DETECTING CROSS-SITE SCRIPTING ATTACKS USING MACHINE LEARNING ALGORITHM

Project work submitted to Avinashilingam institute for Home Science and Higher Education for Women

MASTER OF SCIENCE IN INFORMATION TECHNOLOGY

SUBMITTED BY

VARSHINI S

(21PIT011)

UNDER THE GUIDANCE OF

Mrs. S. KARTHIKA M.C.A., M.Phil. NET

Assistant Professor (Dept of IT)

Project Report Submitted

In partial fulfilment of the requirements for the Award of

DEPARTMENT OF INFORMATION TECHNOLOGY

Avinashilingam Institute for Home Science and Higher Education for Women

Coimbatore – 641043

MAY – 2023

DECLARATION

DECLARATION

I hereby declare that the project entitled "**DETECTING CROSS SITE SCRIPTING ATTACKS USING MACHINE LEARNING ALGORITHM**" Mrs. S. KARTHIKA M.C.A., M.Phil. NET Assistant Professor (Dept of IT) is a record of the original work done by **Varshini S**(21PIT011) under the guidance of Assistant Professor and Head, Department of Information Technology, School of Physical Sciences and Computational Sciences, Avinashilingam Institute for Home Science and Higher Education for Women in the partial fulfillment for the award of the degree of Master of Science in Information Technology, and this project work has not formed the basis for any Degree/Diploma/Associates.

Place: Coimbatore

Date: 19/05/2023

Signature of the candidate

Countersigned By,

19/05/2

Mrs. S. KARTHIKA M.C.A., M.Phil. NET

Assistant Professor (Dept of IT)

Department of Information Technology,
School of Physical Sciences and Computational Sciences.

CERTIFICATE

CERTIFICATE

Avinashilingam Institute for Home Science and Higher Education for Women

(Deemed to be University under Estd. u/s 3 of UGC Act 1956, Category 'A' by MHRD)

Re-accredited with 'A++' Grade by NAAC. CGPA 3.65/4, Category 1 University by UGC

Coimbatore - 641 043, Tamil Nadu, India

DST - CURIE - AI Sponsored

Centre for Cyber Intelligence

CERTIFICATE OF PROJECT COMPLETION

This is to certify that Ms. Varshini S (21PIT011), Master of Information Technology, Avinashilingam Institute for Home Science and Higher Education for Women, has successfully completed the project entitled "Detection of Cross-Site Scripting Attack using Machine Learning Algorithms" under Centre for Cyber Intelligence - Centre for Machine Learning and Intelligence – a DST - CURIE - AI facility during December 2022 - May 2023.

Dr. G. Padmavathi
Dean, School of PSCS
CCI - Principal Investigator

Dr. P. Subashini
Project Coordinator - DST - CURIE - AI

Dr. S. Kowsalya
Registrar

CERTIFICATE

This is to certify that this project work entitled "**DETECTING CROSS SITE SCRIPTING ATTACKS USING MACHINE ALGORITHM"** done by **VARSHINI S (21PIT011)** has been submitted to Avinashilingam Institute for Home science and Higher education for women, Coimbatore-641 043 inpartial fulfilment of the requirement for the award of the degree of **MASTER OF SCIENCE IN INFORMATION TECHNOLOGY.** This Project has not foundthe basis for the award of any Degree/Associate/fellowship or similar title to any Candidate of any University. Certified as a Bonafide record of the work submitted for the Viva voce held on____

Signature of the HOD

19/5/2023
Signature of The Guide

Signature of External Examiner

ACKNOWLEDGEMENT

ACKNOWLEDGEMENT

I sincerely thank the **Lord Almighty** and **My lovable parents** for showering their generous blessings upon me in all endeavors.

I wish to express my gratitude to **Prof.S.P.Thyagarajan,** Chancellor, Avinashilingam Institute for Home Science and Higher Education for Women, Coimbatore, for providing thefacilities to conduct this study.

I extend my thanks to **Dr. Bharathi Harishankar, Ph.D., FRSA.,** Vice Chancellor, Avinashilingam Institute for Home Science and Higher Education for Women, Coimbatore, for providing flamboyant help towards the completion of the study.

I record my deep sense of gratitude and indebtedness to **Dr. S. Kowsalya,** M.Sc., M.Phil., Ph.D., Registrar, Avinashilingam Institute for Home Science and Higher Education for Women, Coimbatore, for providing adequate help for the study

I grateful record my sincere thanks to **Dr. G. Padmavathi** M.Sc., M.Phil., Ph.D., Dean and Professor, School of Physical Sciences & Computational of Sciences, Avinashilingam Institute for Home Science and Higher Education for Women, Coimbatore, for timely help rendered throughout the course of this work.

I heartily Thanks to **Dr. Mrs. D. Shanmugapriya** M.Sc., M.Phil., Ph.D., SET Head of Department of Information Technology for the valuable guidance and encouragement during our project.

I heartily Thank my esteemed project guide **Assistant Professor,Mrs.S.Karthika M.C.A., M.Phil.NET** Department of Information Technology, for imparting tremendous assistance and well-timed support for triumph of our project.

I like to extend my gratitude to Ms.A.Roshini, Technical Assistant –Center of cyber intelligence, Department of Computer Science, For providing Project guidelines and always supported me and encouraged me with valuable advice and Profound belief in my work and abilities

I express my honorable thanks to our project coordinator Department of Information Technology, for imparting tremendous assistance and well-timed support for triumph of our project.

I sincerely thank all **the staff members** of Department of Information Technology Avinashilingam Institute for Home Science and Higher Education for Women, Coimbatore, for their help and support.

I would like to express my special thanks to **my parents, my friends** and all **my well-wishers** for their constant encouragement, support and help in carrying out this work successfully.

I would like to acknowledge the help rendered by Center for Cyber Intelligence, DST

-CURIE – AI Sponsored Phase II for providing the laboratory facilities to execute my project.

ABSTRACT

ABSTRACT

Aim: To propose a machine learning framework to detect cross-site scripting attacks. This project deals with the development of supervised machine-learning algorithms. This is done using the features based on normal and malicious URLs and JavaScript

Method: The detection of Cross-site scripting attacks using a supervised machine learning approach comprises five phases. Phase 1 is Data Importing - first imported the dataset as a panda's data frame. In Phase 2 we performed pre-processing by removing duplicate entries and encoding the 'Class' column using label encoding. In Phase 3, we applied feature selection using the chi-squared method to select the top 20 features. In Phase 4, The supervised machine learning models are developed with stacked ensemble methods (Random Forest Classifier, Decision Tree Classifier, KNeighbours' Classifier, Support Vector Machine and Naive Bayes) and then they are evaluated. The output of different algorithms is evaluated in phase 5 with performance measures such as precision, recall, F1 Score, accuracy score and ROC curve. Decision Tree Classifier gave better results than Naïve Bayes and Support Vector Machine based on the features extracted from URL and JavaScript code. All the algorithms gave comparatively better results with discretized attributes but a noticeable difference in performance was seen only in the case of SVM. the entire project is developed on the Anaconda Jupyter Python.

Result: From This, the score of Random Forest, KNN, SVM, and Decision Tree Classifier is more effective when comparing to Naives Bayes,

Keywords: cross-site scripting, machine learning, XSS attack, Supervised ML

CONTENTS

TABLE OF CONTENTS

INTRODUCTION

CHAPTER 1

INTRODUCTION

1.1 About the Project

In today's digital world, a web application is the most cost-effective mechanism of software delivery and is used by millions of businesses and other organizations to deliver their services over the Internet. Any user equipped with a computer and an internet connection can access web applications. This easy accessibility of web applications also makes them vulnerable to a wide range of attacks from malicious users. The XSS attack ranks as the number two web application security risk among many categories of attack. XSS vulnerabilities are easily exploitable as many freely available tools are there to enable anyone with minimum knowledge to attack web applications. XSS attacks can lead to session hijacking, sensitive data disclosure, cross-site request forgery (csrf) attacks, and other security vulnerabilities including impersonation of the victim. XSS attacks can also lead to code execution on the server, depending on the application and privileges of the user's account [1]. At present, web applications adopt an XSS prevention or XSS detection and prevention approach. In the XSS prevention approach, the programmer or developer must adopt the necessary means to prevent XSS attacks. Thus collecting cookies, altering the website, copying material from the clipboard, scanning ports, or dynamically loading, XSS can have an impact on the victim. Therefore, it is crucial for developers to focus on the security of web apps. The most prevalent security issue in web applications is the absence of client input verification. Both on the client and server sides, these flaws are continually found and taken advantage of. The top ten vulnerabilities published by the Open Web Application Security Project (OWASP) continue to include XSS attacks.

This paper describes an experiment that results in a model for detecting XSS attacks. The most popular parameters found in JavaScript are used as attributes. Machine learning algorithms are considered: the method of support vectors, the decision tree, KNN, Random Forest, the naive Bayesian classifier, and logistic regression. The accuracy of these machine learning methods is compared.

1.1.1 What is a Web Application?

A web application is a piece of software that is accessible and used online. It is intended to offer consumers a certain functionality or group of functionalities, frequently connected to carrying out transactions of any kind online. Email clients, online marketplaces, social networking sites, and productivity tools like online document editors and project management software are some examples of web applications.

1.1.2 Types of Web Application Attacks

There are various types of web application attacks that can be launched by hackers or malicious actors to exploit vulnerabilities in web applications. Here are some of the most common types of web application attacks:

- Cross-site scripting (XSS): This involves injecting malicious code into a web page viewed by other users, allowing an attacker to steal sensitive information or carry out actions on behalf of the victim.

- SQL injection (SQLi): This involves exploiting vulnerabilities in a web application's database by injecting malicious SQL commands, allowing an attacker to access, modify, or delete sensitive data.

- Cross-site request forgery (CSRF): This involves tricking a user into executing unintended actions on a web application, such as making a payment or changing a password.

- Session hijacking: This involves stealing or impersonating a user's session ID, allowing an attacker to gain access to a user's account without needing their login credentials.

- Clickjacking: This involves hiding malicious elements on a web page, making them appear to be legitimate, and tricking users into clicking on them, allowing an attacker to carry out unauthorized actions.

Figure 1.1.2 Main Threats

1.1.3 What is a Cross-site scripting attack?

An attacker can insert malicious code into a web page that is being seen by other users due to a vulnerability known as cross-site scripting (XSS) in online applications. This code can be used to steal sensitive data, including login passwords or credit card numbers, or to act on behalf of the victim by making unauthorized purchases or updating their account information. When a hacker successfully inserts malicious code into an online application, frequently through a form or input field, and that code is subsequently run by other users who see the page, this is known as an XSS attack. An attacker may be able to inject their own code into a web application when it improperly sanitises user input or fails to validate input fields.

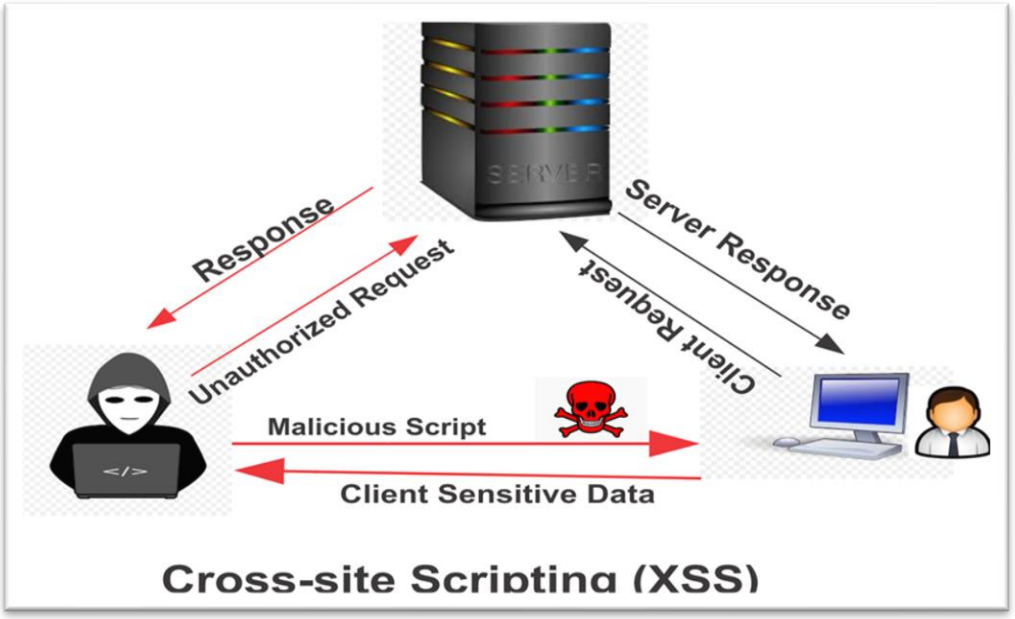


Figure 1.1.3 Cross-site scripting

1.1.4 Types of Cross-site scripting attacks

There are several types of XSS attacks, including:

- Reflected XSS: The attacker injects malicious code into a URL or form field, which is then reflected the user in the application's response.

- Stored XSS: The attacker injects malicious code into a web application's database, which is then displayed to other users who view the affected page.

- DOM-based XSS: The attacker injects malicious code into a web application's Document Object Model (DOM), which is then executed by the victim's browser when they interact with the page.

1.1.5 Machine Learning

Creating a model to recognize trends and anomalies in online traffic data that may point to an attack is the first step in applying machine learning to detect cross-site scripting (XSS) attacks. This often requires training the model on a dataset of known XSS attacks and legal traffic using supervised learning techniques.

The incoming web traffic can then be analyzed by the machine learning model in order to quickly identify any potential XSS threats. The model can send out an alarm or take measures to stop the attack when it recognizes a prospective attack.
These are the Steps involved in building a machine-learning model for detecting XSS Attacks.

- Data collection: Collect a dataset of web traffic that includes examples of XSS attacks and legitimate traffic.

- Data preprocessing: Preprocess the data to clean and normalize it, and extract relevant features that can be used to train the machine learning model.

- Model selection: Choose an appropriate machine learning algorithm and train the model on the preprocessed data.

- Model evaluation: Evaluate the performance of the model using metrics such as accuracy, precision, and recall.

Benefits of using machine learning algorithms for detecting XSS attacks include:

- More accurate detection and fewer false positives are possible due to machine learning algorithms' ability to spot subtle patterns and irregularities in web traffic that rule-based systems or human analysts can miss.

- Machine learning algorithms have the capacity to process massive amounts of online traffic data in real time.

- Scalability: To adapt to varying traffic volumes, machine learning algorithms may be readily scaled up or down and shown on enormous quantities of data.

- Cost savings: Automating the detection process using machine learning algorithms can help cut back on the costs involved with hiring human analysts.

1.2 Motivation and Justification

Cross-site scripting (XSS) threat detection is essential for securing user data and web applications. The ability of traditional detection techniques to identify complex and changing attack patterns can be limited because they frequently rely on rule-based methodologies or pattern matching. By analysing vast amounts of online traffic data and identifying trends and abnormalities that may signal an attack, machine learning algorithms can offer a more effective and efficient method of detecting XSS attacks. These algorithms are better at spotting new and evolving attacks because they can continuously adapt to new attack patterns and learn from previous attack examples.

1.3 Problem Statement

To detect the cross-site scripting attack using machine learning algorithms.

1.4 Objectives

- The objective of the project is to detect XSS attacks using a machine-learning framework to find the accuracy and compare the results.
- Exploring The Classification Models.
- Detecting XSS attacks using machine learning algorithms are to create an accurate, real-time, adaptable, scalable, and user-friendly system for detecting and mitigating XSS attacks in web applications

2. SYSTEM CONFIGURATION

2.1 Hardware Requirement PROCESSOR: Intel I7 And AboveRAM: 8 GB
HARD DISK CAPACITY: 1TB

2.2 Software Requirement OPERATING SYSTEM: Windows10PACKAGE: Anaconda
FRONT END: Python

2.3 About the Software

The tools used in this project are listed below.

- Anaconda
- Jupyter Notebook

2.3.1 Anaconda

Rather to using command lines, Anaconda Navigator's graphical interface can be usedto install packages, manage environments, and run standard Python tasks. Moreover, it enables easy management of channels, environments, and packages for Conda withoutthe need for command-line input. On the Anaconda Cloud or in a local Anaconda Repository, Navigator can search for packages. Windows, macOS, and Linux are all supported**.**

2.3.2 Jupyter Notebook

An open-source web programme called the Jupyter Notebook allows users to create andshare documents with live code, equations, visualizations and text. The folks who workon Project Jupyter maintain Jupyter Notebook.

The IPython project, which once had its own IPython Notebook project, is where Jupyter Notebooks got their start. Jupyter's name is derived from the three primary programming languages it supports: Julia, Python, and R. There are already more than100 additional kernels available, however Jupyter comes with the IPython kernel, whichenables Python programme writing.

Python Package

a) Scikit-Learn

Scikit-learn, a free Python package that is frequently seen as a direct extension of SciPy,

is based on NumPy and SciPy. It is specially made for creating supervised and unsupervised machine learning algorithms and data modeling. Scikit-learn is user-friendly and beginner-friendly because of its straightforward, intuitive, and consistent interface. Scikit-learn performs admirably by enabling users to alter and exchange data as they need, despite the fact that its utility is constrained becauseit only excels at data modeling.

b) Pandas

Python's Pandas package for data research and analysis enables programmers to create simple, seamless high-level data structures. Pandas, which is based on NumPy, is in charge of getting data sets and data points ready for machine learning. Pandas uses one-dimensional (series) and two-dimensional (Data Frame) data structures. These two types of data structures allow Pandas to be used in a range of industries, from science and statistics to banking and engineering.

Due to its adaptability, the Pandas Library can be used with other scientific and numerical libraries. Because they are rapid, compliant, and highly descriptive, their data structures are simple to use. By aggregating, integrating, and re-indexing data with Pandas, one can modify data functionality with a minimum of keystrokes.

c) Matplotlib

Matplotlib is a data visualization library that is used for making plots and graphs. It isan extension of SciPy and is able to handle NumPy data structures as well as complex data models made by Pandas. Although it is expertise is limited to 2D plotting, Matplotlib can produce high-quality and publish-ready diagrams, graphs, plots, histograms, error charts, scatter plots, and bar charts.

Matplotlib is intuitive and easy to use, making it a great choice for beginners. It is eveneasier to use for people with pre-existing knowledge in various other graph-plotting tools. It offers GUI toolkit support, including python, Tkinter, and Qt.

d) NumPy

Open-source and well-known Python library for numbers, NumPy. It can carry out a wide range of mathematical operations on matrices and arrays. One of the most popular libraries for scientific computing, it is frequently used by scientists to analyzedata. It is perfect for machine learning and artificial intelligence (AI) projects since it can process multidimensional arrays, handle linear algebra, and perform Fourier transformation. NumPy arrays demand a considerable reduction in storage space when compared to standard Python lists. They are also a lot easier to operate and considerably faster than the earlier.One can reshape, transpose, and modify data in matrix form with NumPy. Combining NumPy'scapabilities makes it easy to enhance the machine learning model's performance.

e) Seaborn

An open-source Python package for data visualization and graphing is called Seaborn.It uses sophisticated Pandas data structures and is based on the graphing software Matplotlib. Seaborn offers a high-level, feature-rich interface for creating precise, illuminating statistical graphs on its own. Because it can produce logical graphs of learning and execution data, it is employed in machine learning and deep learning applications.The most beautiful and eye-catching graphs and plots are produced by Seaborn, which makes it ideal for use in publishing and marketing. Seaborn can also save you time and effort because it enables you to build complex graphs with little code and basic instructions.

f) train-test split

The train-test split is used to determine how well machine learning algorithms work when employed with prediction-based methods and applications. To compare the output of one's own machine learning model, one can use this quick and simple procedure.

3. REVIEW OF LITERATURE

Table 3.1 Review of Datasets and techniques used for Detection of Cross site Scripting attacks using machine algorithms

Sl.No	Title of the paper	Authors & Year	Algorithms used	Dataset used	Results Accuracy level (%)
1.	Detection of XSS in web applications using Machine Learning Classifiers [1]	Raima Banerjee,Aritra Baksi,Nidhi Singh, Soham Kanti Bishnu(2020)	SVM, KNN, Random Forest and Logistic Regression	dataset of 1611 instances of JavaScript with 158 as malicious and 1453 as benign.	96.9%
2.	The Detecting CrossSite Scripting (XSS)Using Machine Learning Methods [2]	Stanislav Kascheev, Tatyana Olenchikova (2020)	Decision Tree Classifier, SVM, Logistic Regression,Naives Bayes	Malicious Scripts	99.9%
3.	Machine Learning based Cross-site Scripting Detection in Online Social Networks [3]	Rui Wang, Xiaoqi Jia, Qinlei Li, Shengzhi Zhang	ADTree and AdaBoost Classifiers	DMOZ database	**94%**
4.	A Comparative Analysis of	Shaimaa Khalifa Mahmoud, Marco Alfonse, Mohamed	open-source code guidelines from different resources		-

	Cross Site Scripting (XSS) Detecting and Defensive Techniques [4]	Ismail Roushdy, Abdel-Badeeh M. Salem	such as OWASP Secure Coding Practices", "SANS TOP 25 Most Dangerous Software Errors" and “CERT Coding Standards"	---	
5.	Detection and Prevention of Cross-site Scripting Attack with Combined Approaches [5]	Hsing-Chung Chen1,2,*, (Senior Member, IEEE), Aristophane Nshimiyimana1	Random Forest (RF), Logistic Regression (LR), k-Nearest Neighbors (k-NN), and Support Vector Machine (SVM), we propose a new approach that combines the Web Application Firewall (WAF), Intrusion Detection System (IDS), and Intrusion Prevention System (IPS) to detect and prevent XSS attack in real-time.	Cross-Site Script dataset used in our research study was obtained from the GitHub website [5] and contains 24096 rows and 66 columns of scripts, with 14096 labeled as benign and 10000 labeled as malicious.	**99.9%**

6.	XSS Attack Detection with Machine Learning and n-Gram Methods [6]	Gulit Habibi, Nico Surantha(2020)	SVM, Naive Bayes, and KNN, N-gram Method Implementation	400 URL data	99%
7.	A Hybrid Machine Learning Approach for Detecting Cross-Site Scripting Attacks [7]	Nusrat Jahan and Md. Hasanul Kabir(2021)	reflected XSS attack detection tool, called XSS-Guard, on Ubuntu 20.04 LTS Operating system by using python3	Payloads	99%
8.	Cross-site scripting attack detection based on convolution neural network [8]	Huyong Yan1,2,3,4, Li Feng5 , You Yu6 , Weiling Liao5 , Lei Feng7,8*, Jingyue Zhang4 , Dan Liu9(2022)	XSS-Guard helps to determine vulnerable websites	dataset from a csv file where payloads and labels are stored	**97%**
9.	A Comprehensive Inspection Of Cross Site Scripting Attack [9]	Mohit Dayal, Nanhay Singh, Ram Shringar Raw (2016)	N- Stalker It is a web scanning tool which is used for the detection of XSS vulnerabilities and other known attacks.	Malicious scripts and Benign	**----**
10.	Efficient Malicious Code Detection Using	Junho Choi, Hayoung Kim, Chang Choi, Pankoo Kim(2011)	N- Gram and SVM		99%

	N-Gram Analysis and SVM [10]				
11.	An implementation of real-time detection of cross-site scripting attacks on cloud-based web applications using deep learning [11]	Isaac Odun-Ayo1 , Williams Toro-Abasi2 , Marion Adebiyi3, and Oladapo Alagbe4(2021)	MLP deep learning model	1,000 SNS web pages	98.99%

Raima Banerjee,Aritra Baksi,Nidhi Singh, Soham Kanti Bishnu – In this paper, we used machine learning classifiers to except features that may be employed in detecting all the vulnerabilities that cross-site scripting poses at web content and applications. We are going to come up with a processed dataset that will be useful for all different future purposes and relishes the active participation of omitting the liabilities

Stanislav Kascheev, Tatyana Olenchikova - The article explores the use of machine learning methods for building classifiers that allow detecting XSS in JavaScript. Currently, the focus is on research on passive or active XSS, where a malicious script is entered into a web application and stored in a database.

Rui Wang, Xiaoqi Jia, Qinlei Li, Shengzhi Zhang - In this paper, we present a novel approach using machine learning to do XSS detection in OSN. Firstly, we leverage a new method to capture identified features from webpages and then establish classification models which can be used in XSS detection. Secondly, we propose a novel method to simulate XSS worm spreading and build our webpage database. Finally, we setup experiments to verify the classification models using our test database.

Shaimaa Khalifa Mahmoud, Marco Alfonse, Mohamed Ismail Roushdy, Abdel-Badeeh M. Salem - This study discusses the XSS attack, its taxonomy, and its incidence. In addition, the paper presents the XSS mechanisms used to detect and prevent the XSS attacks**.**

Hsing-Chung Chen1,2,*, (Senior Member, IEEE), Aristophane Nshimiyimana1 - Our research combined different algorithms to find the best solution to detect and prevent the Cross-Site Scripting attack in real-time. To achieve our research target, we implemented RF, LR, k-NN, and SVM (linear and poly kernels) as artificial intelligence algorithms to discover, classify and identify XSS malicious and benign script code in the dataset.

Gulit Habibi, Nico Surantha - The machine learning algorithm is then equipped with the n-gram method to each script feature to improve the detection performance of XSS attacks. The simulation results show that the SVM and n-gram method achieves the highest accuracy with 98%.

Nusrat Jahan and Md. Hasanul Kabir - Our model uses a scanner to discover vulnerabilities in a website and convolutional neural networks to predict the most common vulnerabilities that may be used for reflected XSS attacks, which makes the proposed model hybrid. We analysed the model experimentally. Analysis results show that the proposed model is able to detect vulnerable attack surfaces with 99 % accuracy.

Huyong Yan1,2,3,4, Li Feng5 , You Yu6 , Weiling Liao5 , Lei Feng7,8*, Jingyue Zhang4 , Dan Liu9 –

In this paper, we used almost all the characters of XSS Scripts during feature generation and used the Convolutional Neural Network (CNN) technique to classify and detect the XSS scripts as malicious or benign and achieved the accuracy of 98.62 and precision of 98.6 and recall 98.86.

Mohit Dayal, Nanhay Singh, Ram Shringar Raw - In this paper, we authors have discussed about various impacts of XSS, types of XSS, checked whether the site is vulnerable towards the XSS or not, discussed about various tools for examining the XSS vulnerability and summarizes the preventive measures against XSS.

Junho Choi, Hayoung Kim, Chang Choi, Pankoo Kim - This paper proposes an approach that results in an effective n-gram feature extraction from malicious code for classifying executable as malicious or benign with the use of Support Vector Machines (SVM) as the machine learning classifier.

Isaac Odun-Ayo1 , Williams Toro-Abasi2 , Marion Adebiyi3, and Oladapo Alagbe4 -

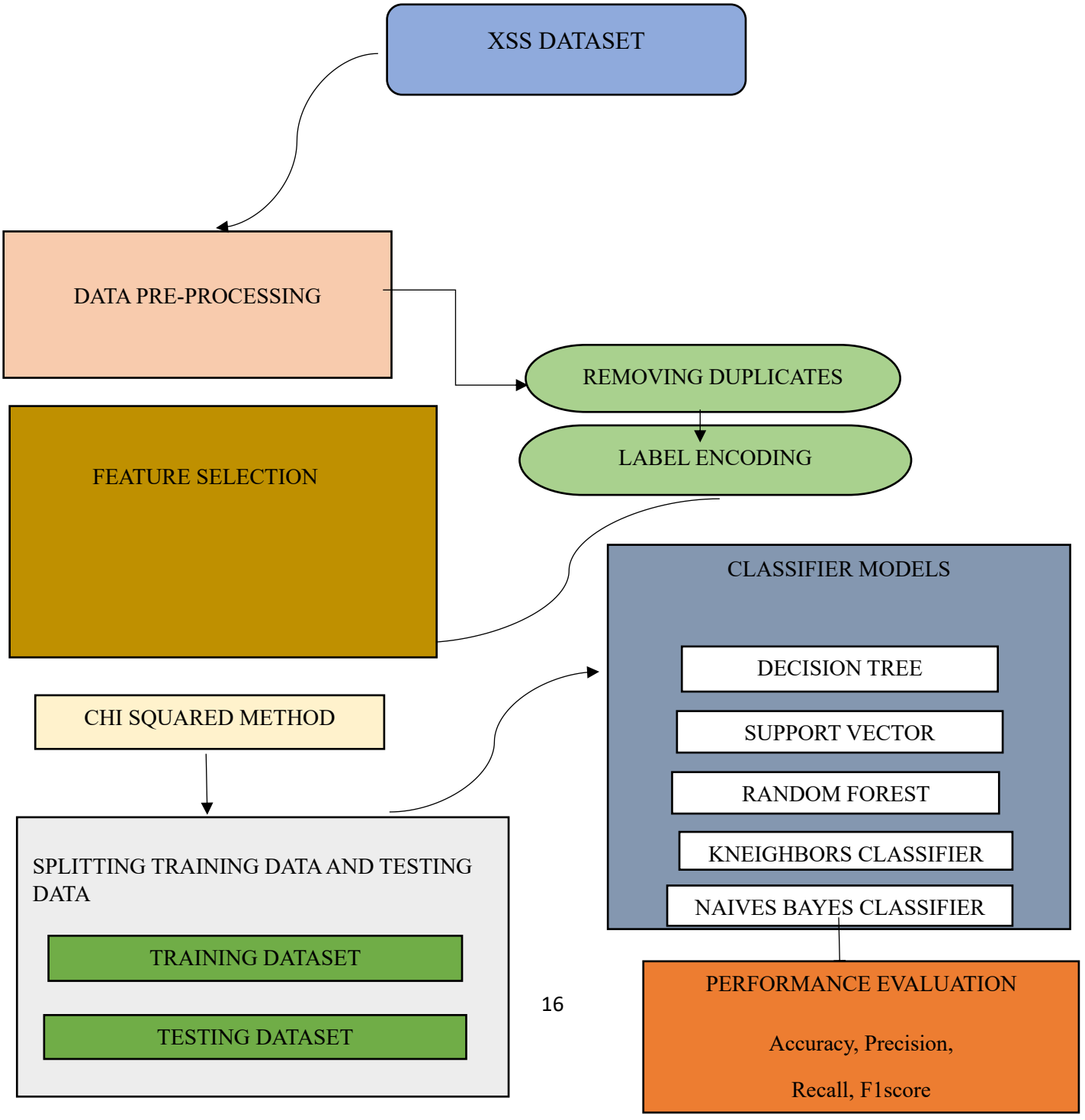
This project work utilized five phases cross-site scripting payloads and Benign user inputs extraction, feature engineering, generation of datasets, deep learning modelling, and classification filter for Malicious cross-site scripting queries.

Observations: In these papers, observations highlight the challenges and opportunities in using machine learning algorithms for XSS detection. They inform researchers and practitioners on areas of improvement, model tuning, feature engineering, and data collection to enhance the performance and robustness of XSS detection systems. This paper is related to calculate the best accuracy and detect the XSS attacks using machine learning models – Decision Tree, SVM, Naives Bayes,Random Forest, KNN

4. METHODOLOGY

The methodology for detecting XSS attacks using machine learning algorithms involves a series of steps that include data collection, pre-processing, model selection, feature selection, model training, model evaluation, and model deployment. Each step may involve statistical techniques to optimize the performance of the machine learning algorithm and ensure effective detection of XSS attacks.

Figure 4.1 Proposed Methodology



PHASE I – DATA IMPORTING

4.1 DATA IMPORTING

Data importing is an important step in the methodology for detecting cross-site scripting (XSS) attacks using machine learning algorithms. In this step, the data is imported into a format that is suitable for preprocessing and analysis. The data may be in different formats, such as CSV files, SQL databases, or JSON objects. Therefore, it is important to choose the appropriate method for importing the data based on the format and the source. After the data has been imported, it is important to look for any missing values. Techniques for cleaning data can be used to remove of duplicates and unnecessary information.

DATASET

XSSDataset1.CSV

XSSDataset1.CSV Description

Dataset Name: XSSDATASET1

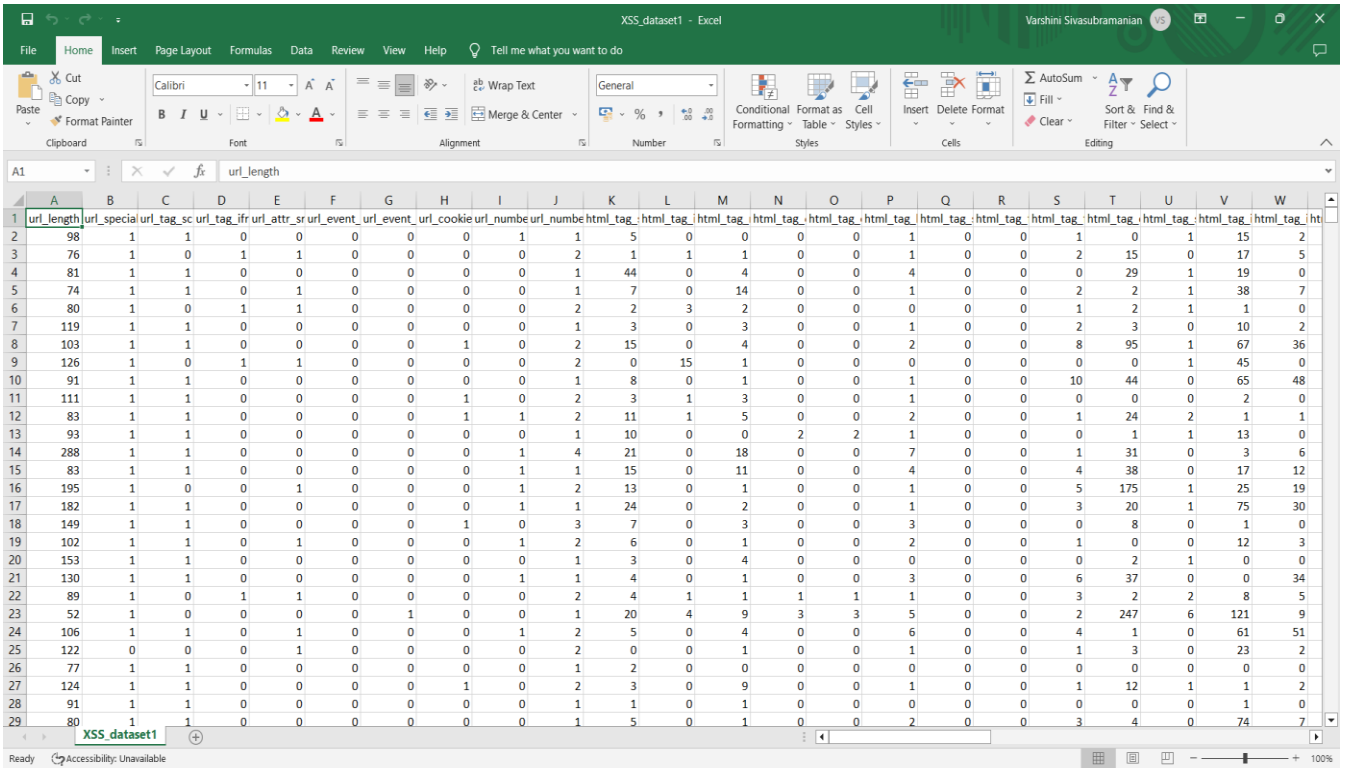
Number of Rows: 100100

Number of Columns: 68

Features present: url_length',
'url_special_characters',
'url_tag_script',
'url_tag_iframe',
'url_attr_src',
'url_event_onload',
'url_event_onmouseover',
'url_cookie',
'url_number_keywords_param',
'url_number_domain',
'html_tag_script',
'html_tag_iframe',
'html_tag_meta',
'html_tag_object',
'html_tag_embed',
'html_tag_link',
'html_tag_svg',
'html_tag_frame',
'html_tag_form',

'html_tag_div',
'html_tag_style',
'html_tag_img',
'html_tag_input',
'html_tag_textarea',
'html_attr_action',
'html_attr_background',
'html_attr_classid',
'html_attr_codebase',
'html_attr_href',
'html_attr_longdesc',
'html_attr_profile',
'html_attr_src',
'html_attr_usemap',
'html_attr_http-equiv',
'html_event_onblur',
'html_event_onchange',
'html_event_onclick',
'html_event_onerror',
'html_event_onfocus',
'html_event_onkeydown',
'html_event_onkeypress',
'html_event_onkeyup',
'html_event_onload',
'html_event_onmousedown',
'html_event_onmouseout',
'html_event_onmouseover',
'html_event_onmouseup',
'html_event_onsubmit',
'html_number_keywords_evil',
'js_file',
'js_pseudo_protocol',
'js_dom_location',
'js_dom_document',
'js_prop_cookie',
'js_prop_referrer',
'js_method_write',
'js_method_getElementsByTagName',
'js_method_getElementById',
'js_method_alert',
'js_method_eval',
'js_method_fromCharCode',
'js_method_confirm',
'js_min_length',
'js_min_define_function',
'js_min_function_calls',
'js_string_max_length',
'html_length', 'Class']

Dataset link: https://figshare.com/articles/dataset/XSS_dataset1_csv/13046138/4?file=24959207



url_length	url_special	url_tag_sc	url_tag_ifr	url_attr_sr	url_event_	url_event_	url_cookie	url_numbe	url_numbe	html_tag_	html_tag_	html_tag_	html_tag_	html_tag_	html_tag_	html_tag_	html_tag_	html_tag_	html_tag_	html_tag_	html_tag_	ht
98	1	1	0	0	0	0	0	1	1	5	0	0	0	0	1	0	0	1	0	1	15	2
76	1	0	1	1	0	0	0	0	2	1	1	1	0	0	1	0	0	2	15	0	17	5
81	1	1	0	0	0	0	0	0	1	44	0	4	0	0	4	0	0	0	29	1	19	0
74	1	1	0	1	0	0	0	0	1	7	0	14	0	0	1	0	0	2	2	1	38	7
80	1	0	1	1	0	0	0	0	2	2	3	2	0	0	0	0	0	1	2	1	1	0
119	1	1	0	0	0	0	0	0	1	3	0	3	0	0	1	0	0	2	3	0	10	2
103	1	1	0	0	0	0	1	0	2	15	0	4	0	0	2	0	0	8	95	1	67	36
126	1	0	1	1	0	0	0	0	2	0	15	1	0	0	0	0	0	0	0	1	45	0
91	1	1	0	0	0	0	0	0	1	8	0	1	0	0	1	0	0	10	44	0	65	48
111	1	1	0	0	0	0	1	0	2	3	1	3	0	0	1	0	0	0	0	0	2	0
83	1	1	0	0	0	0	1	1	2	11	1	5	0	0	2	0	0	1	24	2	1	1
93	1	1	0	0	0	0	0	0	1	10	0	0	2	2	1	0	0	0	1	1	13	0
288	1	1	0	0	0	0	0	1	4	21	0	18	0	0	7	0	0	1	31	0	3	6
83	1	1	0	0	0	0	0	1	1	15	0	11	0	0	4	0	0	4	38	0	17	12
195	1	0	0	1	0	0	0	1	2	13	0	1	0	0	1	0	0	5	175	1	25	19
182	1	1	0	0	0	0	0	1	1	24	0	2	0	0	1	0	0	3	20	1	75	30
149	1	1	0	0	0	0	1	0	3	7	0	3	0	0	3	0	0	0	8	0	1	0
102	1	1	0	1	0	0	0	1	2	6	0	1	0	0	2	0	0	1	0	0	12	3
153	1	1	0	0	0	0	0	0	1	3	0	4	0	0	0	0	0	0	2	1	0	0
130	1	1	0	0	0	0	0	1	1	4	0	1	0	0	3	0	0	6	37	0	0	34
89	1	0	1	1	0	0	0	0	2	4	1	1	1	1	1	0	0	3	2	2	8	5
52	1	0	0	0	0	1	0	0	1	20	4	9	3	3	5	0	0	2	247	6	121	9
106	1	1	0	1	0	0	0	1	2	5	0	4	0	0	6	0	0	4	1	0	61	51
122	0	0	0	1	0	0	0	0	2	0	0	1	0	0	1	0	0	1	3	0	23	2
77	1	1	0	0	0	0	0	0	1	2	0	0	0	0	0	0	0	0	0	0	0	0
124	1	1	0	0	0	0	1	0	2	3	0	9	0	0	1	0	0	1	12	1	1	2
91	1	1	0	0	0	0	0	0	1	1	0	1	0	0	0	0	0	0	0	0	1	0
80	1	1	0	0	0	0	0	0	1	5	0	1	0	0	2	0	0	3	4	0	74	7

PHASE II – DATA PREPROCESSING

4.2 Data Pre-processing

Data pre-processing is a vital phase in machine learning that improves the quality of the data to promote the extraction of valuable insights from the data. Preparing (cleaning and arranging) raw data in orderto make it acceptable for creating and training Machine Learning Models.

It raises reliability and accuracy. Pre-processing data can increase the correctness and quality of a dataset, making it more stable by removing missing or inconsistent data values brought on by human or computer mistake. It ensures consistency in data.

a) Removing Duplicate Values in the dataset

- A dataset contains many instances of a duplicate value. It is frequently discovered when usingExcel to work with huge databases.
- Data processing will be unsuccessful if duplicate records are not eliminated. The goal of this control is to eliminate multiple records from the dataset in order to make it ready for further processing.

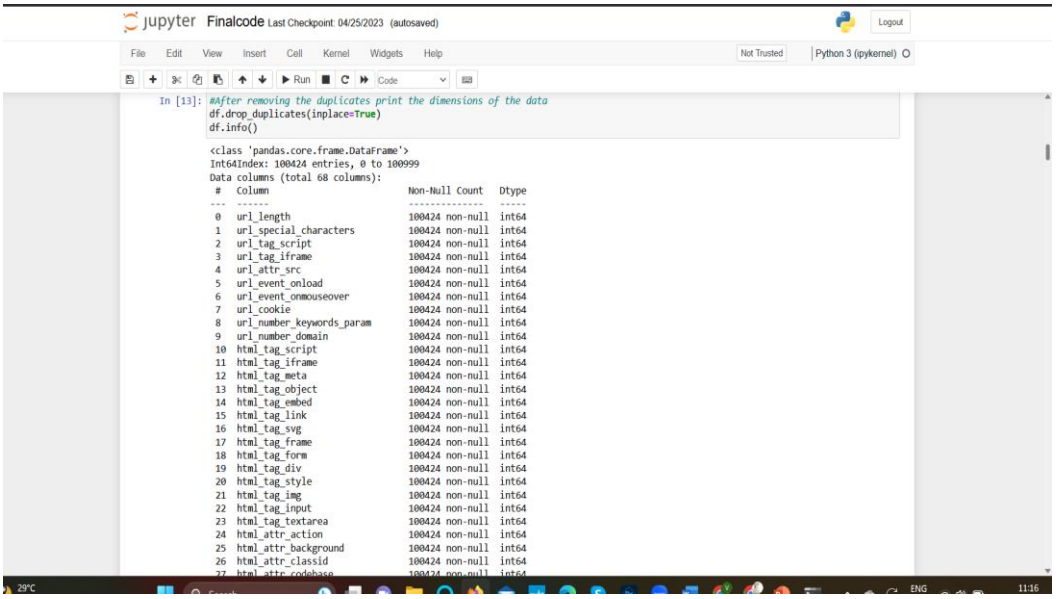


Figure 4.2 Removing Duplicate

b) Label Encoding

- Label encoding is the process of transforming labels into a numeric form so that they may beread by machines.
- Machine learning methods can then be used to determine how well those labels arefunctioning.
- It is a crucial stage in the supervised learning pre-processing of the structured dataset

```
In [11]: from sklearn.preprocessing import LabelEncoder
         le = LabelEncoder()
         df['Class'] = le.fit_transform(df['Class'])
```

Figure 4.2 Label Encoding

c) Informatory Graphs for Any Two Data

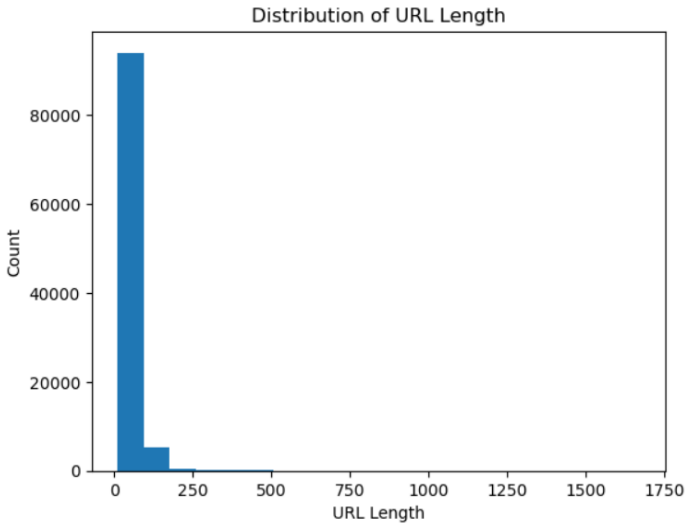


Figure 4.2 URL Length

d) Informatory Graph For any two data.

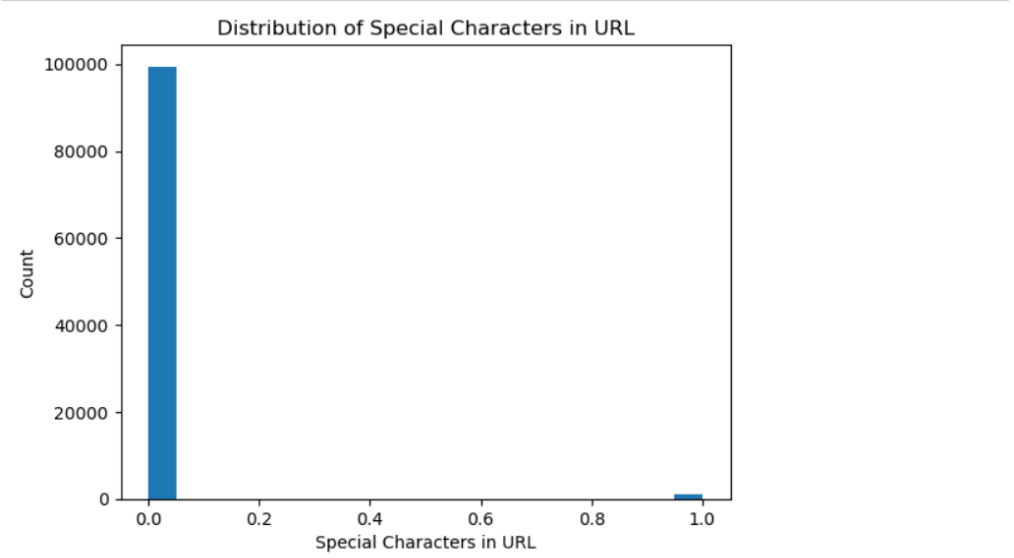


Figure 4.2 Special Characters in URL

PHASE III – FEATURE SELECTION

4.3 Feature selection

The features chosen for training a machine learning model have a significant impact on the model'sperformance. Excessive features may result in the model performing poorly. Therefore, while developing machine learning models, selecting the appropriate set of features is crucial. By choosing the appropriate features, one can speed up detection while simultaneously improving model performance.

These methods come under the following types:

- Filter methods
- Wrapper methods
- Embedded methods
- Hybrid methods

a) Filter Methods

Instead of relying on cross-validation performance, filter approaches have demonstrated the characteristics of the features assessed by univariate statistics. Compared to wrapper methods, these techniques are quicker and more computationally efficient. Using filter methods while working with high- dimensional data is computationally more affordable.

Several prominent filtering methods include

- Information Gain
- Chi-square Test
- Fisher's Score
- Missing Value Ratio

Information Gain

Entropy loss while altering the dataset is referred to as information gain. By computing the information gain of each variable with regard to the target variable, it can be utilised as a feature selectionstrategy.

Chi – square Test

The chi-square test is a method for figuring out how categorical variables are related. Each feature's chi-square value with the target variable is calculated, and the desired number of features with the optimal chi-square value are chosen.

Fisher's Score

The best-supervised feature selection method is Fisher's score. It gives back, in descending order, thevariable's rank on the fisher's criterion. The variables with a high Fisher's score can then be chosen.

Missing Value Ratio

For the purpose of comparing the feature set to the threshold value, the missing value ratio's value can be employed. In order to get the missing value ratio, divide the total number of observations by the number of missing values in each column. The variable can be dropped if it exceeds the cutoff point.

PHASE IV

4.4 MODEL BUILDING (MULTICLASS CLASSIFICATION)

4.4.1 Machine Learning

The most widely used form of artificial intelligence (AI) is machine learning (ML), which has the ability to learn automatically and produce precise and progressing outcomes from experiences. In order to provide a good response to the question and use them to learn on their own, machine learning makes use of the already-existing computations and classification algorithms together with datasets and development programmers. Machine learning analyses enormous amounts of data to produce more accurate results more quickly. Supervised Machine Learning Algorithms, Unsupervised Machine Learning Algorithms, Semi-supervised Machine Learning Algorithms, and Reinforcement Machine Learning Algorithms are four categories into which machine learning algorithms can be divided.

4.4.2 Supervised machine learning algorithms

Classification tasks are used by supervised machine learning techniques to sort the data into labeled data. The dependent variable in these algorithms must be predicted from a predetermined collection of independent variables. As part of the training phase, we employ a function to map a collection of dependent and independent variables to desired outputs up to the desired accuracy is not attained on the training data. Such machine learning techniques are primarily utilized for regression and classification issues. The five most well-known supervised machine learning methods are all applied in the suggested detection model in order to evaluate performance.

K-Nearest Neighbor: One of the simplest, non-parametric, and sluggish learning algorithms, it mostly applies to multi-class problems and is based on instance learning. The distance from a fresh sample's neighbor is used in this technique to categorise it. Additionally, Knearest neighbours are identified from the training dataset, and items are then assigned to the neighborhood's most prevalent class.

Support Vector Machines (SVM): This technique, which may be used for both classification and regression, is the most well-liked among researchers. In comparison to other machine learning classifiers, it can separate the classes independently in n-dimensional space and offer a more precise forecast**.**

Decision Trees: This algorithm aids in decision-making and decision representation, and it may be applied for both classification and regression problems. Node and leaf tree architectures are employed in the Decision Tree algorithm. While the leaf node is used to symbolise the decision, the internal node shows the condition.

Naïve Bayes: This machine learning approach uses the likelihood of any item to predict the outcome and is based on the Bayes Theorem. The Naive Bayes algorithm is a very effective method for dealing with the binary and multi-class classification issue.

Random Forest: This algorithm is a decision tree algorithm that has been upgraded using several decision tree classifiers. The prediction class is indicated by each individual tree in this algorithm, and the end outcome is represented by the prediction class's maximum numbers. The different decision trees used by this classifier are combined into one tree.

Data Classification using Machine Learning

To train and test the detection model during this phase, training and testing datasets were supplied to the machine learning classifier. Six carefully supervised

Support Vector Machines (SVM), Random Forest, Decision Trees, K-Nearest Neighbor, and Naive Bayes methods for classification are examples of machine learning techniques. Data was provided to the Machine Learning Cross site scripting attack for Detection model for additional processing after classification.

Machine Learning Cross site-scripting attack Detection

The machine learning classifier detection model in this phase found the data related to cross site scripting attack. The machine learning classifier and data were used to categorize the data into posts that were authentic or that were part of cyberbullying.

Weighted Average

Weighted average or weighted sum ensemble is an ensemble machine learning technique thataggregates predictions from numerous models, with the weighting of each model's contribution being proportional to its capability or competency. The weighted average ensemble and the voting ensemble are related.In this manner, the averaging strategy is broadened. Each model is given a different weight, indicatingthe importance of that model for making predictions.
Weighted average for quantities (x)i having weights in percentage (P)i% is:

Weighted Average = $\sum$ (P)i% × (x)i

PHASE V

4.5 PERFORMANCE EVALUATION

4.5.1 EVALUATING MODEL PERFORMANCE

A machine learning model's evaluation is essential for evaluation or validation. A machine learning model is evaluated using a variety of measures. To optimize a model based on performance, choosing the bestmetrics is crucial.

The parameters listed below are used for evaluation,

- The Confusion Matrix
- Accuracy
- Precision
- Recall
- F1-score
- Roc Curve

1. **The Confusion Matrix:** The counts of test records that the classification model correctly and incorrectly predicted are used to assess the performance of the model. The confusion matrix gives a more insightful picture of a predictive model's performance, showing which classes are forecasted correctly and incorrectly as well as the kind of errors that are being made.

The formula to calculate the confusion matrix is as follows:

ACTUAL

		NEGATIVE	***POSITIVE***
PREDICTED	***NEGATIVE***	True Negative	False Negative
	POSITIVE	False Positive	True Positive

The confusion matrix consists of four values:

True Positive (TP): The number of instances correctly predicted as positive.

False Negative (FN): The number of instances incorrectly predicted as negative.

False Positive (FP): The number of instances incorrectly predicted as positive.

True Negative (TN): The number of instances correctly predicted as negative.

2. **Accuracy:** The number of accurate predictions made by a machine learning model is known as accuracy.

Accuracy – the percentage of correct responses of the algorithm is calculated by

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN'}$$

In this formula, True Positives (TP) are the instances correctly predicted as positive, True Negatives (TN) are the instances correctly predicted as negative, False Positives (FP) are the instances incorrectly predicted as positive, and False Negatives (FN) are the instances incorrectly predicted as negative.

3. **Precision:** Precision is defined as the ratio of True Positives to all other Positives values, as predicted by machine learning.

$$Precision = TP / (TP + FP)$$

In this formula, True Positives (TP) are the instances correctly predicted as positive, and False Positives (FP) are the instances incorrectly predicted as positive.

4. **Recall:** Recall quantifies the proportion of true positive predictions to total positive predictions and displays the detection rate.

The formula to calculate recall is:

$$Recall = TP / (TP + FN)$$

In this formula, True Positives (TP) are the instances correctly predicted as positive, and False Negatives (FN) are the instances incorrectly predicted as negative.

5. **F1-score:** The harmonic average of Precision and Recall is shown by the F-Score. Precision (P) and Recall (R) combined trends are provided by F-Score. The FScore yields the highest rating when Precision and Recall are identical in value.

$$F1\ score = 2 * (Precision * Recall) / (Precision + Recall)$$

In this formula,

Precision is the proportion of correctly predicted positive instances out of the total instances predicted as positive, and Recall is the proportion of correctly predicted positive instances out of the total actual positive instances.

6. **Mean Absolute Error:** Absolute error in the context of machine learning refers to the size of the discrepancy between the forecast of an observation and its actual value.

The formula to calculate MAE is:

$$MAE = (1/n) * \Sigma|y - y'|$$

In this formula, n represents the number of instances or observations, y refers to the actual values, and y' represents the predicted values.

To calculate the MAE, you take the absolute difference between each predicted value and its corresponding actual value, sum up all these differences, and then divide by the total number of instances. This yields the average absolute difference or the mean absolute error.

7. **Mean Squared Error:** The Mean Squared Error. It is a risk function that corresponds to the squared error loss's expected value. The average, more particularly the mean, of errors squared from data related to a function is used to determine mean square error.

   The formula to calculate MSE is:

   $$MSE = (1/n) * \Sigma (y - y')^2$$

   In this formula,

   n represents the number of instances or observations, y refers to the actual values, and y' represents the predicted values.

   To calculate the MSE, you calculate the squared difference between each predicted value and its corresponding actual value, sum up all these squared differences, and then divide by the total number of instances. This yields the average squared difference or the mean squared error.

8. **Root Mean Squared Error:** One of the methods most frequently used to assess the accuracy of forecasts is root mean square error, also known as root mean square deviation. It illustrates the Euclidean distance between measured true values and forecasts.

   The formula to calculate RMSE:

   $$RMSE = sqrt(MSE)$$

   In this formula,

   MSE represents the Mean Squared Error, which is calculated by taking the average of the squared differences between predicted values and actual values.

By taking the square root of the MSE, you obtain the RMSE, which is expressed in the same units as the target variable.

9. **R-Squared error:** To assess the accuracy of a regression algorithm's predictions, R square is a crucial error statistic. Regression error is measured by R squared (R2), which demonstrates how well the model performed. It shows how well the independent variables may predict the value of the response or target variable

   The formula to calculate R-Squared error:

   $$R^2 = 1 - (SSR / SST)$$

   In this formula,

   SSR – the sum of squared residuals (also known as the sum of squared errors), which is the sum of the squared differences between the predicted values and the actual values.

   SST - SST represents the total sum of squares, which is the sum of the squared differences between the actual values and the mean of the dependent variable.

   R-squared ranges from 0 to 1, with 0 indicating that the independent variables explain none of the variance in the dependent variable, and 1 indicating that the independent variables explain all of the variance

5. RESULTS AND DISCUSSION

In order to analyze the performance of supervised machine learning algorithms in the suggested detection model, this part will examine experimental work, findings, performance metrics, and time complexity additionally; the output of our detection model will be contrasted with results from comparable works.

The performance of the model is typically evaluated using statistical metrics such as accuracy, precision, recall, and F1 score. These metrics help in assessing how well the system detects attacks while reducing false positives. Additionally, the effectiveness of the model in distinguishing between normal and malicious traffic is assessed using the receiver's ROC curves and area under the curve (AUC). The results of the evaluation may show that the machine learning algorithm has a high accuracy in detecting XSS attacks and low false positive rate, which indicates that it can effectively differentiate between normal and malicious traffic. However, it is important to note that the performance of the model may vary depending on the quality of the data and the specific machine learning algorithm used.

The discussion of the findings might focus on how they affect web security and the possibility of further study. For instance, the findings could imply that machine learning algorithms can identify XSS threats and could potentially be used in web security systems. To improve the algorithm's performance and assess its usefulness in practical applications, additional study may be required.

DATASET

url_length	url_specia	url_tag_sc	url_tag_ifi	url_attr_si	url_event	url_event	url_cookie	url_numb	url_numb	html_tag_	html_tag_	html_tag_	html_tag_	html_tag_	html_tag_	html_tag_	html_tag_
98	1	1	0	0	0	0	0	1	1	5	0	0	0	0	1	0	0
76	1	0	1	1	0	0	0	0	2	1	1	1	0	0	1	0	0
81	1	1	0	0	0	0	0	0	1	44	0	4	0	0	4	0	0
74	1	1	0	1	0	0	0	0	1	7	0	14	0	0	1	0	0
80	1	0	1	1	0	0	0	0	2	2	3	2	0	0	0	0	0
119	1	1	0	0	0	0	0	0	1	3	0	3	0	0	1	0	0
103	1	1	0	0	0	0	1	0	2	15	0	4	0	0	2	0	0
126	1	0	1	1	0	0	0	0	2	0	15	1	0	0	0	0	0
91	1	1	0	0	0	0	0	0	1	8	0	1	0	0	1	0	0
111	1	1	0	0	0	0	1	0	2	3	1	3	0	0	1	0	0
83	1	1	0	0	0	0	1	1	2	11	1	5	0	0	2	0	0
93	1	1	0	0	0	0	0	0	1	10	0	0	2	2	1	0	0
288	1	1	0	0	0	0	0	1	4	21	0	18	0	0	7	0	0
83	1	1	0	0	0	0	0	1	1	15	0	11	0	0	4	0	0
195	1	0	0	1	0	0	0	1	2	13	0	1	0	0	1	0	0
182	1	1	0	0	0	0	0	1	1	24	0	2	0	0	1	0	0
149	1	1	0	0	0	0	1	0	3	7	0	3	0	0	3	0	0
102	1	1	0	1	0	0	0	1	2	6	0	1	0	0	2	0	0
153	1	1	0	0	0	0	0	0	1	3	0	4	0	0	0	0	0
130	1	1	0	0	0	0	0	1	1	4	0	1	0	0	3	0	0
89	1	0	1	1	0	0	0	0	2	4	1	1	1	1	1	0	0
52	1	0	0	0	0	1	0	0	1	20	4	9	3	3	5	0	0
106	1	1	0	1	0	0	0	1	2	5	0	4	0	0	6	0	0
122	0	0	0	1	0	0	0	0	2	0	0	1	0	0	1	0	0
77	1	1	0	0	0	0	0	0	1	2	0	0	0	0	0	0	0

5.1 Performance of algorithm.

As per the Experimental Result Decision Tree accuracy(99%),Random forest Accuracy(99%),k-nearest Neighbor(99%), Support Vector machine(99%),Naïve Bayes(47%).

The Decision Tree, Random Forest, K-nearest neighbour and Support Vector machine has the highest accuracy in the result. As per the experimental result, it was found that Random forest and Support vector machine has achieved the best training time, Naïve Bayes, KNN, Decision Tree has obtained the worst training time, Random forest and Support Vector Machine have achieved the best prediction while K-Nearest Neighbor, Naïve Bayes ,Decision tree has obtained the worst prediction .

Algorithm	**Accuracy**	**Precision**	**Recall**	**F1 score**
Decision Tree Algorithm	99%	99%	99%	99%
Random Forest Classifier	99%	99%	99%	99%
KNeighborsClassifier	99%	99%	99%	99%
Naives Bayes	45%	45%	45%	45%
SVM	99%	99%	99%	99%

Table 1: Accuracy of the algorithms

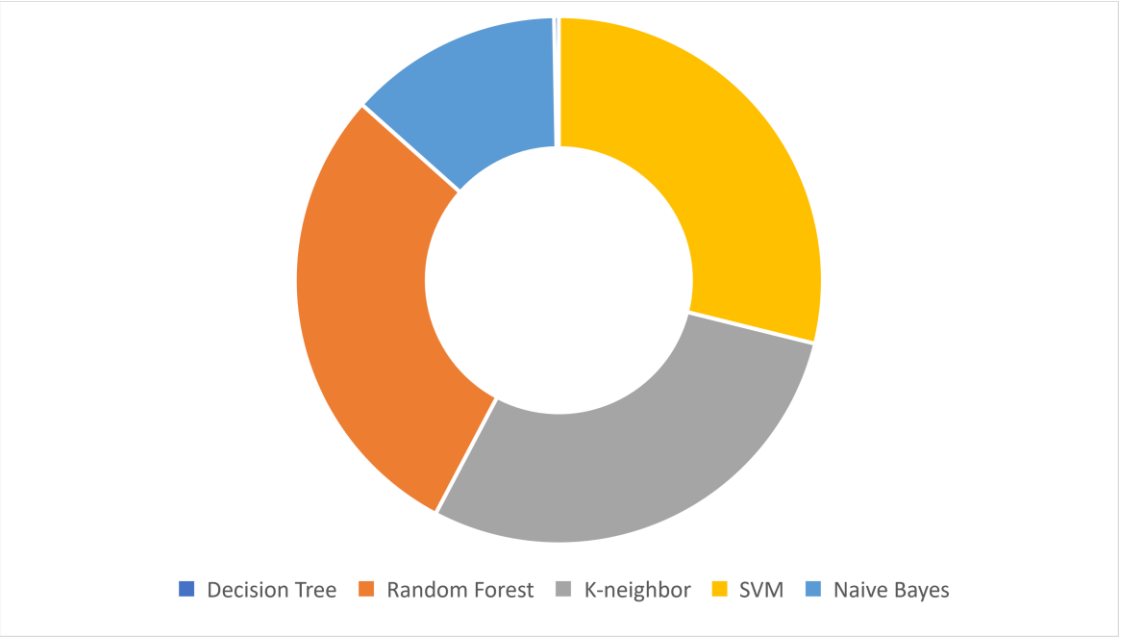


Figure 2: Accuracy

5.2 Comparison of Results

Additionally, the performance outcomes of our suggested strategy were contrasted with those of the researchers' earlier tests.

In terms of accuracy and F-score, our suggested model performed better for the dataset, but recall and precision were nearly comparable.

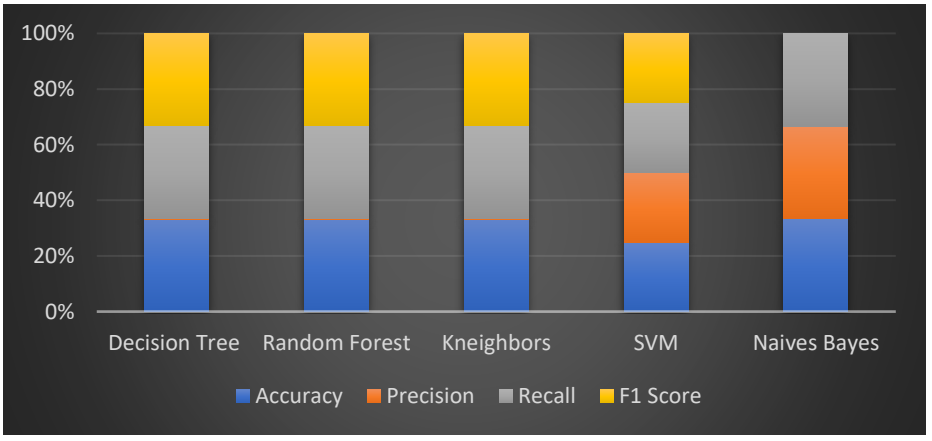


Figure 3: Comparsion

6. CONCLUSION AND FUTURE SCOPE

The project has demonstrated that Decision Tree, SVM, k-NN, and Random Forest can be used to build classifiers for XSS coded in JavaScript giving high accuracy (up to 99.75%) and precision (up to 99.88%) when applied to a large real-world data set. This shows that these classifiers can be added as a security layer either in a browser or (as intended) on a server. The training data was designed to give fair coverage of scripts, including scripts of a variety of lengths and both obfuscated and non-obfuscated scripts. The data is labelled as malicious or benign, rather than using obfuscation as a proxy for maliciousness. Whilst SVM, k-NN, and Random Forest have been used in the experiments, it is expected that other classification methods would also work well. A systematic direct comparison with previous studies is not possible, however, the new classifiers give performance statistics that stand up well. The current study works with a larger and more diverse suite of scripts than many of these previous studies and is the first study to use Random Forests as a classifier for XSS. The key to building successful classifiers is the choice of feature set and how the features are measured. With a large design space, there is motivation to investigate a wide range of approaches to feature selection. The features chosen in this project fall into two categories: firstly the complete set of symbols used in the JavaScript language, and secondly aspects of the scripts that are associated with malicious code. This allows the classifiers to find patterns based on the shape of the program (symbols) and the constructs used (behavioural features). One particularly interesting aspect of this work is that, in contrast to other studies, a binary measure has been used for all features. This has given higher accuracy and precision than earlier experiments using weighted measures. This hints that it may be possible to extract rules from the classifiers that describe malicious scripts. Another interesting aspect is the value of k used in the final experiments is 1. This suggests that malicious scripts might well be singletons that stand apart from clusters of benign scripts.

FUTURE WORK

In future work, this method can get developed using a Deep learning approach. Moreover, this is a detection approach, hence this method can get integrated within real-world security risk prevention systems such as intrusion prevention systems (IPS). Future research work will involve the definition of more DOM-based features that could lead to detection of other code and server-side injection vulnerabilities like SQL and cross-site request forgery attacks. Also, the method could be implemented using other soft computing approaches like genetic algorithm and neural networks

7. **REFERENCES**

1. Raima Banerjee, Aritra Baksi, Nidhi Singh, Soham Kanti Bishnu "Detection of XSS in web applications using Machine Learning Classifiers" in *IEEE Access, **2020***

2. Stanislav Kascheev, Tatyana Olenchikova South Ural State University (national research university), Chelyabinsk, Russia " The Detecting Cross-Site Scripting (XSS) Using Machine Learning Methods" in *IEEE Access, **2021***

3. Wang, R., Jia, X., Li, Q., & Zhang, S. (2014, August). Machine learning based cross-site scripting detection in online social network. In 2014 IEEE Intl Conf on High Performance Computing and Communications, 2014 IEEE 6th Intl Symp on Cyberspace Safety and Security, 2014 IEEE 11th Intl Conf on Embedded Software and Syst (HPCC, CSS, ICESS) (pp. 823-826). IEEE.

4. Mahmoud, S. K., Alfonse, M., Roushdy, M. I., & Salem, A. B. M. (2017, December). comparative analysis of Cross Site Scripting (XSS) detecting and defensive techniques. In 2017 Eighth International Conference on Intelligent Computing and Information Systems (ICICIS) (pp. 36-42). IEEE.

5. Chen, H. C., Nshimiyimana, A., Damarjati, C., & Chang, P. H. (2021, January). Detection and Prevention of Cross-site Scripting Attack with Combined Approaches. In 2021 International Conference on Electronics, Information, and Communication (ICEIC) (pp. 1-4). IEEE.

6. Habibi, G., & Surantha, N. (2020, August). Xss attack detection with machine learning and n-gram methods. In 2020 International Conference on Information Management and Technology (ICIMTech) (pp. 516-520). IEEE.

7. Yan, R., Xiao, X., Hu, G., Peng, S., & Jiang, Y. (2018). New deep learning method to detect code injection attacks on hybrid applications. Journal of Systems and Software, 137, 67-77.

8. Kumar, J., Santhanavijayan, A., & Rajendran, B. (2022, January). Cross site scripting attacks classification using convolutional neural network. In 2022 International Conference on Computer Communication and Informatics (ICCCI) (pp. 1-6). IEEE.

9. Ambedkar, M. D., Ambedkar, N. S., & Raw, R. S. (2016, April). A comprehensive inspection of cross site scripting attack. In 2016 international conference on computing, communication and automation (ICCCA) (pp. 497-502). IEEE.

10. Choi, J., Kim, H., Choi, C., & Kim, P. (2011, September). Efficient malicious code detection using N-gram analysis and SVM. In 2011 14th International Conference on Network-Based Information Systems (pp. 618-621). IEEE.

11. Odun-Ayo, I., Toro-Abasi, W., Adebiyi, M., & Alagbe, O. (2021). An implementation of real-time detection of cross-site scripting attacks on cloud-based web applications using deep learning. Bulletin of Electrical Engineering and Informatics, 10(5), 2442-2453.

12. Gopal R. Chaudhari, Prof. Madhav V. Vaidya, "A survey on security and vulnerabilities of web application", International Journal of Computer Science and Information Technologies (IJCSIT), ISSN: 0975-9646, Vol. 5 (2), pp. 1856-1860, 2014.

13. Katy Anton, Jim Manico, and Jim Bird, "Top 10 proactive controls 2016", OWASP, US, 2016.

14. Jeff Wiliams and Dave Wichers, "Top 10-2017 rc1", OWASP, US, June 30, 2017.

15. Almudena Alcaide Raya, Jorge Blasco Alis, Eduardo Galán Herrero and Agustín Orfila Diaz-Pabón, "Cross-Site Scripting: An overview",Innovations in SMEs and Conducting E-Business: Technologies, Trends, and Solutions, chapter 4, pp. 61-75, BN13: 9781609607654, 2011.

16. "Detecting XSS Attacks with Recurrent Neural Networks" by Juan Manuel Naranjo, published in the Proceedings of the 2018 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining, 2018.

17. "A Comparative Study of Machine Learning Techniques for Detecting Cross-Site Scripting Attacks" by Ali Javed, Muhammad Imran Malik, and Muddassar Farooq, published in the International Journal of Computer Science and Network Security, 2019.

18. "Machine Learning Based Cross-Site Scripting Detection: A Comparative Study" by Ahmet Buğra Turhan and İbrahim Ethem Göksu, published in the Proceedings of the 2019 International Conference on Computer Science and Engineering, 2019.

19. M. Johns, “Script-templates for the content security policy,” J. Inf. Secur. Appl., vol. 19, no. 3, pp. 209–223, 2014, doi: 10.1016/j.jisa.2014.03.007.

20. K. Nagendran, S. Balaji, B. A. Raj, P. Chanthrika, and R. G. Amirthaa, “Web Application Firewall Evasion Techniques,” 2020 6th Int. Conf. Adv. Comput. Commun. Syst. ICACCS 2020, pp. 194–199, 2020, doi: 10.1109/ICACCS48705.2020.9074217.

21. C. Day, “Intrusion prevention and detection systems,” Manag. Inf. Secur. Second Ed., pp. 119–142, 2013, doi: 10.1016/B978-0-12-416688-2.00005-2.

22. S. Han, J. Jung, H. Kim, S. J. Cho, and K. Suh, “Efficient android malware detection using API rank and machine learning,” J. Internet Serv. Inf. Secur., vol. 9, no. 1, pp. 48–59, 2019, doi: 10.22667/JISIS.2019.02.28.048.

23. H.-C. Chen, S.-S. Kuo, "Active Detecting DDoS Attack Approach Based on Entropy Measurement for the Next Generation Instant Messaging App on Smartphones," Intelligent Automation and Soft Computing (Autosoft Journal), Vol. 25, No. 1, pp. 217-228, 2019.

8. APPENDIX

8.1 SAMPLE CODING

#Import Libraries

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

#Creating Dataframes

```
df = pd.read_csv("XSS_dataset1.csv")
```

#2. Print the original dimensions of the data

```
df.info()
```

#3. After removing the duplicates print the dimensions of the data

```
df.drop_duplicates(inplace=True)
df.info()
```

#4. Generate any 2 informatory graphs for the data

```
plt.hist(df['url_length'], bins=20)
plt.title('Distribution of URL Length')
plt.xlabel('URL Length')
plt.ylabel('Count')
plt.show()
```

Distribution of Special Characters in URL

```
plt.hist(df['url_special_characters'], bins=20)
plt.title('Distribution of Special Characters in URL')
plt.xlabel('Special Characters in URL')
plt.ylabel('Count')
```

```
plt.show()
```

##Label Encoding

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df['Class'] = le.fit_transform(df['Class'])
```

#5. After feature selection display the top 20 features selected

```
X = df.drop('Class', axis=1)
y = df['Class']
from sklearn.feature_selection import f_classif
f_value = f_classif(X, y)
feature_names=list(df.columns)
feature_names
```

#Feature Selection

```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
import numpy as np
test = SelectKBest(score_func=chi2, k=4)
fit = test.fit(X, y)
```

 # Summarize scores

```
np.set_printoptions(precision=3)
print(fit.scores_)
features = fit.transform(X)
```

 # Summarize selected features

```
print(features[0:5,:])
```

#6. Then apply that selected features for model building

```
del df['url_length']

del df['url_tag_iframe']

del df['url_event_onload']

del df['url_event_onmouseover']

del df['url_cookie']

del df['html_tag_frame']

del df['html_tag_div']

del df['html_attr_action']

del df['html_attr_classid']

del df['html_attr_codebase']

del df['html_attr_longdesc']

del df['html_attr_src']

del df['html_attr_usemap']

del df['html_event_onkeydown']

del df['html_event_onmouseup']

del df['html_event_onsubmit']

del df['html_number_keywords_evil']

del df['js_prop_referrer']

del df['js_method_confirm']


feature_names=list(df.columns)


cdf = df[['url_special_characters', 'url_tag_script', 'url_attr_src', 'url_number_keywords_param',
```

'url_number_domain', 'html_tag_script', 'html_tag_iframe', 'html_tag_meta', 'html_tag_object', 'html_tag_embed', 'html_tag_link', 'html_tag_svg', 'html_tag_form', 'html_tag_style', 'html_tag_img', 'html_tag_input', 'html_tag_textarea', 'html_attr_background', 'html_attr_href', 'html_attr_profile', 'html_attr_http-equiv', 'html_event_onblur', 'html_event_onchange', 'html_event_onclick', 'html_event_onerror', 'html_event_onfocus', 'html_event_onkeypress', 'html_event_onkeyup', 'html_event_onload', 'html_event_onmousedown', 'html_event_onmouseout', 'html_event_onmouseover', 'js_file', 'js_pseudo_protocol', 'js_dom_location', 'js_dom_document', 'js_prop_cookie', 'js_method_write', 'js_method_getElementsByTagName', 'js_method_getElementById', 'js_method_alert', 'js_method_eval', 'js_method_fromCharCode', 'js_min_length', 'js_min_define_function', 'js_min_function_calls', 'js_string_max_length', 'html_length', 'Class']]

x = cdf.iloc[:, :48]

y = cdf.iloc[:, -1]

#7. Develop 4 classification models

 # Split the data into training and testing sets

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, X_test.shape, y_train.shape, y_test.shape

print("Training samples: {}; Test samples: {}".format(len(X_train), len(X_test)))

##Decision Tree Multi-class classification

from sklearn.tree import DecisionTreeClassifier # Import Decision Tree Classifier

from sklearn.model_selection import train_test_split # Import train_test_split function

from sklearn import tree

from sklearn import metrics

 # Train the decision tree

```
dtree = tree.DecisionTreeClassifier(criterion='gini', max_depth=10, random_state=0)
dtree = dtree.fit(X_train, y_train)
dtree
```

 # Use the model to make predictions with the test data

```
y_pred = dtree.predict(X_test)
print(y_pred)
print(y_test)
df=pd.DataFrame({'Actual':y_test, 'Predicted':y_pred})
df
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import f1_score
from sklearn.metrics import auc
from matplotlib import pyplot
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt
```

#Model Accuracy, how often is the classifier correct?

```
print("Decision Tree Classifier Accuracy:",metrics.accuracy_score(y_test, y_pred))
print('\nPrecision:',metrics.precision_score(y_test, y_pred, average='micro'))
print('\nRecall: ',metrics.recall_score(y_test, y_pred, average='micro'))
```

 # how did our model perform

```
count_misclassified = (y_test != y_pred).sum()
print('Misclassified samples: {}'.format(count_misclassified))
accuracy = metrics.accuracy_score(y_test, y_pred)
#print('Accuracy: {:.2f}'.format(accuracy))
print("Confusion Matrix:\n", metrics.confusion_matrix(y_test, y_pred))
from sklearn import metrics
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
print("R-Squared:", metrics.r2_score(y_test,y_pred))
print ("F1 score:", metrics.f1_score(y_test, y_pred, average='micro'))
from sklearn.metrics import classification_report, confusion_matrix
print("Confusion Matrix:\n",confusion_matrix(y_test, y_pred))
print("Classification Report:\n",classification_report(y_test, y_pred))
```

#Random Forest

```
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(n_estimators = 100)
rf = clf.fit(X_train, y_train)
forest_y_pred = clf.predict(X_test)
score = accuracy_score(y_test, forest_y_pred) * 100
rounded_score = round(score, 1)
print("Random Forest (n_est: 100) Accuracy: {}%".format(rounded_score))
print('\nPrecision:',metrics.precision_score(y_test, forest_y_pred, average = 'micro'))
```

```
print('\nRecall: ',metrics.recall_score(y_test, forest_y_pred, average = 'micro'))
# use the model to make predictions with the test data
forest_y_pred = rf.predict(X_test)
print(forest_y_pred)
print(y_test)
df=pd.DataFrame({'Actual':y_test, 'Predicted':forest_y_pred})
df
# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test,forest_y_pred))
# how did our model perform
count_misclassified = (y_test != forest_y_pred).sum()
print('Misclassified samples: {}'.format(count_misclassified))
accuracy = metrics.accuracy_score(y_test,forest_y_pred)
print('Accuracy: {:.2f}'.format(accuracy))
print("Confusion Matrix:\n", metrics.confusion_matrix(y_test, forest_y_pred))
from sklearn import metrics
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, forest_y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, forest_y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, forest_y_pred)))
print("R-Squared:", metrics.r2_score(y_test,forest_y_pred))
print ("F1 score:", metrics.f1_score(y_test, forest_y_pred, average = 'micro'))
from sklearn.metrics import classification_report, confusion_matrix
print("Confusion Matrix:\n",confusion_matrix(y_test, forest_y_pred))
print("Classification report:\n",classification_report(y_test, forest_y_pred))
```

```
#KNN
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
knn_y_pred = knn.predict(X_test)
score = accuracy_score(y_test, knn_y_pred) * 100
rounded_score = round(score, 1)
print("KNeighborsClassifier Accuracy: {}%".format(rounded_score))
print('\nPrecision:',metrics.precision_score(y_test, knn_y_pred, average ='micro'))
print('\nRecall: ',metrics.recall_score(y_test, knn_y_pred, average ='micro'))
#using the model to make predictions with the test data
knn_y_pred = knn.predict(X_test)
print(knn_y_pred)
print(y_test)
df=pd.DataFrame({'Actual':y_test, 'Predicted':knn_y_pred})
df
from sklearn import metrics
import numpy as np
# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test,knn_y_pred))
# how did our model perform
count_misclassified = (y_test != knn_y_pred).sum()
print('Misclassified samples: {}'.format(count_misclassified))
accuracy = metrics.accuracy_score(y_test,knn_y_pred)
```

```
print ('Accuracy: {:.2f}'.format(accuracy))
print ("Confusion Matrix:\n", metrics.confusion_matrix(y_test, knn_y_pred))
print ('Mean Absolute Error:', metrics.mean_absolute_error(y_test, knn_y_pred))
print ('Mean Squared Error:', metrics.mean_squared_error(y_test, knn_y_pred))
print ('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, knn_y_pred)))
print ("R-Squared:", metrics.r2_score(y_test,knn_y_pred))
print ("F1 score:", metrics.f1_score(y_test, knn_y_pred, average = 'micro'))
from sklearn.metrics import classification_report, confusion_matrix
print ("Confusion Matrix:\n",confusion_matrix(y_test, knn_y_pred))
print ("Classification Report:\n",classification_report(y_test, knn_y_pred))
```

#SVM

```
from sklearn.svm import SVC
svm = SVC(probability=True)
svm.fit(X_train, y_train)
svm_y_pred = svm.predict(X_test)
score = accuracy_score(y_test, svm_y_pred) * 100
rounded_score = round(score, 1)
print("SVM Classifier Accuracy: {}%".format(rounded_score))
print('\nPrecision:',metrics.precision_score(y_test, svm_y_pred, average ='micro'))
print('\nRecall: ',metrics.recall_score(y_test, svm_y_pred, average ='micro'))
```

#use the model to make predictions with the test data

```
svm_y_pred = svm.predict(X_test)
print(svm_y_pred)
print(y_test)
```

```
df=pd.DataFrame({'Actual':y_test, 'Predicted':svm_y_pred})
df
from sklearn import metrics
import numpy as np
# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test,svm_y_pred))
# how did our model perform
count_misclassified = (y_test != svm_y_pred).sum()
print('Misclassified samples: {}'.format(count_misclassified))
accuracy = metrics.accuracy_score(y_test,svm_y_pred)
print('Accuracy: {:.2f}'.format(accuracy))
print("Confusion Matrix:\n", metrics.confusion_matrix(y_test, svm_y_pred))
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, svm_y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, svm_y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, svm_y_pred)))
print("R-Squared:", metrics.r2_score(y_test,svm_y_pred))
print ("F1 score:", metrics.f1_score(y_test, svm_y_pred, average = 'micro'))
from sklearn.metrics import classification_report, confusion_matrix
print("Confusion Matrix:\n",confusion_matrix(y_test, svm_y_pred))
print("Classification Report:\n",classification_report(y_test, svm_y_pred))
#Naive Bayes
from sklearn.naive_bayes import GaussianNB
nclf = GaussianNB()
nclf.fit(X_train, y_train)
n_y_pred = nclf.predict(X_test)
```

```
accuracy_score(y_test, n_y_pred, normalize = True)
print (np.count_nonzero(n_y_pred == y_test) / float(y_test.size))
score = accuracy_score(y_test, n_y_pred ) * 100
rounded_score = round(score, 1)
print("Naive Bayes Classifier Accuracy: {}%".format(rounded_score))
print('\nPrecision:',metrics.precision_score(y_test, n_y_pred, average = 'micro'))
print('\nRecall: ',metrics.recall_score(y_test, n_y_pred, average = 'micro'))
# use the model to make predictions with the test data
n_y_pred = nclf.predict(X_test)
print(n_y_pred)
print(y_test)
df=pd.DataFrame({'Actual':y_test, 'Predicted':n_y_pred})
df
from sklearn import metrics
import numpy as np
# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test,n_y_pred))
# how did our model perform
count_misclassified = (y_test != n_y_pred).sum()
print('Misclassified samples: {}'.format(count_misclassified))
accuracy = metrics.accuracy_score(y_test,n_y_pred)
print('Accuracy: {:.2f}'.format(accuracy))
print("Confusion Matrix:\n", metrics.confusion_matrix(y_test, n_y_pred))
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, n_y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, n_y_pred))
```

```
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, n_y_pred)))

print("R-Squared:", metrics.r2_score(y_test,n_y_pred))

print ("F1 score:", metrics.f1_score(y_test, n_y_pred, average = 'micro'))

from sklearn.metrics import classification_report, confusion_matrix

print("Confusion Matrix:\n", confusion_matrix(y_test, n_y_pred))

print("Classification Report:\n",classification_report(y_test, n_y_pred))
```

8.2 SCREENSHOTS

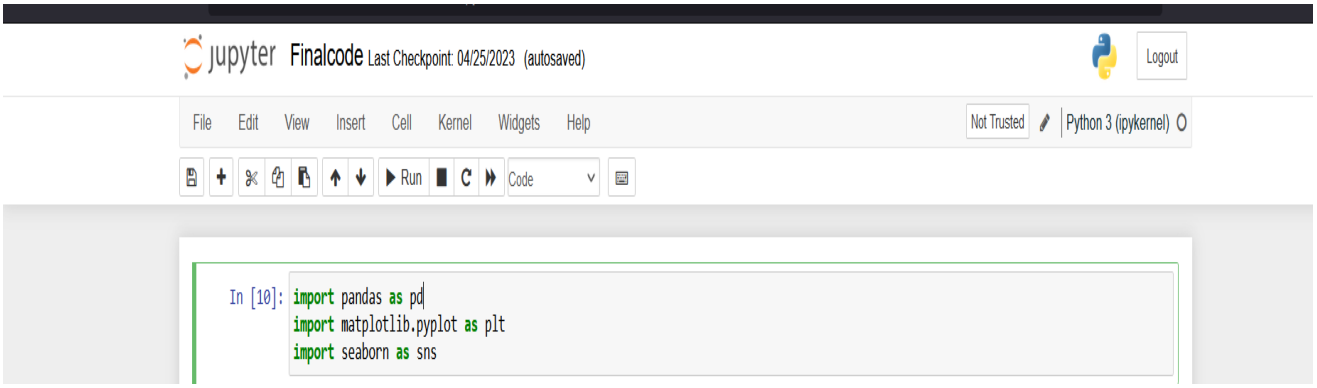


Figure 8.2.1 Importing Libraries

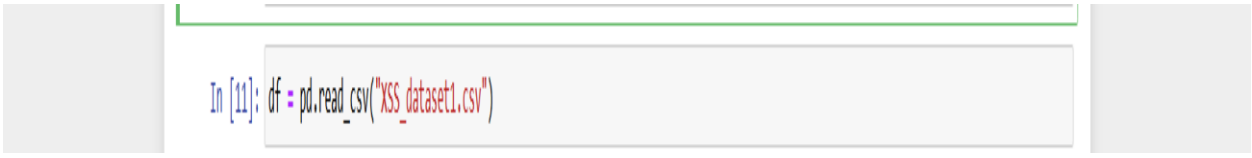


Figure 8.2.2 Reading The CSV File

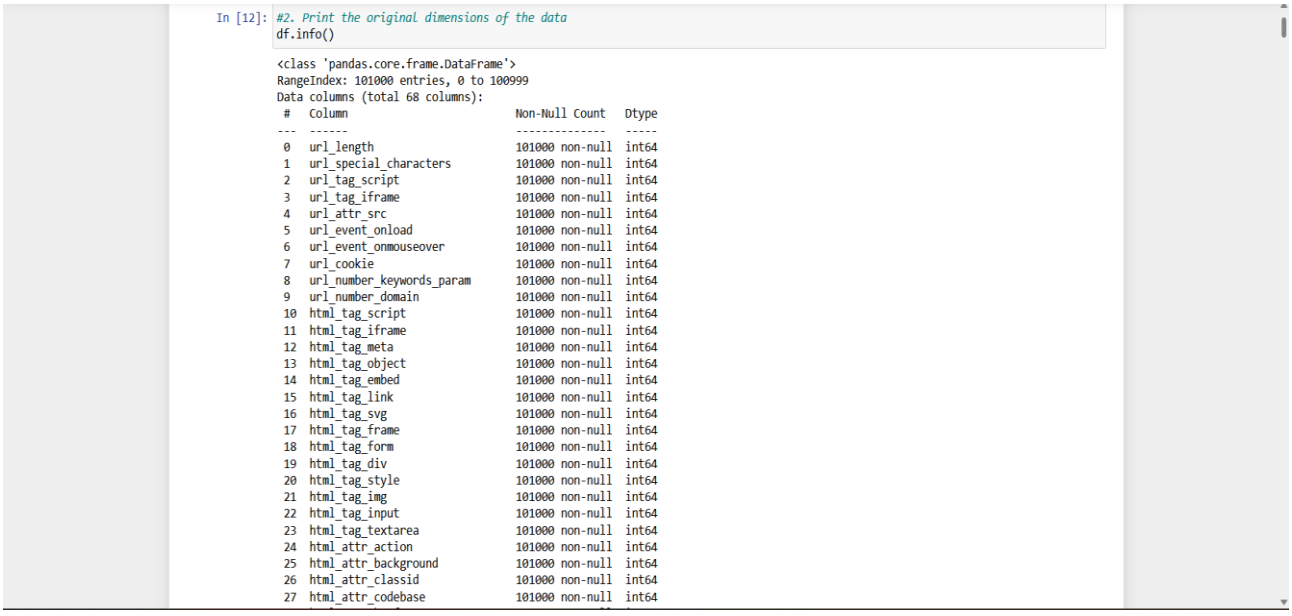


Figure 8.2.3 Printing Original Dimensions of the data

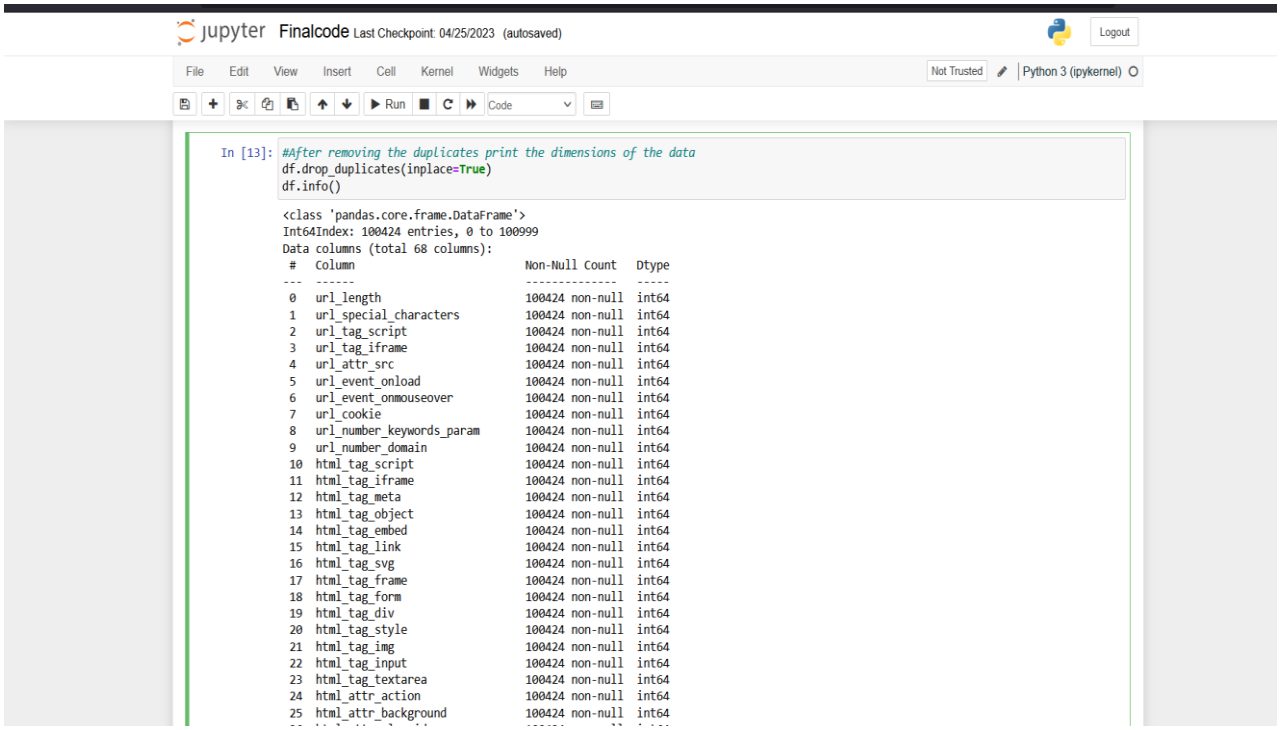


Figure 8.2.4 Removing the duplicates print the dimensions of the data

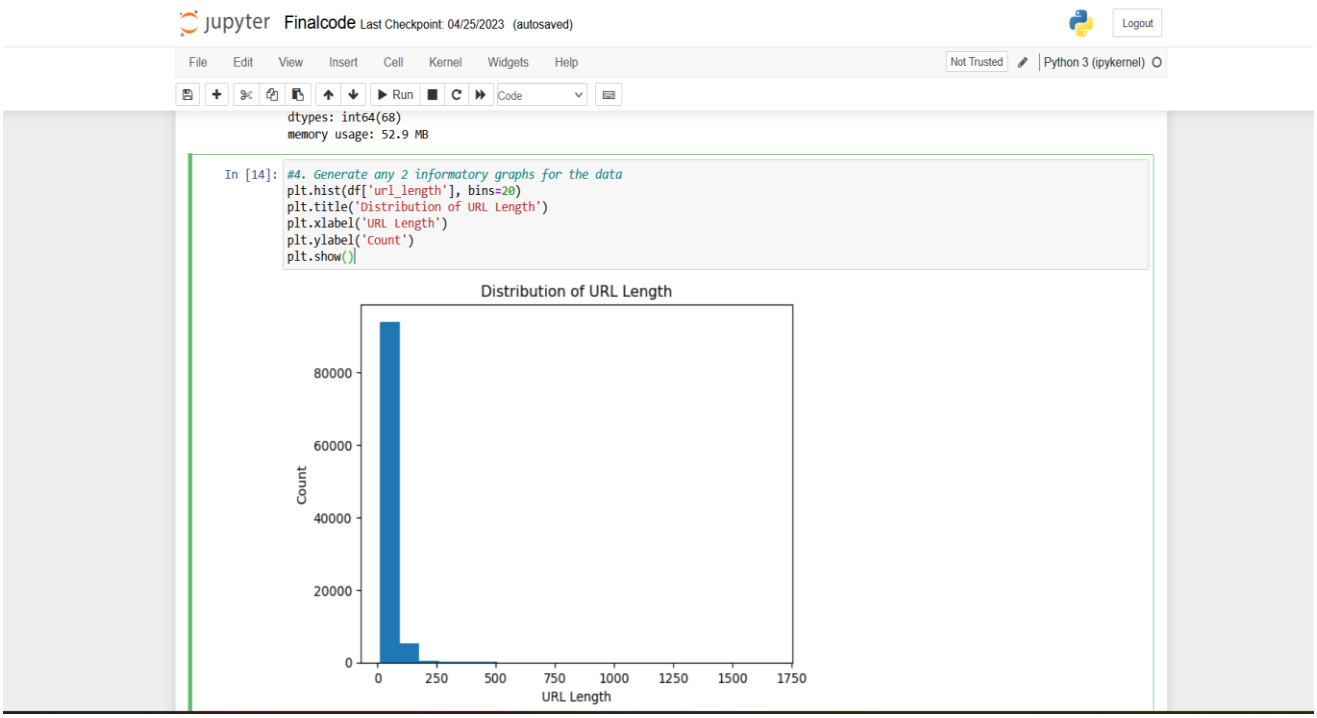


Figure 8.2.5 Generate any 2 informatory graphs for the data

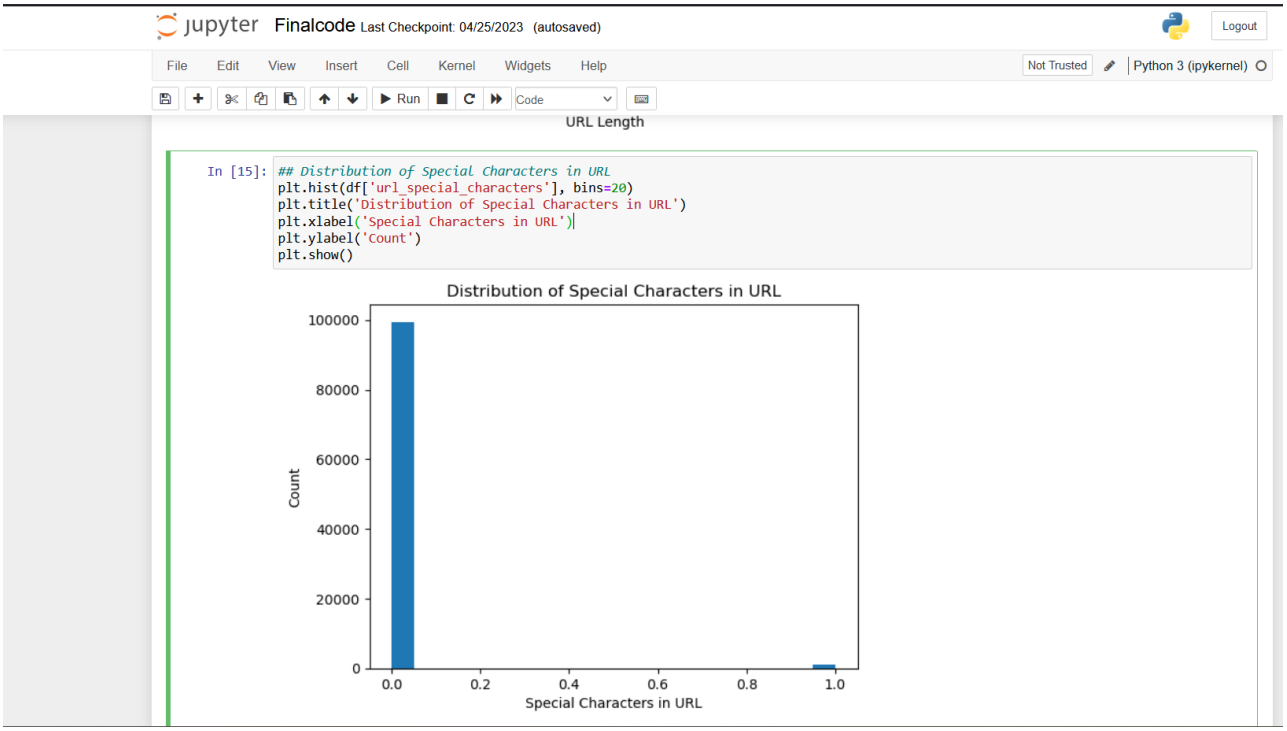


Figure 8.2.6 informatory graphs for the data

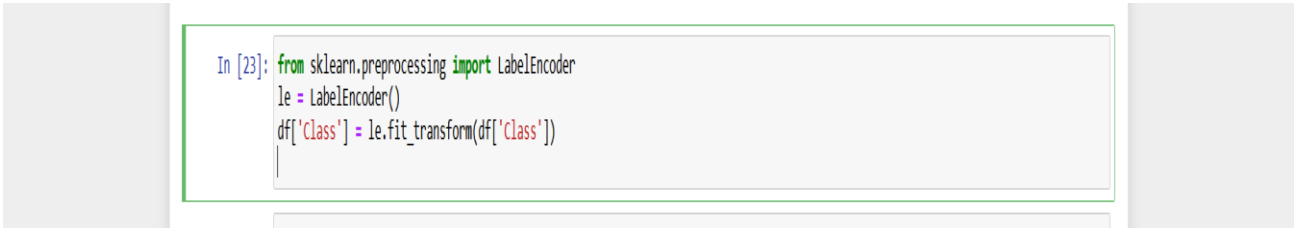


Figure 8.2.7: Label Encoding

```
In [24]: #5. After feature selection display the top 20 features selected

         X = df.drop('Class', axis=1)
         y = df['Class']
         from sklearn.feature_selection import f_classif
         f_value = f_classif(X, y)

         feature_names=list(df.columns)
         feature_names

Out[24]: ['url_length',
          'url_special_characters',
          'url_tag_script',
          'url_tag_iframe',
          'url_attr_src',
          'url_event_onload',
          'url_event_onmouseover',
          'url_cookie',
          'url_number_keywords_param',
          'url_number_domain',
          'html_tag_script',
          'html_tag_iframe',
          'html_tag_meta',
          'html_tag_object',
          'html_tag_embed',
          'html_tag_link',
```

Figure 8.2.8: Feature Selection

```
In [25]: from sklearn.feature_selection import SelectKBest
         from sklearn.feature_selection import chi2
         import numpy as np

In [26]: test = SelectKBest(score_func=chi2, k=4)
         fit = test.fit(X, y)
```

Figure 8.2.9 Chi Square Method

```
In [27]: # Summarize scores
         np.set_printoptions(precision=3)
         print(fit.scores_)

[1.180e+05 8.725e+04 8.274e+04 2.698e+03 4.413e+03 1.299e+03 1.099e+03
 1.809e+04 3.416e+03 2.226e+02 5.085e+03 2.665e+02 7.419e+03 2.189e+02
 9.290e+02 8.838e+03 2.610e+03 1.108e+02 3.232e+01 1.499e+05 8.316e+02
 6.528e+03 1.006e+03 6.655e+01 1.419e+01 2.642e+04 9.797e+02 1.621e+03
 1.141e+05 9.689e-01 1.178e+01 9.300e+03 1.396e+02 1.993e+01 4.534e+02
 3.424e+00 3.185e+03 3.289e+02 5.216e+02 1.837e+01 3.028e+01 2.625e+01
 3.844e+01 9.835e+01 6.123e+03 4.678e+03 1.103e+01 1.950e+01 1.058e+04
 8.575e+01 5.021e+00 6.179e+02 5.133e+01 9.631e+01 1.047e+01 5.198e+01
 6.175e+02 2.564e+00 3.916e+03 4.097e+02 9.876e+01 1.708e+00 1.551e+05
 3.485e-01 5.654e+02 6.998e+06 5.996e+07]
```

Figure 8.2.10 Summarize Scores

```
In [28]: features = fit.transform(X)
         # Summarize selected features
         print(features[0:5,:])
```

```
[[    0    13   275 10712]
 [   15  1879  1879 14735]
 [   29    10   139 15286]
 [    2     1  1231 29001]
 [    2   178   180 10356]]
```

Figure 8.2.11 Summarize Selected Features

```
In [29]: #6. Then apply that selected features for model building

         del df['url_length']
         del df['url_tag_iframe']
         del df['url_event_onload']
         del df['url_event_onmouseover']
         del df['url_cookie']
         del df['html_tag_frame']
         del df['html_tag_div']
         del df['html_attr_action']
         del df['html_attr_classid']
         del df['html_attr_codebase']
         del df['html_attr_longdesc']
         del df['html_attr_src']
         del df['html_attr_usemap']
         del df['html_event_onkeydown']
         del df['html_event_onmouseup']
         del df['html_event_onsubmit']
         del df['html_number_keywords_evil']
         del df['js_prop_referrer']
         del df['js_method_confirm']

         feature_names=list(df.columns)

         cdf = df[['url_special_characters', 'url_tag_script', 'url_attr_src', 'url_number_keywords_param', 'url_number_domain', 'html_tag
         x = cdf.iloc[:, :48]
         y = cdf.iloc[:, -1]
```

Figure 8.2.12 Applying Selected Features for Model Building

```
In [30]: #7. Develop 5 classification models
         # Split the data into training and testing sets
         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, X_test.shape, y_train.shape, y_test.shape
         print("Training samples: {}; Test samples: {}".format(len(X_train), len(X_test)))

         Training samples: 80339; Test samples: 20085
```

Figure 8.2.13 Spliting the Data into Training and Testing Samples for Model Building

```
In [31]: ##Decision Tree Multi-class classification
         from sklearn.tree import DecisionTreeClassifier # Import Decision Tree Classifier
         from sklearn.model_selection import train_test_split # Import train_test_split function
         from sklearn import tree
         from sklearn import metrics

In [32]: # train the decision tree
         dtree = tree.DecisionTreeClassifier(criterion='gini', max_depth=10, random_state=0)
         dtree = dtree.fit(X_train, y_train)
         dtree

Out[32]: DecisionTreeClassifier(max_depth=10, random_state=0)
```

Figure 8.2.14 Decision Tree Multi-class classification

```
In [33]: # use the model to make predictions with the test data
         y_pred = dtree.predict(X_test)
         print(y_pred)
         print(y_test)
         df=pd.DataFrame({'Actual':y_test, 'Predicted':y_pred})
         df

         [0 0 0 ... 0 0 0]
         87020    0
         19286    0
         22217    0
         14933    0
         19870    0
                ..
         9219     0
         3948     0
         66709    0
         10025    0
         36875    0
         Name: Class, Length: 20085, dtype: int64
```

Out[33]:

	Actual	Predicted
87020	0	0
19286	0	0
22217	0	0
14933	0	0
19870	0	0
...	...	...
9219	0	0
3948	0	0
66709	0	0
10025	0	0
36875	0	0

20085 rows × 2 columns

Figure 8.2.15 use the model to make predictions with the test data

In [34]:
```
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import f1_score
from sklearn.metrics import auc
from matplotlib import pyplot
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt
```

In [35]:
```
 #Model Accuracy, how often is the classifier correct?
print("Decision Tree Classifier Accuracy:",metrics.accuracy_score(y_test, y_pred))
print('\nPrecision:',metrics.precision_score(y_test, y_pred, average='micro'))
print('\nRecall: ',metrics.recall_score(y_test, y_pred, average='micro'))
# how did our model perform
count_misclassified = (y_test != y_pred).sum()
print('Misclassified samples: {}'.format(count_misclassified))
accuracy = metrics.accuracy_score(y_test, y_pred)
#print('Accuracy: {:.2f}'.format(accuracy))
print("Confusion Matrix:\n", metrics.confusion_matrix(y_test, y_pred))
```

```
Decision Tree Classifier Accuracy: 0.9990540204132438

Precision: 0.9990540204132438

Recall:  0.9990540204132438
Misclassified samples: 19
Confusion Matrix:
 [[19878     9]
 [   10   188]]
```

Figure 8.2.16 Model Accuracy

```
In [36]: from sklearn import metrics
         print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
         print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
         print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
         print("R-Squared:", metrics.r2_score(y_test,y_pred))
         print ("F1 score:", metrics.f1_score(y_test, y_pred, average='micro'))

         Mean Absolute Error: 0.0009459795867562858
         Mean Squared Error: 0.0009459795867562858
         Root Mean Squared Error: 0.030756781150768783
         R-Squared: 0.9030850060417115
         F1 score: 0.9990540204132438
```

Figure 8.2.17 Metrics

```
In [37]: from sklearn.metrics import classification_report, confusion_matrix
         print("Confusion Matrix:\n",confusion_matrix(y_test, y_pred))
         print("Classification Report:\n",classification_report(y_test, y_pred))

         Confusion Matrix:
          [[19878     9]
          [   10   188]]
         Classification Report:
                       precision    recall  f1-score   support

                    0       1.00      1.00      1.00     19887
                    1       0.95      0.95      0.95       198

             accuracy                           1.00     20085
            macro avg       0.98      0.97      0.98     20085
         weighted avg       1.00      1.00      1.00     20085
```

Figure 8.2.18 Confusion and Classification Report

In [38]:
```
#Random Forest
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(n_estimators = 100)
rf = clf.fit(X_train, y_train)
forest_y_pred = clf.predict(X_test)
score = accuracy_score(y_test, forest_y_pred) * 100
rounded_score = round(score, 1)
print("Random Forest (n_est: 100) Accuracy: {}%".format(rounded_score))
print('\nPrecision:',metrics.precision_score(y_test, forest_y_pred, average = 'micro'))
print('\nRecall: ',metrics.recall_score(y_test, forest_y_pred, average = 'micro'))
```

```
Random Forest (n_est: 100) Accuracy: 99.9%

Precision: 0.9994025392083644

Recall:  0.9994025392083644
```

Figure 8.2.19 Random Forest Classifier

In [39]:
```
# use the model to make predictions with the test data
forest_y_pred = rf.predict(X_test)
print(forest_y_pred)
print(y_test)
df=pd.DataFrame({'Actual':y_test, 'Predicted':forest_y_pred})
df
```

```
[0 0 0 ... 0 0 0]
87020    0
19286    0
22217    0
14933    0
19870    0
        ..
9219     0
3948     0
66709    0
10025    0
36875    0
Name: Class, Length: 20085, dtype: int64
```

Out[39]:

	Actual	Predicted
87020	0	0
19286	0	0
22217	0	0
14933	0	0
19870	0	0
...	...	...
9219	0	0
3948	0	0
66709	0	0
10025	0	0
36875	0	0

20085 rows × 2 columns

Figure 8.2.20 Using model to make predictions with the test data

In [40]:
```
# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test,forest_y_pred))
# how did our model perform
count_misclassified = (y_test != forest_y_pred).sum()
print('Misclassified samples: {}'.format(count_misclassified))
accuracy = metrics.accuracy_score(y_test,forest_y_pred)
print('Accuracy: {:.2f}'.format(accuracy))
print("Confusion Matrix:\n", metrics.confusion_matrix(y_test, forest_y_pred))
```

```
Accuracy: 0.9994025392083644
Misclassified samples: 12
Accuracy: 1.00
Confusion Matrix:
 [[19886     1]
 [   11   187]]
```

Figure 8.2.21 Model accuracy

```
In [41]: from sklearn import metrics
         print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, forest_y_pred))
         print('Mean Squared Error:', metrics.mean_squared_error(y_test, forest_y_pred))
         print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, forest_y_pred)))
         print("R-Squared:", metrics.r2_score(y_test,forest_y_pred))
         print ("F1 score:", metrics.f1_score(y_test, forest_y_pred, average = 'micro'))
         from sklearn.metrics import classification_report, confusion_matrix
         print("Confusion Matrix:\n",confusion_matrix(y_test, forest_y_pred))
         print("Classification report:\n",classification_report(y_test, forest_y_pred))

         Mean Absolute Error: 0.000597460791635549
         Mean Squared Error: 0.000597460791635549
         Root Mean Squared Error: 0.02444301110001689
         R-Squared: 0.9387905301316072
         F1 score: 0.9994025392083644
         Confusion Matrix:
          [[19886     1]
          [   11   187]]
         Classification report:
                       precision    recall  f1-score   support

                    0       1.00      1.00      1.00     19887
                    1       0.99      0.94      0.97       198

             accuracy                           1.00     20085
            macro avg       1.00      0.97      0.98     20085
         weighted avg       1.00      1.00      1.00     20085
```

Figure 8.2.22 Confusion and Classification Report

```
In [42]: #KNN
         from sklearn.neighbors import KNeighborsClassifier
         knn = KNeighborsClassifier(n_neighbors=5)
         knn.fit(X_train, y_train)
         knn_y_pred = knn.predict(X_test)
         score = accuracy_score(y_test, knn_y_pred) * 100
         rounded_score = round(score, 1)
         print("KNeighborsClassifier Accuracy: {}%".format(rounded_score))
         print('\nPrecision:',metrics.precision_score(y_test, knn_y_pred, average ='micro'))
         print('\nRecall: ',metrics.recall_score(y_test, knn_y_pred, average ='micro'))

         C:\Users\HP\anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:228: FutureWarning: Unlike other reduction functio
         ns (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts along. In SciPy 1.11.0, this
         behavior will change: the default value of `keepdims` will become False, the `axis` over which the statistic is taken will be e
         liminated, and the value None will no longer be accepted. Set `keepdims` to True or False to avoid this warning.
           mode, _ = stats.mode(_y[neigh_ind, k], axis=1)

         KNeighborsClassifier Accuracy: 99.0%

         Precision: 0.9901916853373164

         Recall:  0.9901916853373164
```

Figure 8.2.23 KNN Classifier

```
In [43]: #using the model to make predictions with the test data
knn_y_pred = knn.predict(X_test)
print(knn_y_pred)
print(y_test)
df=pd.DataFrame({'Actual':y_test, 'Predicted':knn_y_pred})
df
```

```
C:\Users\HP\anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:228: FutureWarning: Unlike other reduction functio
ns (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts along. In SciPy 1.11.0, this
behavior will change: the default value of `keepdims` will become False, the `axis` over which the statistic is taken will be e
liminated, and the value None will no longer be accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
```

```
[0 0 0 ... 0 0 0]
87020    0
19286    0
22217    0
14933    0
19870    0
        ..
9219     0
3948     0
66709    0
10025    0
36875    0
Name: Class, Length: 20085, dtype: int64
```

Out[43]:

	Actual	Predicted
87020	0	0
19286	0	0
22217	0	0
14933	0	0
19870	0	0
...	...	...
9219	0	0
3948	0	0
66709	0	0
10025	0	0
36875	0	0

20085 rows × 2 columns

Figure 8.2.24 Using model to make predictions with the test data

In [45]:
```
from sklearn import metrics
import numpy as np
# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test,knn_y_pred))
# how did our model perform
count_misclassified = (y_test != knn_y_pred).sum()
print('Misclassified samples: {}'.format(count_misclassified))
accuracy = metrics.accuracy_score(y_test,knn_y_pred)
print('Accuracy: {:.2f}'.format(accuracy))
print("Confusion Matrix:\n", metrics.confusion_matrix(y_test, knn_y_pred))
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, knn_y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, knn_y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, knn_y_pred)))
print("R-Squared:", metrics.r2_score(y_test,knn_y_pred))
print ("F1 score:", metrics.f1_score(y_test, knn_y_pred, average = 'micro'))
from sklearn.metrics import classification_report, confusion_matrix
print("Confusion Matrix:\n",confusion_matrix(y_test, knn_y_pred))
print("Classification Report:\n",classification_report(y_test, knn_y_pred))
```

```
Accuracy: 0.9901916853373164
Misclassified samples: 197
Accuracy: 0.99
Confusion Matrix:
 [[19880     7]
 [  190     8]]
Mean Absolute Error: 0.009808314662683594
Mean Squared Error: 0.009808314662683594
Root Mean Squared Error: 0.09903693585063905
R-Squared: -0.004855463672781335
```

```
Confusion Matrix:
 [[19880     7]
 [  190     8]]
Classification Report:
              precision    recall  f1-score   support

           0       0.99      1.00      1.00     19887
           1       0.53      0.04      0.08       198

    accuracy                           0.99     20085
   macro avg       0.76      0.52      0.54     20085
weighted avg       0.99      0.99      0.99     20085
```

Figure 8.2.25 Metrics and Confusion matrix, Classifiaction report

In [46]:
```
#SVM
from sklearn.svm import SVC
svm = SVC(probability=True)
svm.fit(X_train, y_train)
svm_y_pred = svm.predict(X_test)
score = accuracy_score(y_test, svm_y_pred) * 100
rounded_score = round(score, 1)
print("SVM Classifier Accuracy: {}%".format(rounded_score))
print('\nPrecision:',metrics.precision_score(y_test, svm_y_pred, average ='micro'))
print('\nRecall: ',metrics.recall_score(y_test, svm_y_pred, average ='micro'))
```

```
SVM Classifier Accuracy: 99.0%

Precision: 0.9901418969380135

Recall:  0.9901418969380135
```

Figure 8.2.26 SVM

In [47]:
```
#use the model to make predictions with the test data
svm_y_pred = svm.predict(X_test)
print(svm_y_pred)
print(y_test)
df=pd.DataFrame({'Actual':y_test, 'Predicted':svm_y_pred})
df
```

```
[0 0 0 ... 0 0 0]
87020    0
19286    0
22217    0
14933    0
19870    0
        ..
9219     0
3948     0
66709    0
10025    0
36875    0
Name: Class, Length: 20085, dtype: int64
```

Out[47]:

	Actual	Predicted
87020	0	0
19286	0	0
22217	0	0
14933	0	0
19870	0	0
...	...	...
9219	0	0

Figure 8.2.27 use the model to make prediction with the test data

In [48]:
```
from sklearn import metrics
import numpy as np
# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test,svm_y_pred))
# how did our model perform
count_misclassified = (y_test != svm_y_pred).sum()
print('Misclassified samples: {}'.format(count_misclassified))
accuracy = metrics.accuracy_score(y_test,svm_y_pred)
print('Accuracy: {:.2f}'.format(accuracy))
print("Confusion Matrix:\n", metrics.confusion_matrix(y_test, svm_y_pred))
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, svm_y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, svm_y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, svm_y_pred)))
print("R-Squared:", metrics.r2_score(y_test,svm_y_pred))
print ("F1 score:", metrics.f1_score(y_test, svm_y_pred, average = 'micro'))
from sklearn.metrics import classification_report, confusion_matrix
print("Confusion Matrix:\n",confusion_matrix(y_test, svm_y_pred))
print("Classification Report:\n",classification_report(y_test, svm_y_pred))
```

```
Accuracy: 0.9901418969380135
Misclassified samples: 198
Accuracy: 0.99
Confusion Matrix:
 [[19887     0]
 [  198     0]]
Mean Absolute Error: 0.009858103061986557
Mean Squared Error: 0.009858103061986557
Root Mean Squared Error: 0.09928798045074014
R-Squared: -0.009956252828480894
F1 score: 0.9901418969380135
```

Figure 8.2.28 Metrics

```
Confusion Matrix:
 [[19887     0]
 [  198     0]]
Classification Report:
              precision    recall  f1-score   support

           0       0.99      1.00      1.00     19887
           1       0.00      0.00      0.00       198

    accuracy                           0.99     20085
   macro avg       0.50      0.50      0.50     20085
weighted avg       0.98      0.99      0.99     20085
```

Figure 8.2.29 Confusion Matrix and Classification Report

In [49]:
```
#Naive Bayes
from sklearn.naive_bayes import GaussianNB
nclf = GaussianNB()
nclf.fit(X_train, y_train)
n_y_pred = nclf.predict(X_test)
accuracy_score(y_test, n_y_pred, normalize = True)
print (np.count_nonzero(n_y_pred == y_test) / float(y_test.size))
score = accuracy_score(y_test, n_y_pred ) * 100
rounded_score = round(score, 1)
print("Naive Bayes Classifier Accuracy: {}%".format(rounded_score))
print('\nPrecision:',metrics.precision_score(y_test, n_y_pred, average = 'micro'))
print('\nRecall: ',metrics.recall_score(y_test, n_y_pred, average = 'micro'))
```

```
0.457007717201892
Naive Bayes Classifier Accuracy: 45.7%

Precision: 0.457007717201892

Recall:  0.457007717201892
```

In [50]:
```
# use the model to make predictions with the test data
n_y_pred = nclf.predict(X_test)
print(n_y_pred)
print(y_test)
df=pd.DataFrame({'Actual':y_test, 'Predicted':n_y_pred})
df
```

```
[0 1 0 ... 1 1 0]
87020    0
19286    0
```

Figure 8.2.30 Naive Bayes

In [50]:
```
# use the model to make predictions with the test data
n_y_pred = nclf.predict(X_test)
print(n_y_pred)
print(y_test)
df=pd.DataFrame({'Actual':y_test, 'Predicted':n_y_pred})
df
```

```
[0 1 0 ... 1 1 0]
87020    0
19286    0
22217    0
14933    0
19870    0
        ..
9219     0
3948     0
66709    0
10025    0
36875    0
Name: Class, Length: 20085, dtype: int64
```

Out[50]:

	Actual	Predicted
87020	0	0
19286	0	1
22217	0	0
14933	0	1
19870	0	1
...	...	...
9219	0	1
3948	0	0

Figure 8.2.31 use the model to make predictions with the test data

```
In [51]: from sklearn import metrics
         import numpy as np
         # Model Accuracy, how often is the classifier correct?
         print("Accuracy:",metrics.accuracy_score(y_test,n_y_pred))
         # how did our model perform
         count_misclassified = (y_test != n_y_pred).sum()
         print('Misclassified samples: {}'.format(count_misclassified))
         accuracy = metrics.accuracy_score(y_test,n_y_pred)
         print('Accuracy: {:.2f}'.format(accuracy))
         print("Confusion Matrix:\n", metrics.confusion_matrix(y_test, n_y_pred))

         print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, n_y_pred))
         print('Mean Squared Error:', metrics.mean_squared_error(y_test, n_y_pred))
         print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, n_y_pred)))
         print("R-Squared:", metrics.r2_score(y_test,n_y_pred))
         print ("F1 score:", metrics.f1_score(y_test, n_y_pred, average = 'micro'))
         from sklearn.metrics import classification_report, confusion_matrix
         print("Confusion Matrix:\n", confusion_matrix(y_test, n_y_pred))
         print("Classification Report:\n",classification_report(y_test, n_y_pred))

         Accuracy: 0.457007717201892
         Misclassified samples: 10906
         Accuracy: 0.46
         Confusion Matrix:
          [[ 8990 10897]
          [    9   189]]
         Mean Absolute Error: 0.5429922827981081
         Mean Squared Error: 0.5429922827981081
         Root Mean Squared Error: 0.7368801006935308
         R-Squared: -54.62920653205764
         F1 score: 0.457007717201892
```

Figure 8.2.31 Model And Metrics

```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.45      0.62     19887
           1       0.02      0.95      0.03       198

    accuracy                           0.46     20085
   macro avg       0.51      0.70      0.33     20085
weighted avg       0.99      0.46      0.62     20085
```

Figure 8.2.32 Classification Report