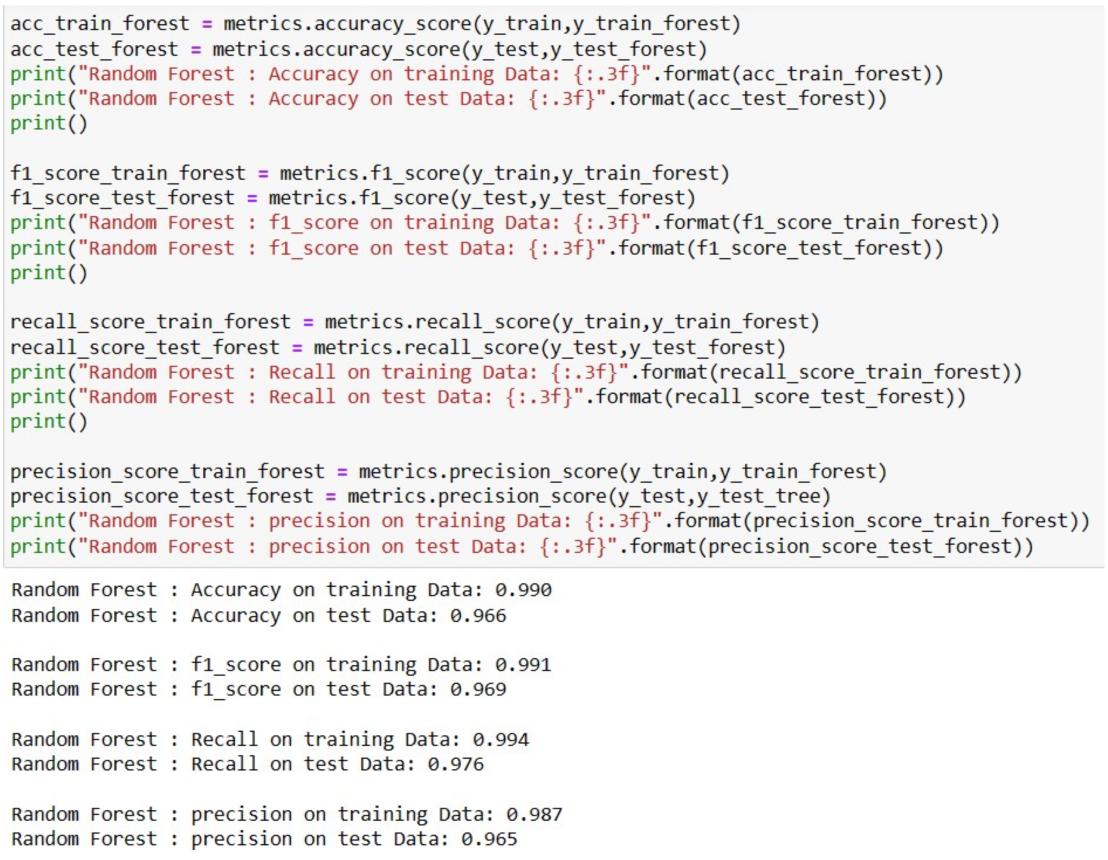
- Now we have use the random forest classifier, check the feature importance of the model.



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

# instantiate the model
forest = RandomForestClassifier(n_estimators=10)

# fit the model
forest.fit(X_train,y_train)
```

```
▾            RandomForestClassifier
RandomForestClassifier(n_estimators=10)
```

```
#predicting the target value from the model for the samples
y_train_forest = forest.predict(X_train)
y_test_forest = forest.predict(X_test)
```



```python
acc_train_forest = metrics.accuracy_score(y_train,y_train_forest)
acc_test_forest = metrics.accuracy_score(y_test,y_test_forest)
print("Random Forest : Accuracy on training Data: {:.3f}".format(acc_train_forest))
print("Random Forest : Accuracy on test Data: {:.3f}".format(acc_test_forest))
print()

f1_score_train_forest = metrics.f1_score(y_train,y_train_forest)
f1_score_test_forest = metrics.f1_score(y_test,y_test_forest)
print("Random Forest : f1_score on training Data: {:.3f}".format(f1_score_train_forest))
print("Random Forest : f1_score on test Data: {:.3f}".format(f1_score_test_forest))
print()

recall_score_train_forest = metrics.recall_score(y_train,y_train_forest)
recall_score_test_forest = metrics.recall_score(y_test,y_test_forest)
print("Random Forest : Recall on training Data: {:.3f}".format(recall_score_train_forest))
print("Random Forest : Recall on test Data: {:.3f}".format(recall_score_test_forest))
print()

precision_score_train_forest = metrics.precision_score(y_train,y_train_forest)
precision_score_test_forest = metrics.precision_score(y_test,y_test_tree)
print("Random Forest : precision on training Data: {:.3f}".format(precision_score_train_forest))
print("Random Forest : precision on test Data: {:.3f}".format(precision_score_test_forest))
```

```
Random Forest : Accuracy on training Data: 0.990
Random Forest : Accuracy on test Data: 0.966

Random Forest : f1_score on training Data: 0.991
Random Forest : f1_score on test Data: 0.969

Random Forest : Recall on training Data: 0.994
Random Forest : Recall on test Data: 0.976

Random Forest : precision on training Data: 0.987
Random Forest : precision on test Data: 0.965
```

Figure 4.1.6 Random Forest

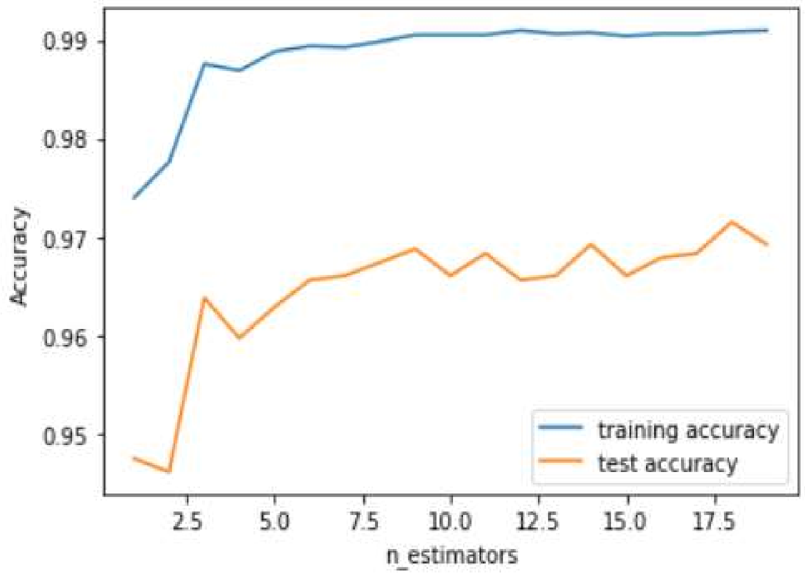
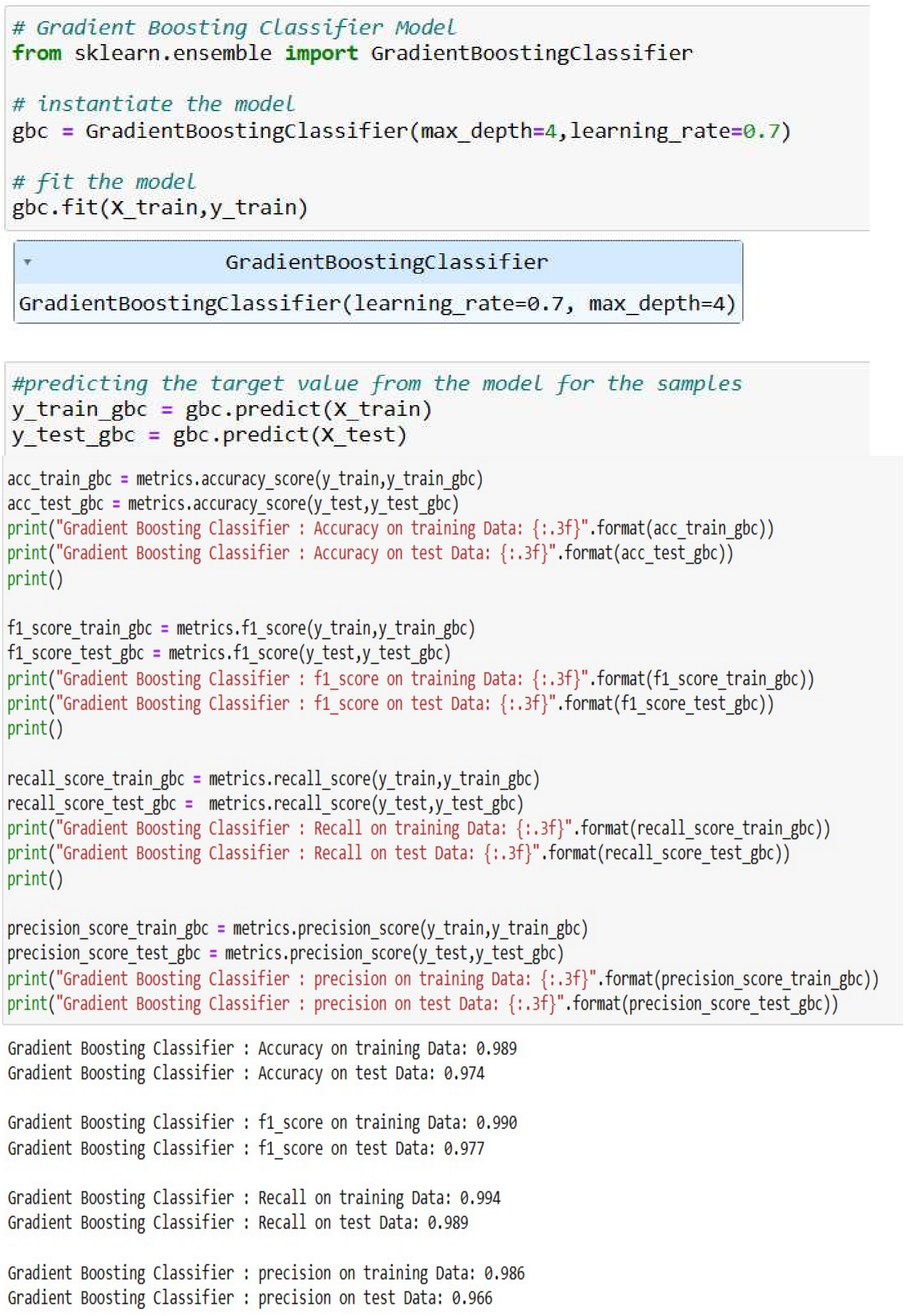


Figure 4.1.7 Training Accuracy vs Test Accuracy

- Now we have used Gradient Boost Classifier.



```
# Gradient Boosting Classifier Model
from sklearn.ensemble import GradientBoostingClassifier

# instantiate the model
gbc = GradientBoostingClassifier(max_depth=4,learning_rate=0.7)

# fit the model
gbc.fit(X_train,y_train)
```

```
GradientBoostingClassifier
GradientBoostingClassifier(learning_rate=0.7, max_depth=4)
```

```
#predicting the target value from the model for the samples
y_train_gbc = gbc.predict(X_train)
y_test_gbc = gbc.predict(X_test)
```

```
acc_train_gbc = metrics.accuracy_score(y_train,y_train_gbc)
acc_test_gbc = metrics.accuracy_score(y_test,y_test_gbc)
print("Gradient Boosting Classifier : Accuracy on training Data: {:.3f}".format(acc_train_gbc))
print("Gradient Boosting Classifier : Accuracy on test Data: {:.3f}".format(acc_test_gbc))
print()

f1_score_train_gbc = metrics.f1_score(y_train,y_train_gbc)
f1_score_test_gbc = metrics.f1_score(y_test,y_test_gbc)
print("Gradient Boosting Classifier : f1_score on training Data: {:.3f}".format(f1_score_train_gbc))
print("Gradient Boosting Classifier : f1_score on test Data: {:.3f}".format(f1_score_test_gbc))
print()

recall_score_train_gbc = metrics.recall_score(y_train,y_train_gbc)
recall_score_test_gbc =  metrics.recall_score(y_test,y_test_gbc)
print("Gradient Boosting Classifier : Recall on training Data: {:.3f}".format(recall_score_train_gbc))
print("Gradient Boosting Classifier : Recall on test Data: {:.3f}".format(recall_score_test_gbc))
print()

precision_score_train_gbc = metrics.precision_score(y_train,y_train_gbc)
precision_score_test_gbc = metrics.precision_score(y_test,y_test_gbc)
print("Gradient Boosting Classifier : precision on training Data: {:.3f}".format(precision_score_train_gbc))
print("Gradient Boosting Classifier : precision on test Data: {:.3f}".format(precision_score_test_gbc))
```

```
Gradient Boosting Classifier : Accuracy on training Data: 0.989
Gradient Boosting Classifier : Accuracy on test Data: 0.974

Gradient Boosting Classifier : f1_score on training Data: 0.990
Gradient Boosting Classifier : f1_score on test Data: 0.977

Gradient Boosting Classifier : Recall on training Data: 0.994
Gradient Boosting Classifier : Recall on test Data: 0.989

Gradient Boosting Classifier : precision on training Data: 0.986
Gradient Boosting Classifier : precision on test Data: 0.966
```

Figure 4.1.8 Gradient Boost Accuracy

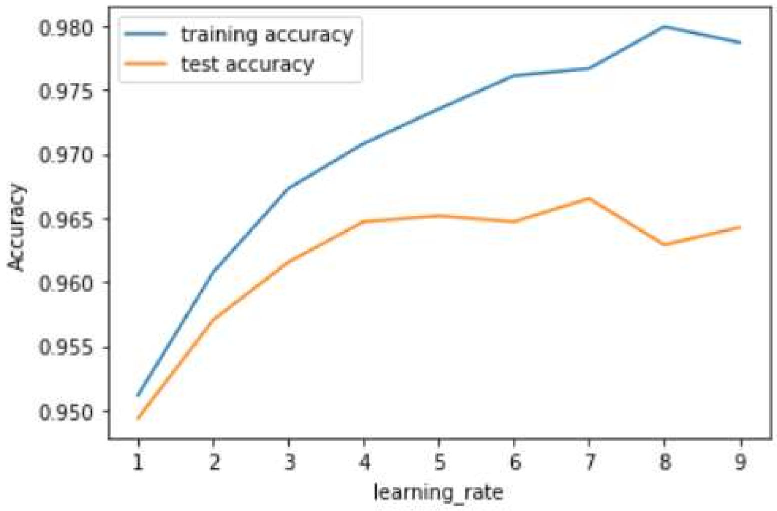
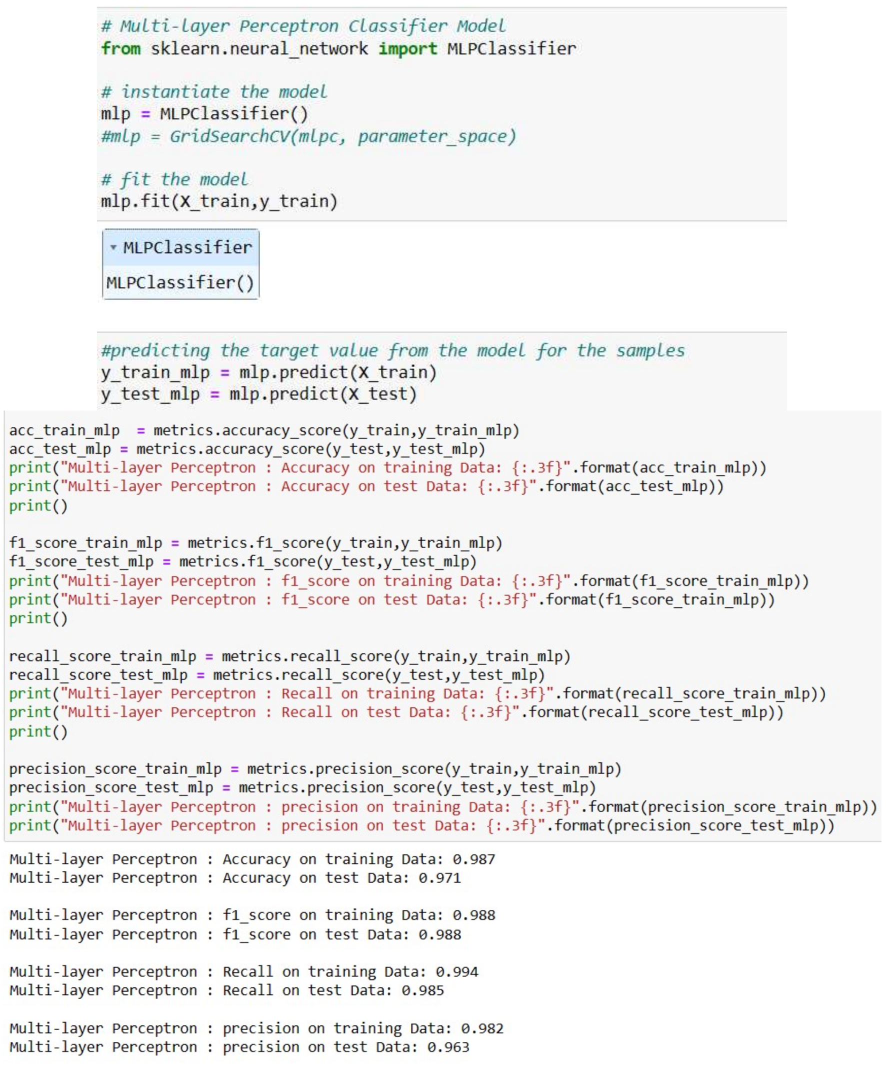


Figure 4.1.9 Training Accuracy vs Test Accuracy

- Now we have used Multilayer Perceptron



```
# Multi-layer Perceptron Classifier Model
from sklearn.neural_network import MLPClassifier

# instantiate the model
mlp = MLPClassifier()
#mlp = GridSearchCV(mlpc, parameter_space)

# fit the model
mlp.fit(X_train,y_train)
```

```
▾ MLPClassifier
MLPClassifier()
```

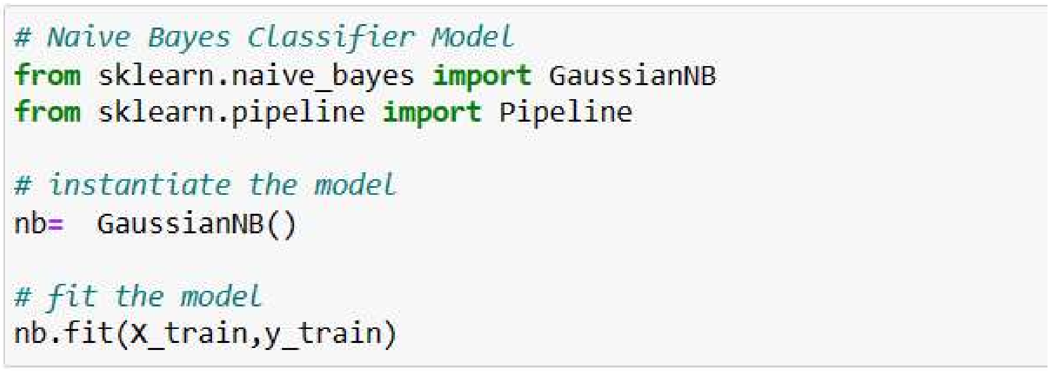
```
#predicting the target value from the model for the samples
y_train_mlp = mlp.predict(X_train)
y_test_mlp = mlp.predict(X_test)
```

```
acc_train_mlp  = metrics.accuracy_score(y_train,y_train_mlp)
acc_test_mlp = metrics.accuracy_score(y_test,y_test_mlp)
print("Multi-layer Perceptron : Accuracy on training Data: {:.3f}".format(acc_train_mlp))
print("Multi-layer Perceptron : Accuracy on test Data: {:.3f}".format(acc_test_mlp))
print()

f1_score_train_mlp = metrics.f1_score(y_train,y_train_mlp)
f1_score_test_mlp = metrics.f1_score(y_test,y_test_mlp)
print("Multi-layer Perceptron : f1_score on training Data: {:.3f}".format(f1_score_train_mlp))
print("Multi-layer Perceptron : f1_score on test Data: {:.3f}".format(f1_score_train_mlp))
print()

recall_score_train_mlp = metrics.recall_score(y_train,y_train_mlp)
recall_score_test_mlp = metrics.recall_score(y_test,y_test_mlp)
print("Multi-layer Perceptron : Recall on training Data: {:.3f}".format(recall_score_train_mlp))
print("Multi-layer Perceptron : Recall on test Data: {:.3f}".format(recall_score_test_mlp))
print()

precision_score_train_mlp = metrics.precision_score(y_train,y_train_mlp)
precision_score_test_mlp = metrics.precision_score(y_test,y_test_mlp)
print("Multi-layer Perceptron : precision on training Data: {:.3f}".format(precision_score_train_mlp))
print("Multi-layer Perceptron : precision on test Data: {:.3f}".format(precision_score_test_mlp))
```

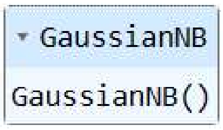
```
Multi-layer Perceptron : Accuracy on training Data: 0.987
Multi-layer Perceptron : Accuracy on test Data: 0.971

Multi-layer Perceptron : f1_score on training Data: 0.988
Multi-layer Perceptron : f1_score on test Data: 0.988

Multi-layer Perceptron : Recall on training Data: 0.994
Multi-layer Perceptron : Recall on test Data: 0.985

Multi-layer Perceptron : precision on training Data: 0.982
Multi-layer Perceptron : precision on test Data: 0.963
```

Figure 4.1.10 Multi-Layer Accuracy

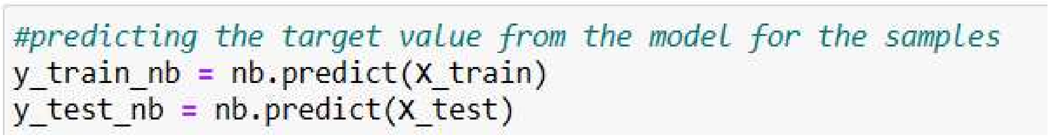
- Now we have used Naive Bayes Classifier.



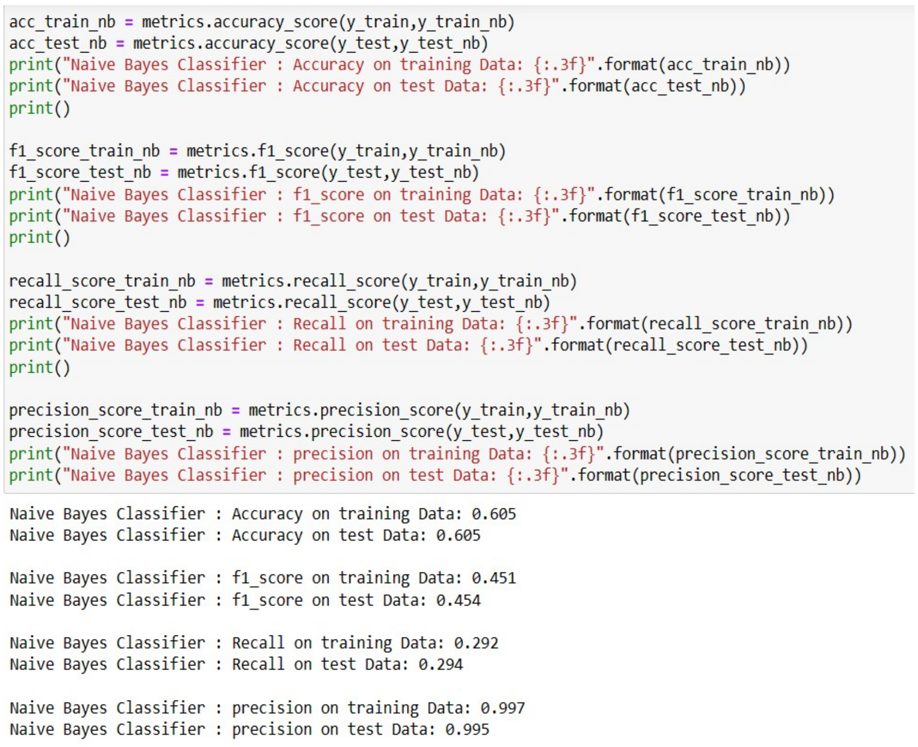
```
# Naive Bayes Classifier Model
from sklearn.naive_bayes import GaussianNB
from sklearn.pipeline import Pipeline

# instantiate the model
nb=  GaussianNB()

# fit the model
nb.fit(X_train,y_train)
```



```
▾ GaussianNB
GaussianNB()
```



```
#predicting the target value from the model for the samples
y_train_nb = nb.predict(X_train)
y_test_nb = nb.predict(X_test)
```

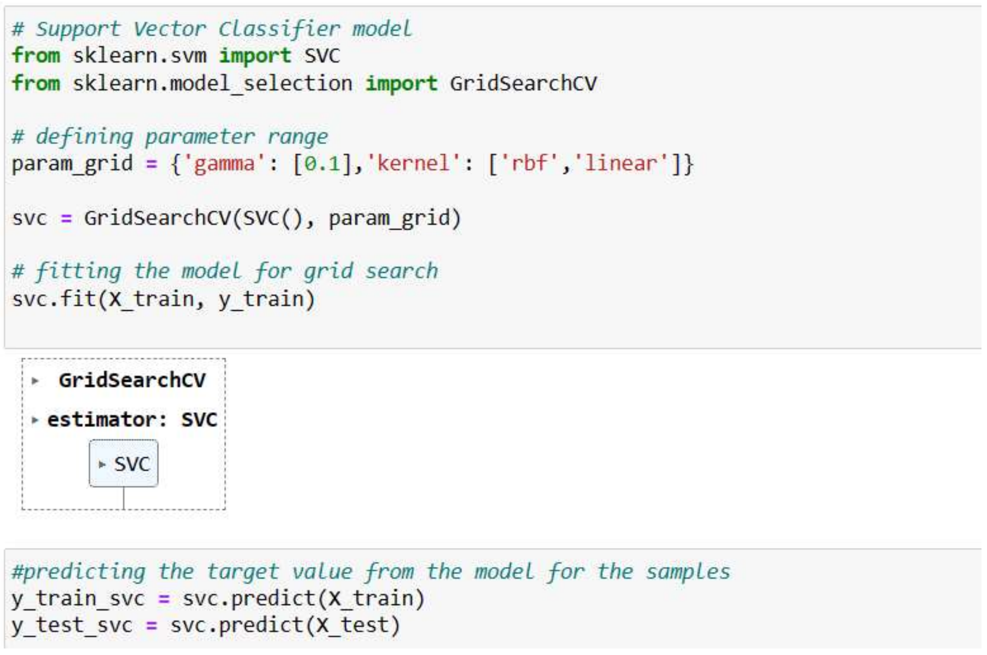


```
acc_train_nb = metrics.accuracy_score(y_train,y_train_nb)
acc_test_nb = metrics.accuracy_score(y_test,y_test_nb)
print("Naive Bayes Classifier : Accuracy on training Data: {:.3f}".format(acc_train_nb))
print("Naive Bayes Classifier : Accuracy on test Data: {:.3f}".format(acc_test_nb))
print()

f1_score_train_nb = metrics.f1_score(y_train,y_train_nb)
f1_score_test_nb = metrics.f1_score(y_test,y_test_nb)
print("Naive Bayes Classifier : f1_score on training Data: {:.3f}".format(f1_score_train_nb))
print("Naive Bayes Classifier : f1_score on test Data: {:.3f}".format(f1_score_test_nb))
print()

recall_score_train_nb = metrics.recall_score(y_train,y_train_nb)
recall_score_test_nb = metrics.recall_score(y_test,y_test_nb)
print("Naive Bayes Classifier : Recall on training Data: {:.3f}".format(recall_score_train_nb))
print("Naive Bayes Classifier : Recall on test Data: {:.3f}".format(recall_score_test_nb))
print()

precision_score_train_nb = metrics.precision_score(y_train,y_train_nb)
precision_score_test_nb = metrics.precision_score(y_test,y_test_nb)
print("Naive Bayes Classifier : precision on training Data: {:.3f}".format(precision_score_train_nb))
print("Naive Bayes Classifier : precision on test Data: {:.3f}".format(precision_score_test_nb))
```

```
Naive Bayes Classifier : Accuracy on training Data: 0.605
Naive Bayes Classifier : Accuracy on test Data: 0.605

Naive Bayes Classifier : f1_score on training Data: 0.451
Naive Bayes Classifier : f1_score on test Data: 0.454

Naive Bayes Classifier : Recall on training Data: 0.292
Naive Bayes Classifier : Recall on test Data: 0.294

Naive Bayes Classifier : precision on training Data: 0.997
Naive Bayes Classifier : precision on test Data: 0.995
```

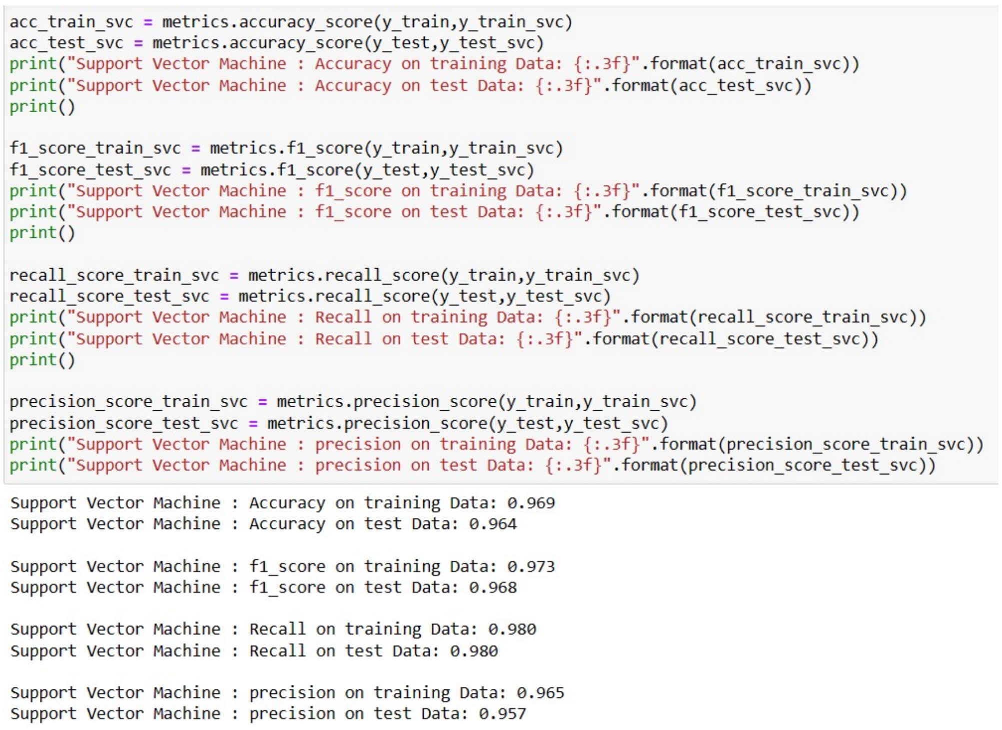
Figure 4.1.11 Naive Bayes Accuracy

- Now we have used Support Vector Classifier.



```
# Support Vector Classifier model
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV

# defining parameter range
param_grid = {'gamma': [0.1],'kernel': ['rbf','linear']}

svc = GridSearchCV(SVC(), param_grid)

# fitting the model for grid search
svc.fit(X_train, y_train)
```

```
#predicting the target value from the model for the samples
y_train_svc = svc.predict(X_train)
y_test_svc = svc.predict(X_test)
```

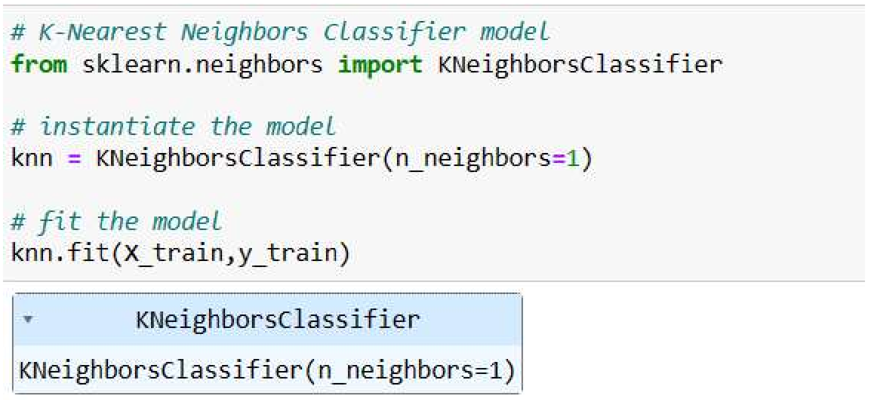


```
acc_train_svc = metrics.accuracy_score(y_train,y_train_svc)
acc_test_svc = metrics.accuracy_score(y_test,y_test_svc)
print("Support Vector Machine : Accuracy on training Data: {:.3f}".format(acc_train_svc))
print("Support Vector Machine : Accuracy on test Data: {:.3f}".format(acc_test_svc))
print()

f1_score_train_svc = metrics.f1_score(y_train,y_train_svc)
f1_score_test_svc = metrics.f1_score(y_test,y_test_svc)
print("Support Vector Machine : f1_score on training Data: {:.3f}".format(f1_score_train_svc))
print("Support Vector Machine : f1_score on test Data: {:.3f}".format(f1_score_test_svc))
print()

recall_score_train_svc = metrics.recall_score(y_train,y_train_svc)
recall_score_test_svc = metrics.recall_score(y_test,y_test_svc)
print("Support Vector Machine : Recall on training Data: {:.3f}".format(recall_score_train_svc))
print("Support Vector Machine : Recall on test Data: {:.3f}".format(recall_score_test_svc))
print()

precision_score_train_svc = metrics.precision_score(y_train,y_train_svc)
precision_score_test_svc = metrics.precision_score(y_test,y_test_svc)
print("Support Vector Machine : precision on training Data: {:.3f}".format(precision_score_train_svc))
print("Support Vector Machine : precision on test Data: {:.3f}".format(precision_score_test_svc))
```

```
Support Vector Machine : Accuracy on training Data: 0.969
Support Vector Machine : Accuracy on test Data: 0.964

Support Vector Machine : f1_score on training Data: 0.973
Support Vector Machine : f1_score on test Data: 0.968

Support Vector Machine : Recall on training Data: 0.980
Support Vector Machine : Recall on test Data: 0.980

Support Vector Machine : precision on training Data: 0.965
Support Vector Machine : precision on test Data: 0.957
```

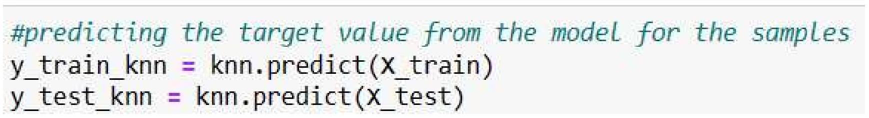
Figure 4.1.12 Support Vector Machine Accuracy

- Now we have used KNN Classifier Model.

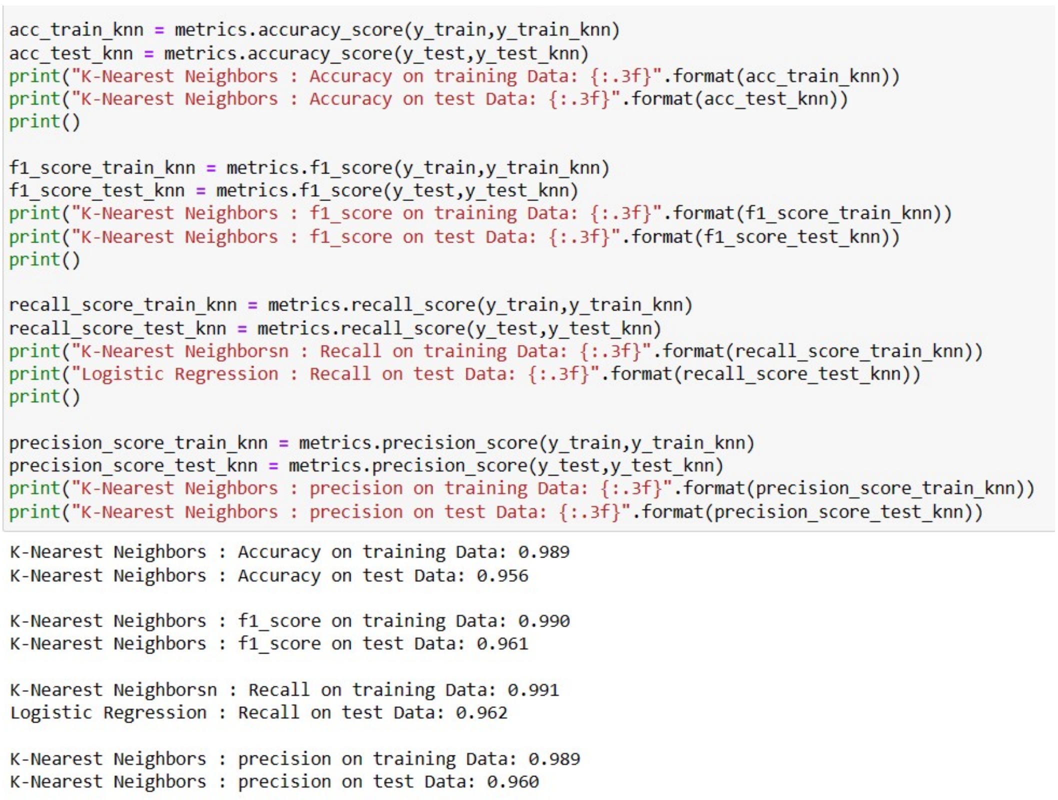


```
# K-Nearest Neighbors Classifier model
from sklearn.neighbors import KNeighborsClassifier

# instantiate the model
knn = KNeighborsClassifier(n_neighbors=1)

# fit the model
knn.fit(X_train,y_train)
```

```
▾        KNeighborsClassifier
KNeighborsClassifier(n_neighbors=1)
```



```
#predicting the target value from the model for the samples
y_train_knn = knn.predict(X_train)
y_test_knn = knn.predict(X_test)
```



```
acc_train_knn = metrics.accuracy_score(y_train,y_train_knn)
acc_test_knn = metrics.accuracy_score(y_test,y_test_knn)
print("K-Nearest Neighbors : Accuracy on training Data: {:.3f}".format(acc_train_knn))
print("K-Nearest Neighbors : Accuracy on test Data: {:.3f}".format(acc_test_knn))
print()

f1_score_train_knn = metrics.f1_score(y_train,y_train_knn)
f1_score_test_knn = metrics.f1_score(y_test,y_test_knn)
print("K-Nearest Neighbors : f1_score on training Data: {:.3f}".format(f1_score_train_knn))
print("K-Nearest Neighbors : f1_score on test Data: {:.3f}".format(f1_score_test_knn))
print()

recall_score_train_knn = metrics.recall_score(y_train,y_train_knn)
recall_score_test_knn = metrics.recall_score(y_test,y_test_knn)
print("K-Nearest Neighborsn : Recall on training Data: {:.3f}".format(recall_score_train_knn))
print("Logistic Regression : Recall on test Data: {:.3f}".format(recall_score_test_knn))
print()

precision_score_train_knn = metrics.precision_score(y_train,y_train_knn)
precision_score_test_knn = metrics.precision_score(y_test,y_test_knn)
print("K-Nearest Neighbors : precision on training Data: {:.3f}".format(precision_score_train_knn))
print("K-Nearest Neighbors : precision on test Data: {:.3f}".format(precision_score_test_knn))
```

```
K-Nearest Neighbors : Accuracy on training Data: 0.989
K-Nearest Neighbors : Accuracy on test Data: 0.956

K-Nearest Neighbors : f1_score on training Data: 0.990
K-Nearest Neighbors : f1_score on test Data: 0.961

K-Nearest Neighborsn : Recall on training Data: 0.991
Logistic Regression : Recall on test Data: 0.962

K-Nearest Neighbors : precision on training Data: 0.989
K-Nearest Neighbors : precision on test Data: 0.960
```

Figure 4.1.13 K-Nearest Neighbor Accuracy

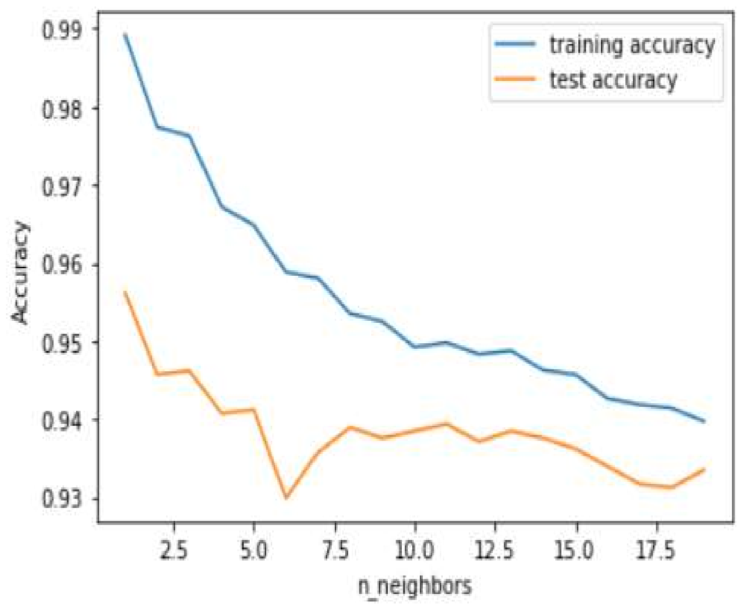
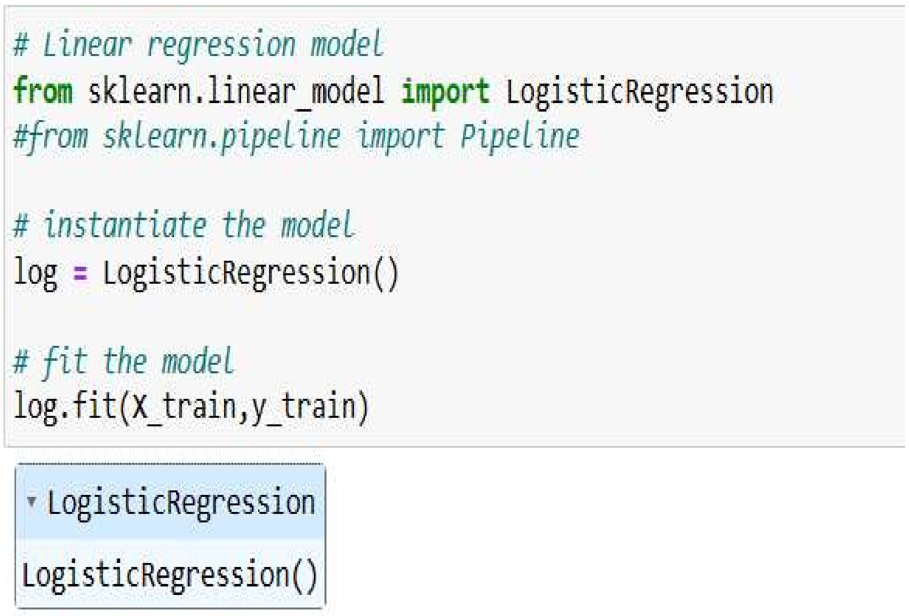


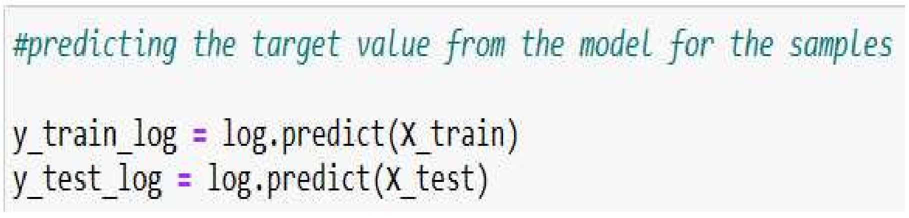
Figure 4.1.14 Training Accuracy vs Test Accuracy

- Now we have used Logistic Regression Model.

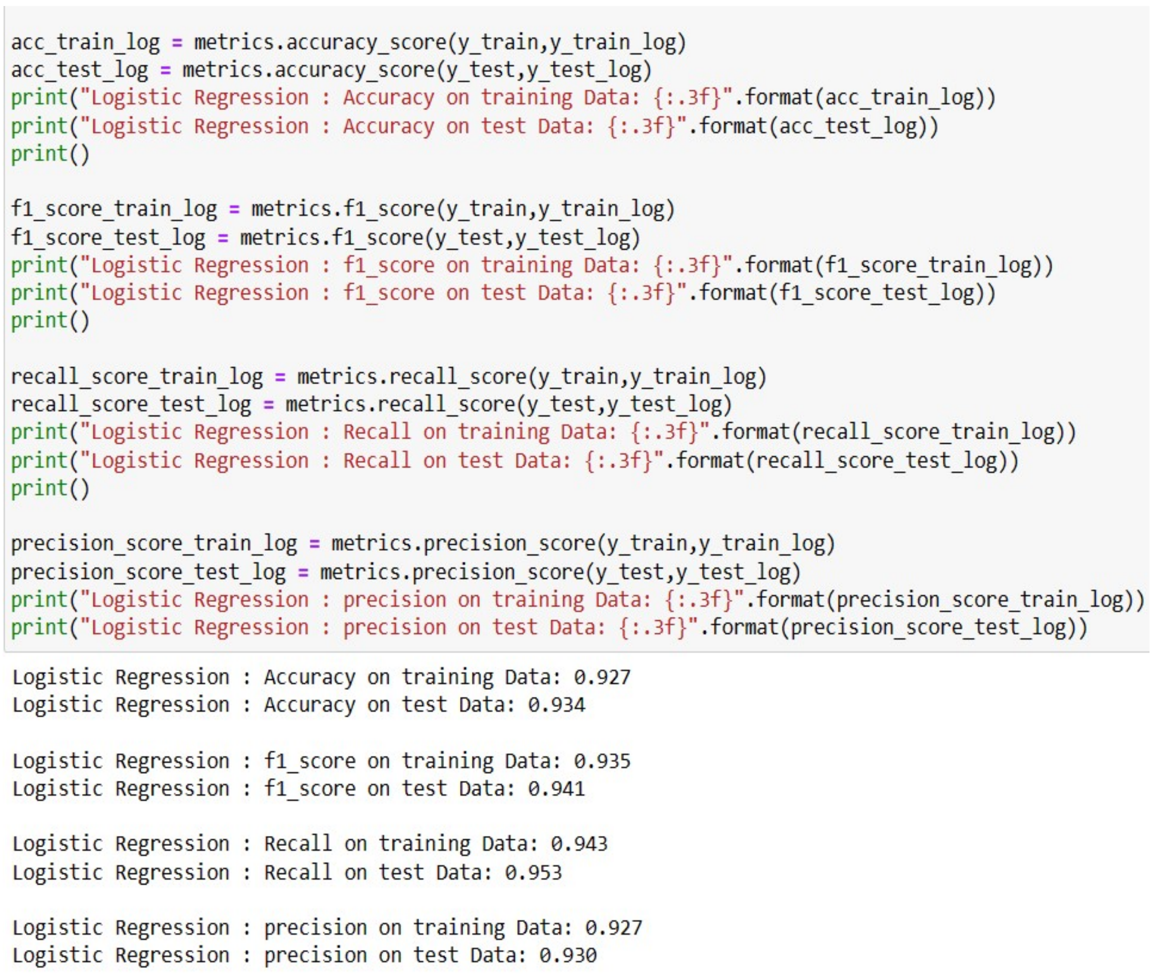


```
# Linear regression model
from sklearn.linear_model import LogisticRegression
#from sklearn.pipeline import Pipeline

# instantiate the model
log = LogisticRegression()

# fit the model
log.fit(X_train,y_train)
```

```
▾ LogisticRegression
LogisticRegression()
```



```
#predicting the target value from the model for the samples

y_train_log = log.predict(X_train)
y_test_log = log.predict(X_test)
```



```
acc_train_log = metrics.accuracy_score(y_train,y_train_log)
acc_test_log = metrics.accuracy_score(y_test,y_test_log)
print("Logistic Regression : Accuracy on training Data: {:.3f}".format(acc_train_log))
print("Logistic Regression : Accuracy on test Data: {:.3f}".format(acc_test_log))
print()

f1_score_train_log = metrics.f1_score(y_train,y_train_log)
f1_score_test_log = metrics.f1_score(y_test,y_test_log)
print("Logistic Regression : f1_score on training Data: {:.3f}".format(f1_score_train_log))
print("Logistic Regression : f1_score on test Data: {:.3f}".format(f1_score_test_log))
print()

recall_score_train_log = metrics.recall_score(y_train,y_train_log)
recall_score_test_log = metrics.recall_score(y_test,y_test_log)
print("Logistic Regression : Recall on training Data: {:.3f}".format(recall_score_train_log))
print("Logistic Regression : Recall on test Data: {:.3f}".format(recall_score_test_log))
print()

precision_score_train_log = metrics.precision_score(y_train,y_train_log)
precision_score_test_log = metrics.precision_score(y_test,y_test_log)
print("Logistic Regression : precision on training Data: {:.3f}".format(precision_score_train_log))
print("Logistic Regression : precision on test Data: {:.3f}".format(precision_score_test_log))
```

```
Logistic Regression : Accuracy on training Data: 0.927
Logistic Regression : Accuracy on test Data: 0.934

Logistic Regression : f1_score on training Data: 0.935
Logistic Regression : f1_score on test Data: 0.941

Logistic Regression : Recall on training Data: 0.943
Logistic Regression : Recall on test Data: 0.953

Logistic Regression : precision on training Data: 0.927
Logistic Regression : precision on test Data: 0.930
```

Figure 4.1.15 Logistic Regression Accuracy

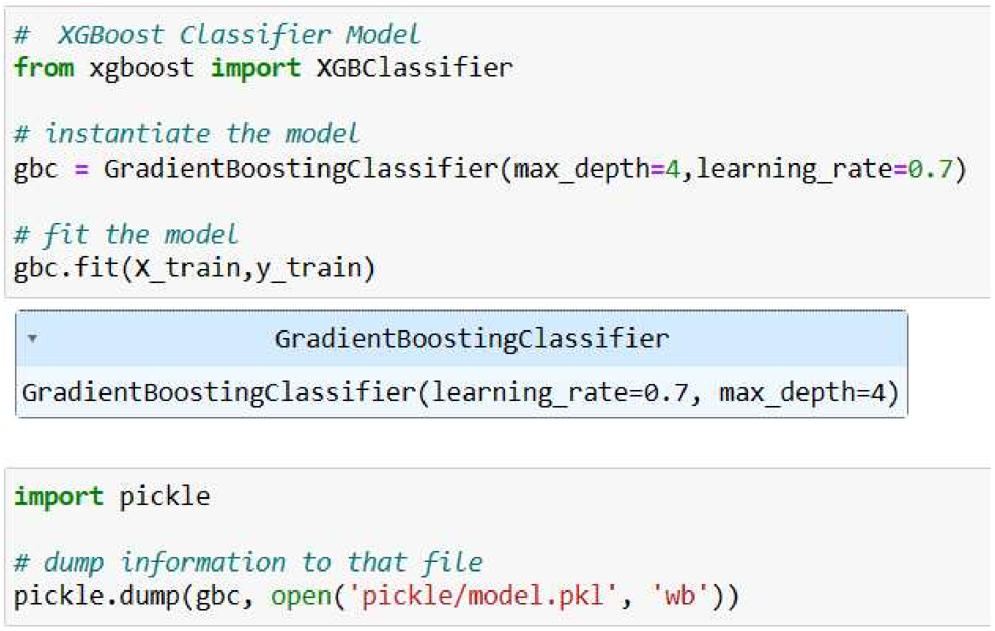
When comparing the accuracy of different machine learning models, it's crucial to consider several key points to ensure a meaningful and reliable evaluation. First, it's important to choose appropriate evaluation metrics beyond just accuracy, such as precision, recall, F1-score, or AUC-ROC, depending on the nature of the data and problem. Second, utilize techniques like k-fold cross-validation to assess each model's performance robustness by training and evaluating on multiple data subsets. Third, balance model complexity with performance gains and interpretability, especially for real-world deployment. Some models may be more complex but achieve higher accuracy, so it's essential to evaluate their practical implications.

- At last we have made the comparisons of the model on the basis of accuracy.

	ML Model	**Accuracy**	**F1_score**	**Recall**	**Precision**
0	Gradient Boosting Classifier	0.974	0.977	0.994	0.986
1	Multi-layer Perceptron	0.971	0.974	0.994	0.982
2	Random Forest	0.966	0.969	0.994	0.987
3	SVM	0.964	0.968	0.980	0.965
4	KNN	0.956	0.961	0.991	0.993
5	Logistic Regression	0.934	0.941	0.943	0.927
6	Decision Tree	0.959	0.964	0.980	0.965
7	Naive Bayes Classifier	0.605	0.454	0.292	0.997

Table 4.1 Comparison of Accuracy

- Now we have saved our model



```
#  XGBoost Classifier Model
from xgboost import XGBClassifier

# instantiate the model
gbc = GradientBoostingClassifier(max_depth=4,learning_rate=0.7)

# fit the model
gbc.fit(X_train,y_train)
```

```
GradientBoostingClassifier
GradientBoostingClassifier(learning_rate=0.7, max_depth=4)
```

```
import pickle

# dump information to that file
pickle.dump(gbc, open('pickle/model.pkl', 'wb'))
```

Figure 4.1.16 Saving the Model

4.2 Result

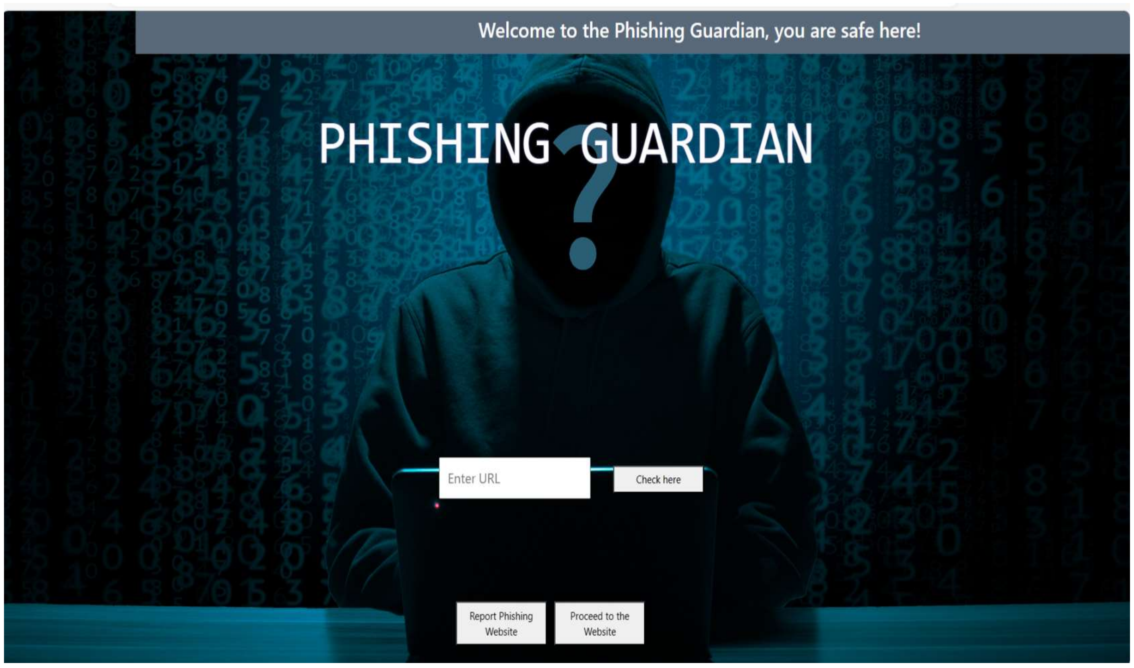
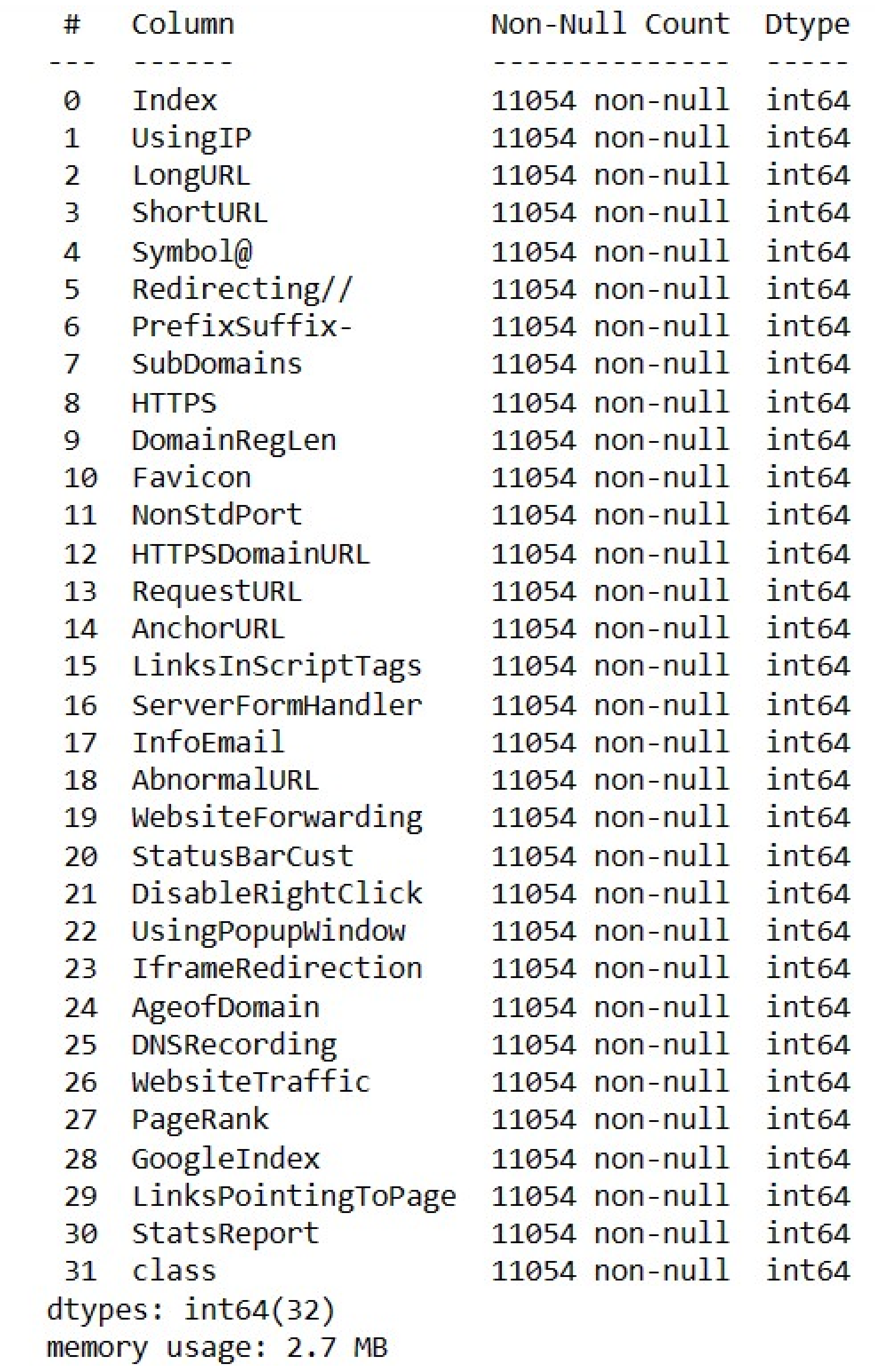


Figure 4.2.1 UI



```
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   Index                11054 non-null  int64
 1   UsingIP              11054 non-null  int64
 2   LongURL              11054 non-null  int64
 3   ShortURL             11054 non-null  int64
 4   Symbol@              11054 non-null  int64
 5   Redirecting//        11054 non-null  int64
 6   PrefixSuffix-        11054 non-null  int64
 7   SubDomains           11054 non-null  int64
 8   HTTPS                11054 non-null  int64
 9   DomainRegLen         11054 non-null  int64
 10  Favicon              11054 non-null  int64
 11  NonStdPort           11054 non-null  int64
 12  HTTPSDomainURL       11054 non-null  int64
 13  RequestURL           11054 non-null  int64
 14  AnchorURL            11054 non-null  int64
 15  LinksInScriptTags    11054 non-null  int64
 16  ServerFormHandler    11054 non-null  int64
 17  InfoEmail            11054 non-null  int64
 18  AbnormalURL          11054 non-null  int64
 19  WebsiteForwarding    11054 non-null  int64
 20  StatusBarCust        11054 non-null  int64
 21  DisableRightClick    11054 non-null  int64
 22  UsingPopupWindow     11054 non-null  int64
 23  IframeRedirection    11054 non-null  int64
 24  AgeofDomain          11054 non-null  int64
 25  DNSRecording         11054 non-null  int64
 26  WebsiteTraffic       11054 non-null  int64
 27  PageRank             11054 non-null  int64
 28  GoogleIndex          11054 non-null  int64
 29  LinksPointingToPage  11054 non-null  int64
 30  StatsReport          11054 non-null  int64
 31  class                11054 non-null  int64
dtypes: int64(32)
memory usage: 2.7 MB
```

Figure 4.2.2 Dataset Features

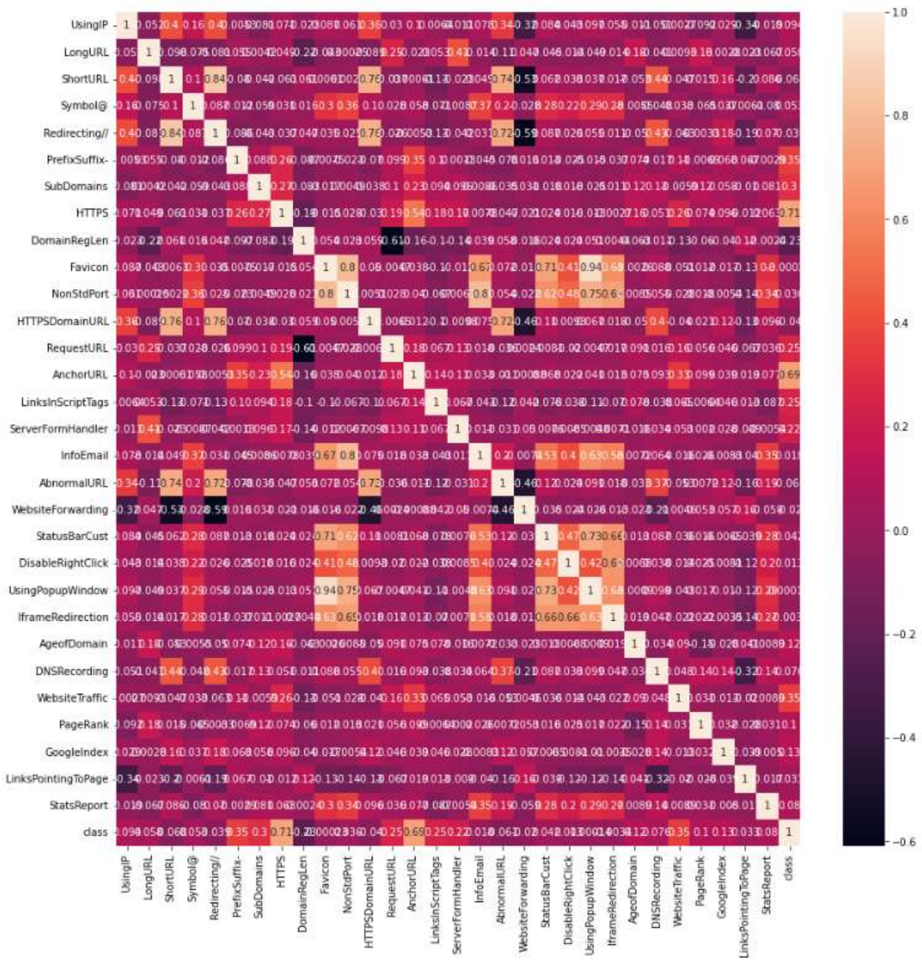


Figure 4.2.3 Features Heat Map

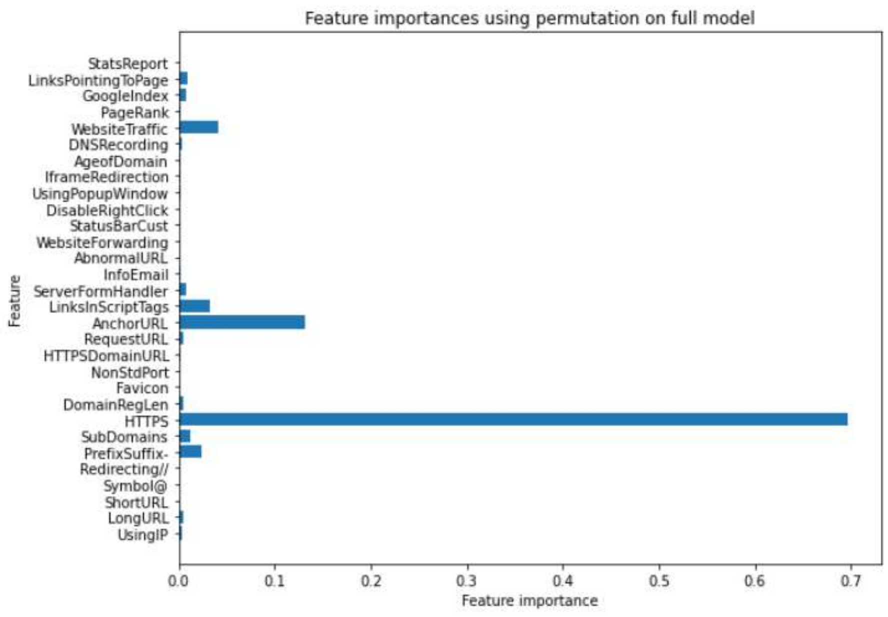


Figure 4.2.4 Feature Importance

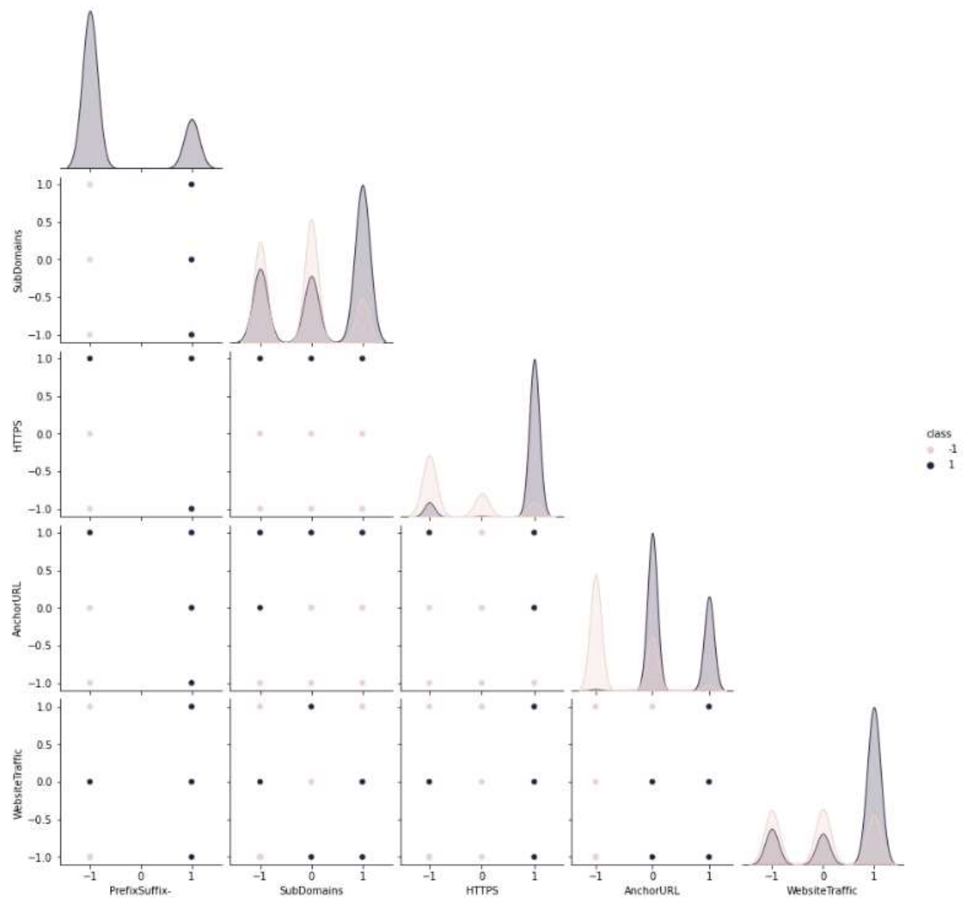


Figure 4.2.5 Pair Plot of Features

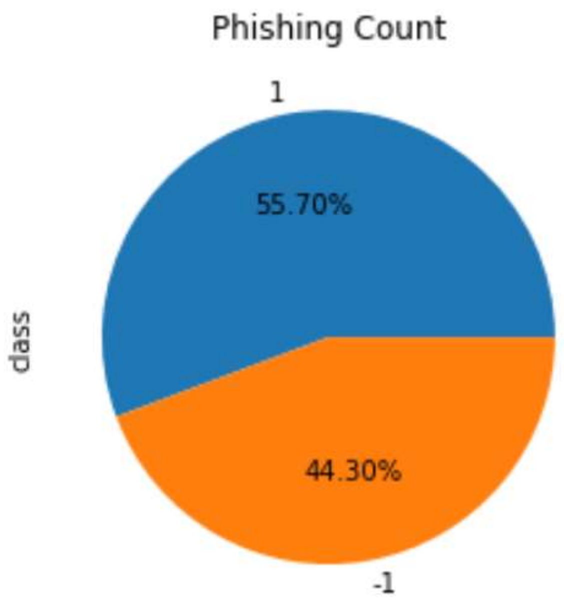
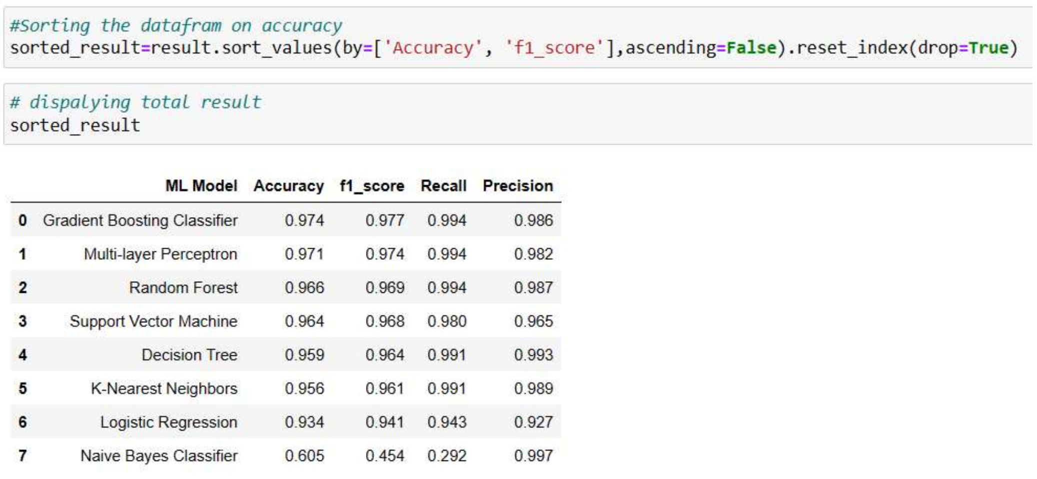


Figure 4.2.6 Pie Chart Phishing Count

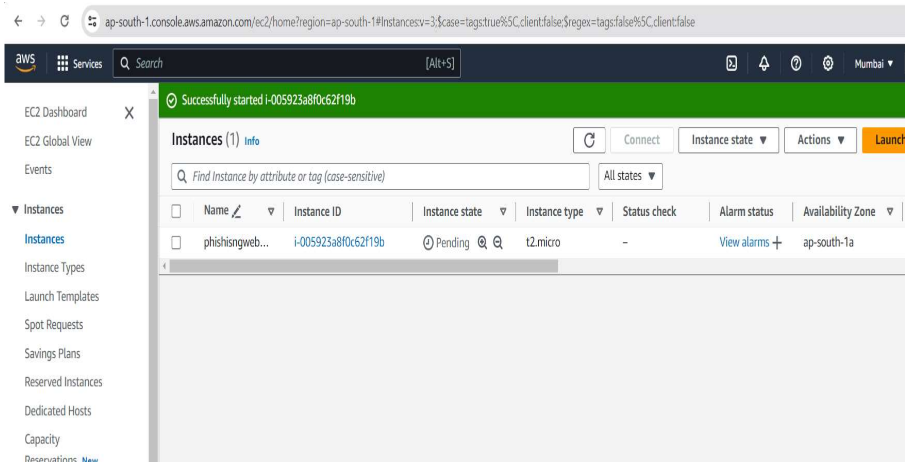


```python
#Sorting the datafram on accuracy
sorted_result=result.sort_values(by=['Accuracy', 'f1_score'],ascending=False).reset_index(drop=True)
```

```python
# dispalying total result
sorted_result
```

	ML Model	Accuracy	f1_score	Recall	Precision
0	Gradient Boosting Classifier	0.974	0.977	0.994	0.986
1	Multi-layer Perceptron	0.971	0.974	0.994	0.982
2	Random Forest	0.966	0.969	0.994	0.987
3	Support Vector Machine	0.964	0.968	0.980	0.965
4	Decision Tree	0.959	0.964	0.991	0.993
5	K-Nearest Neighbors	0.956	0.961	0.991	0.989
6	Logistic Regression	0.934	0.941	0.943	0.927
7	Naive Bayes Classifier	0.605	0.454	0.292	0.997

Figure 4.2.7 Final Accuracy

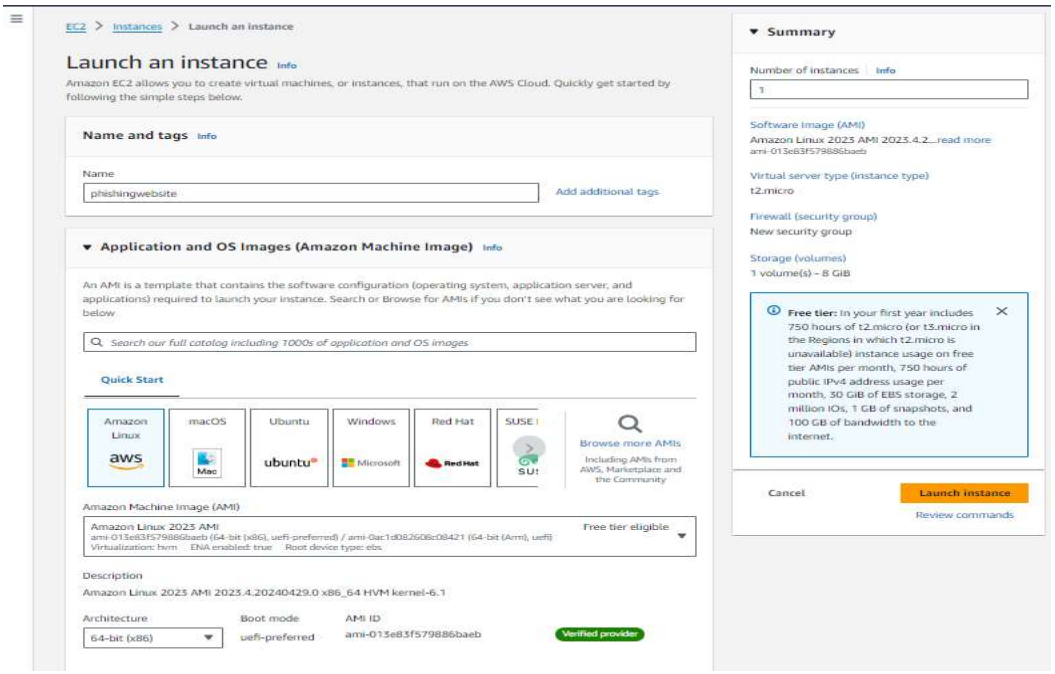

ap-south-1.console.aws.amazon.com/ec2/home?region=ap-south-1#Instances:v=3;$case=tags:true%5C,client:false;$regex=tags:false%5C,client:false
Services
Search
[Alt+S]
Mumbai
EC2 Dashboard
EC2 Global View
Events
Instances
Instances
Instance Types
Launch Templates
Spot Requests
Savings Plans
Reserved Instances
Dedicated Hosts
Capacity Reservations New
Connect
Instance state
Actions
Launch
Find Instance by attribute or tag (case-sensitive)
All states

Successfully started i-005923a8f0c62f19b

Instances (1) Info

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone
phishingweb...	i-005923a8f0c62f19b	Pending	t2.micro	-	View alarms +	ap-south-1a

Figure 4.2.8 AWS Console



Launch an instance

Amazon EC2 allows you to create virtual machines, or instances, that run on the AWS Cloud. Quickly get started by following the simple steps below.

Name and tags

Name: phishingwebsite

Add additional tags

Application and OS Images (Amazon Machine Image)

An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. Search or Browse for AMIs if you don't see what you are looking for below

Search our full catalog including 1000s of application and OS images

Amazon Machine Image (AMI)

Amazon Linux 2023 AMI — Free tier eligible
ami-013e83f579886baeb (64-bit (x86), uefi-preferred) / ami-0ac1d082606c08421 (64-bit (Arm), uefi)
Virtualization: hvm ENA enabled: true Root device type: ebs

Description

Amazon Linux 2023 AMI 2023.4.20240429.0 x86_64 HVM kernel-6.1

Architecture	Boot mode	AMI ID	
64-bit (x86)	uefi-preferred	ami-013e83f579886baeb	Verified provider

Summary

Number of instances: 1

Software Image (AMI): Amazon Linux 2023 AMI 2023.4.2...read more ami-013e83f579886baeb

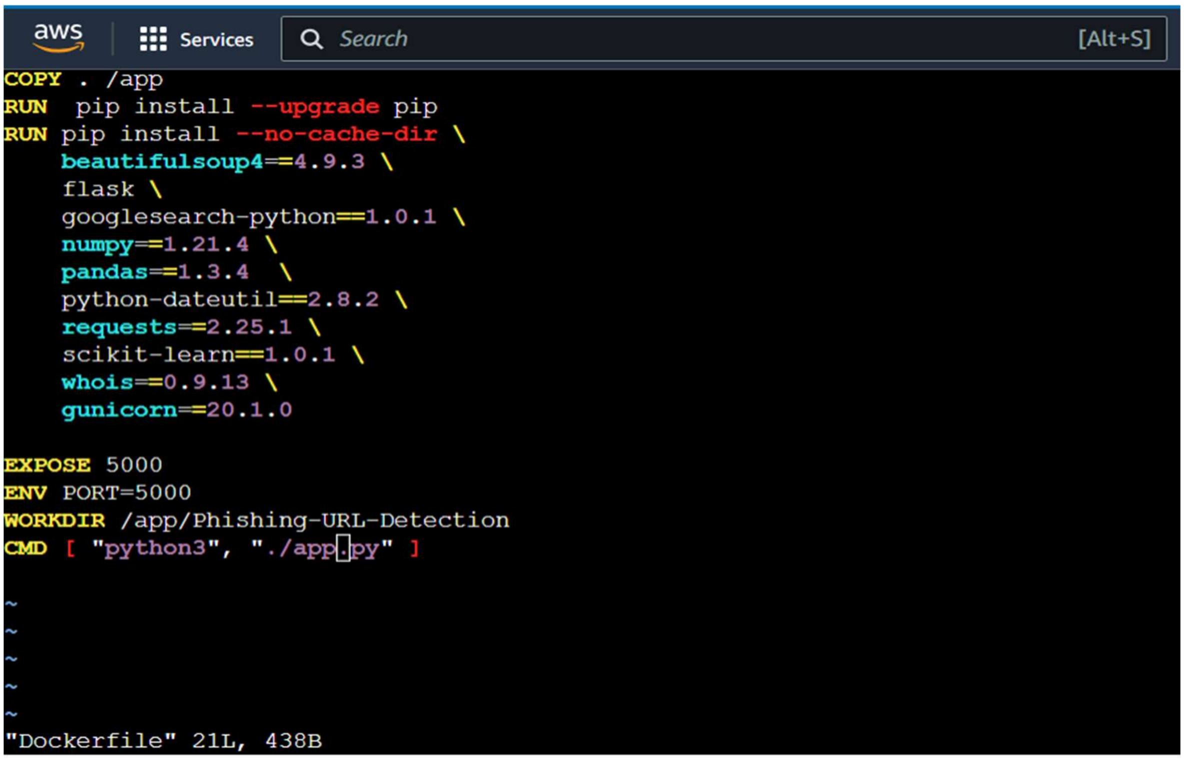
Virtual server type (instance type): t2.micro

Firewall (security group): New security group

Storage (volumes): 1 volume(s) - 8 GiB

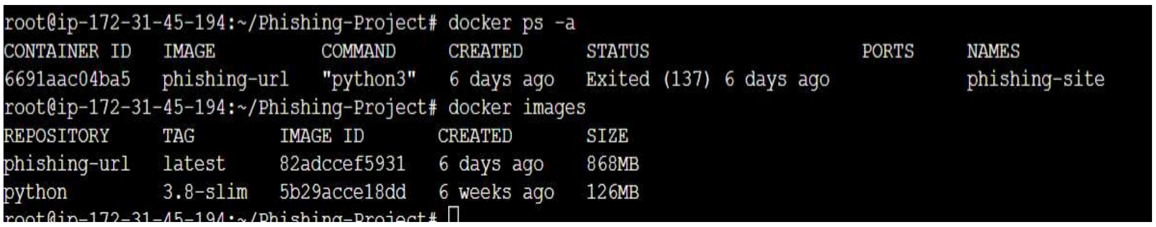
Free tier: In your first year includes 750 hours of t2.micro (or t3.micro in the Regions in which t2.micro is unavailable) instance usage on free tier AMIs per month, 750 hours of public IPv4 address usage per month, 30 GiB of EBS storage, 2 million IOs, 1 GB of snapshots, and 100 GB of bandwidth to the internet.

Figure 4.2.9 EC2 Instance



```
COPY . /app
RUN  pip install --upgrade pip
RUN pip install --no-cache-dir \
    beautifulsoup4==4.9.3 \
    flask \
    googlesearch-python==1.0.1 \
    numpy==1.21.4  \
    pandas==1.3.4   \
    python-dateutil==2.8.2 \
    requests==2.25.1 \
    scikit-learn==1.0.1 \
    whois==0.9.13 \
    gunicorn==20.1.0

EXPOSE 5000
ENV PORT=5000
WORKDIR /app/Phishing-URL-Detection
CMD [ "python3", "./app.py" ]

~
~
~
~
~
"Dockerfile" 21L, 438B
```

Figure 4.2.10 Docker File



```
root@ip-172-31-45-194:~/Phishing-Project# docker ps -a
CONTAINER ID   IMAGE         COMMAND     CREATED      STATUS                    PORTS     NAMES
6691aac04ba5   phishing-url  "python3"   6 days ago   Exited (137) 6 days ago             phishing-site
root@ip-172-31-45-194:~/Phishing-Project# docker images
REPOSITORY     TAG        IMAGE ID       CREATED      SIZE
phishing-url   latest     82adccef5931   6 days ago   868MB
python         3.8-slim   5b29acce18dd   6 weeks ago  126MB
root@ip-172-31-45-194:~/Phishing-Project# 
```

Figure 4.2.11 Docker Container

Chapter 05: CONCLUSIONS

In conclusion, our project focused on the classification of URLs into different categories, differentiating between legitimate and potentially malicious links. We employed a comprehensive approach, utilizing machine learning techniques and comparing four algorithms: Random Forest, Decision Tree, Support Vector Machine (SVM), Gradient Boost, Logistic Regression, Naive-Bayes, KNN and Multilayer Perceptron. After thorough evaluation, Gradient Boost emerged as the most effective algorithm, achieving an impressive accuracy score of 97.4%. The success of Gradient Boost underscores its suitability for the complexities of URL classification, demonstrating robust performance indistinguishing between benign and harmful links. The project encompassed key steps such as feature extraction, where 32 relevant features were derived from address bar, domain-based, and HTML/JavaScript-based attributes. Additionally, the incorporation of a diverse dataset, encompassingboth legitimate and phishing URLs, contributed to the model's comprehensive training and performance evaluation. To make the solution accessible, we implemented a Flask web application with an intuitive user interface. The application allows users to input URLs for real-time classification using the trained Gradient Boost model. This user- friendly tool serves as a practical means for individuals and organizations to assess the nature of URLs and identify potential security threats.

We created a user-friendly Flask web application in order to make the solution accessible. Users can input URLs into the application to have the trained Gradient Boost model classify them in real time. With the help of this easy-to-use tool, people and organizations can evaluate URLs' characteristics and spot possiblesecurity risks. In essence, the project successfully addressed the challenge of URL classification, providing a reliable and accurate solution for distinguishing between safe and malicious links. The utilization of Gradient Boost, coupled with the Flask web application, results in a versatile tool with practical applications for enhancing online security and threat detection. At last we have created the Dockerfile which offers us several benefits such as consistency, portability, isolation and scalability

5.2 Future Scope

The future scope of this project entails enriching the Flask web application (app.py) by adding features like an enhanced user interface, detailed result insights, and real-time threat updates. There's room for advancing the machine learning and deep learning models, incorporating more sophisticated techniques, and maintaining an up-to-date dataset to adapt to evolving online threats. Potential improvements include user authentication, features for managing and tracking URL classifications, and enhanced security measures. These developments will solidify the project's position as a valuable tool for users seeking effective and reliable means of assessing the security of online links.

REFERENCES

[1] https://www.kaggle.com/eswarchandt/phishing-website-detector .

[2] https://www.unb.ca/cic/datasets/url-2016.html.

[3] Chunlin Liu, Bo Lang : Finding effective classifier for malicious URL detection : InACM,2018https://www.researchgate.net/profile/ErPurviPujara/publication/331198983 Phishing_Website_Detection_using_Machine_Learning_A_Review/links/5c6bd4ae4585 156b5706e727/PhishingWebsite-Detection-using-Machine-Learning-A-Review.pdf

[4] Ankit Kumar Jain, B. B. Gupta : Towards detection of phishing websites on client-side using machine learning based approach :In Springer Science+Business Media, LLC, part of Springer Nature 2017

[5] https://Rishkesh_Mahajan/publication/328541785_Phishing_Website_De tection_using_Machin e_Learning_Algorithms/links/5d0397fd92851c9004394af4/Phishing-Website-Detection-using-Machine-Learning-Algorithms.pdf

[6] Ahmad Abunadi, Anazida Zainal, Oluwatobi Akanb: Feature Extraction Process: A Phishing Detection Approach :In IEEE,2013, https://www.irjet.net/archives/V3/i5/IRJET-V3I5420.pdf

[7] Sahingoz, O.K., Buber, E., Demir, O. and Diri, B., 2019. Machine learning based phishing detection from URLs. Expert Systems with Applications, 117, pp.345-357. https://avesis.yildiz.edu.tr/yayin/be703f82-6dbf-4657-bf14-c671f1e89304/machine-learning-based- phishing-detection-from-urls

[8] https://www.datacamp.com/blog/classification-machine-learning

[9] https://www.researchgate.net/figure/Classification-vs-Regression_fig2_350993856

[10] https://medium.com/analytics-vidhya/random-forest-classifier-and-its-hyperparameters-8467bec755f6

[11] https://www.javatpoint.com/machine-learning-decision-tree-classificatio-algorithm

[12] Sci-kit learn, SVM library. http://scikit-learn.org/stable/modules/svm.html.

[13] https://www.simplilearn.com/what-is-xgboost-algorithm-in-machine-learning-article#:~:text=XGBoost%20is%20a%20robust%20machine,optimize%20their%20machine%2Dlearning%20m

[14] https://pythonbasics.org/what-is-flask-python/

[15] https://www.unb.ca/cic/datasets/url-2016.html

[16] A. Ng, M. I. Jordan, "On Discriminative vs. Generative Classifiers: A Comparison of Logistic Regression and Naive Bayes," in Proceedings of the 14th International Conference on Neural Information Processing Systems, Vancouver, BC, Canada, 2001, pp. 841-848.

[17] J. Friedman, T. Hastie, R. Tibshirani, "The Elements of Statistical Learning: Data Mining, Inference, and Prediction," 2nd ed. New York, NY, USA: Springer, 2009.

[18] Y. LeCun, Y. Bengio, G. Hinton, "Deep learning," Nature, vol. 521, no. 7553, pp. 436-444, May 2015.

[19] J. Wang, J. Yang, and Y. Xu, "A Comparative Study of K-Nearest Neighbor and Support Vector Machine Algorithms for Classification," in Proceedings of the IEEE International Conference on Data Mining, Washington, DC, USA, 2018, pp. 572-581

[20] https://nupmanyu.medium.com/why-docker-is-so-fast-7bd1f91cf21e

APPENDICES

Raw Code-

```
#importing required libraries

from flask import Flask, request, render_template
import numpy as np
import pandas as pd
from sklearn import metrics
import warnings,import pickle
warnings.filterwarnings('ignore')
from feature import FeatureExtraction
file = open("pickle/model.pkl","rb")
gbc = pickle.load(file)
file.close()
app = Flask(__name__)
@app.route("/", methods=["GET", "POST"])
def index():
   if request.method == "POST":
      url = request.form["url"]
      obj = FeatureExtraction(url)
      x = np.array(obj.getFeaturesList()).reshape(1,30)
      y_pred =gbc.predict(x)[0]

      y_pro_phishing = gbc.predict_proba(x)[0,0]
      y_pro_non_phishing = gbc.predict_proba(x)[0,1]
      if(y_pred ==1 ):
      pred = "It is {0:.2f} % safe to go ".format(y_pro_phishing*100)
      return render_template('index.html',xx =round(y_pro_non_phishing,2),url=url )
   return render_template("index.html", xx =-1)
if __name__ == "__main__":
   app.run(debug=True)
```

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY, WAKNAGHAT

PLAGIARISM VERIFICATION REPORT

Date: ……………………….

Type of Document (Tick): PhD Thesis | M.Tech Dissertation/ Report | B.Tech Project Report | Paper

Name: ____________________________**Department:** ________________**Enrolment No** _________

Contact No. ____________________________**E-mail.** ____________________________________

Name of the Supervisor: ____

Title of the Thesis/Dissertation/Project Report/Paper (In Capital letters): _______________________

____

____

UNDERTAKING

I undertake that I am aware of the plagiarism related norms/ regulations, if I found guilty of any plagiarism and copyright violations in the above thesis/report even after award of degree, the University reserves the rights to withdraw/revoke my degree/report. Kindly allow me to avail Plagiarism verification report for the document mentioned above.

Complete Thesis/Report Pages Detail:

– Total No. of Pages =
– Total No. of Preliminary pages =
– Total No. of pages accommodate bibliography/references =

(Signature of Student)

FOR DEPARTMENT USE

We have checked the thesis/report as per norms and found **Similarity Index** at.................. (%). Therefore, we
are forwarding the complete thesis/report for final plagiarism check. The plagiarism verification report may be handed over to the candidate.

(Signature of Guide/Supervisor) **Signature of HOD**

FOR LRC USE

The above document was scanned for plagiarism check. The outcome of the same is reported below:

Copy Received on	**Excluded**	**Similarity Index (%)**	**Generated Plagiarism Report Details** (Title, Abstract & Chapters)	
	• All Preliminary Pages • Bibliography/Images/Quotes • 14 Words String		Word Counts	
Report Generated on			Character Counts	
		Submission ID	Total Pages Scanned	
			File Size	

Checked by
Name & Signature **Librarian**

………

Please send your complete thesis/report in (PDF) with Title Page, Abstract and Chapters in (Word File) through the supervisor at plagcheck.juit@gmail.com

project report

ORIGINALITY REPORT

17%	11%	3%	8%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	www.ir.juit.ac.in:8080 Internet Source	4%
2	github.com Internet Source	1%
3	bmsit.ac.in Internet Source	1%
4	www.ijraset.com Internet Source	1%
5	www.mdpi.com Internet Source	1%
6	Submitted to Intercollege Student Paper	1%
7	Submitted to University of Sydney Student Paper	1%
8	Submitted to Queen Mary and Westfield College Student Paper	1%
9	Submitted to Southeast University Student Paper	<1%