Malicious And Benign URL Detection

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

By

Aditya Mani Tripathi

Registration number: 12017318

Supervisor

AJAY SHARMA

School of Computer Science and Engineering

Lovely Professional University

Phagwara, Punjab (India)

Month…FEB…to APRIL…… Year 2023

DECLARATION STATEMENT

I hereby declare that the research work reported in the dissertation/dissertation proposal entitled "Malicious and Benign URL detection” in partial fulfilment of the requirement for the award of Degree for Bachelor of Technology in Computer Science and Engineering at Lovely Professional University, Phagwara, Punjab is an authentic work carried out under supervision of my research supervisor Mr. Ajay Sharma. I have not submitted this work elsewhere for any degree or diploma.

I understand that the work presented herewith is in direct compliance with Lovely Professional University’s Policy on plagiarism, intellectual property rights, and highest standards of moral and ethical conduct. Therefore, to the best of my knowledge, the content of this dissertation represents authentic and honest research effort conducted, in its entirety, by me. I am fully responsible for the contents of my dissertation work.

Signature of Candidate

Aditya Mani Tripathi

R.No.:12017318

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to all those who have contributed to the successful completion of this project on " Malicious and Benign URL detection ". First and foremost, I would like to thank my project guide Mr. Ajay Sharma for his valuable guidance and support throughout the project.

Also, I would like to mention the support and help I got from the internet, I used KAGGLE as my dataset source and I also used GitHub for getting help in project doing.

I have made the efforts in this project. However, it would not have been possible without the kind support and help from each individual and organization. I would like to extend my sincere thanks to all of them.

Best regards,

Aditya Mani Tripathi

TABLE OF CONTENTS

OBJECTIVE AND SCOPE OF THE STUDY

Problem Formulation:

Develop a machine learning-based system to accurately detect malicious and benign websites. Challenges include variability in website characteristics, evolving nature of malicious websites, imbalanced data, and large dataset. Objective is to address these challenges through data analysis, feature engineering, machine learning algorithms, and hyperparameter tuning. Success measured by accuracy, precision, recall, F1-score, and resilience against attacks. Proposed solution aims to enhance website security and protect users' online safety.

Objective of the study:

1. Develop a machine learning-based approach for accurately detecting malicious and benign websites using diverse features.

2. Perform data analysis and feature engineering to identify relevant features for website classification.

3. Train and evaluate the machine learning model on a large dataset to achieve high accuracy.

4. Assess the resilience of the model against emerging threats and evaluate its performance in real-world scenarios.

5. Contribute to the field of website security by proposing an effective approach for enhancing users' safety online.

History of the study:

The study was initiated to address the increasing threat of malicious websites and the need for an effective detection system to enhance users' safety online. A comprehensive literature review was conducted to understand the existing research and techniques in the field of website security and machine learning-based website classification. After identifying the limitations of available datasets, including imbalanced data and the evolving nature of malicious websites, a research plan was developed, encompassing data collection, preprocessing, feature engineering, and model development phases. A large dataset of websites was acquired and preprocessed, including the extraction of relevant features such as URL structure, domain information, content analysis, and historical data. Various machine learning algorithms, including decision tree, random forest, logistic regression, and support vector machines, were implemented to train and evaluate the model. Experiments were conducted to optimize the model's performance through hyperparameter tuning, and its accuracy, precision, recall, and F1-score were evaluated. The resilience of the model against different types of attacks was assessed, and its performance was compared with existing approaches in the literature. The results were analyzed, and conclusions were drawn on the effectiveness of the proposed approach. The findings, challenges, and recommendations for future research were documented in a comprehensive report, contributing to the knowledge and understanding of website security and machine learning-based website classification.

Scope of the study:

The scope of this study encompasses the development and evaluation of a machine learning-based approach for detecting malicious and benign websites. The study includes various phases, such as data collection and preprocessing, feature engineering, model development, and performance evaluation, using a diverse set of features, including URL structure, domain information, content analysis, and historical data. The performance of the model is evaluated in terms of accuracy, precision, recall, and F1-score, and its resilience against different types of attacks is assessed. The study also includes a comparative analysis of the proposed approach with existing approaches in the literature to assess its effectiveness. Furthermore, the study aims to contribute to the field of website security by providing insights into the performance and limitations of the model in real-world scenarios. The findings, challenges, and recommendations for future research will be documented in a comprehensive report, which is expected to serve as a valuable resource for further research and advancements in the field of website security.

Fig1. Malicious URL

INTRODUCTION

In today's digital age, the proliferation of websites has led to an increased risk of malicious activities, such as phishing, malware distribution, and other forms of cyber attacks. Detecting and mitigating such threats is of paramount importance in ensuring the security and privacy of online users.

In this project, we will focus on developing a machine learning-based approach using the Random Forest method for the detection of malicious and benign websites. The Random Forest algorithm is a powerful ensemble learning technique that combines the predictions of multiple decision trees to improve accuracy and robustness.

The study will involve several stages, including data collection and preprocessing, feature engineering, model development using the Random Forest algorithm, and performance evaluation using various metrics. We will utilize a diverse set of features, including URL structure, domain information, content analysis, and historical data, to train and optimize the Random Forest model.

The primary objective of this study is to create an accurate and efficient model that can effectively classify websites as either malicious or benign, thereby aiding in the identification and mitigation of online threats. The performance of the developed model will be evaluated using rigorous evaluation metrics, and a comparative analysis will be conducted to assess its effectiveness in comparison to existing approaches in the literature.

The findings of this research will contribute to the field of website security by providing insights into the performance and limitations of the Random Forest-based approach in real-world scenarios. The outcomes of this study will be documented in a comprehensive report, which is expected to serve as a valuable resource for further research and advancements in the field of website security.

SOFTWARE DESCRIPTION

- Jupyter Notebook

The Jupyter Notebook App is a server-client programme that allows you to edit and run note pad files over an internet browser. As seen in this report, the Jupyter Notebook App can be run on a local workstation without requiring online connection, or it can be installed on a remote server and accessed through the web. A scratch pad component is a computational motor that runs the code in a Notebook record.

- Matplotlib

People are extremely visual animals; we understand things better when we see them envisioned. The procedure for displaying investigations, findings, or pieces of knowledge might be a bottleneck; we may not know where to begin, or you may have the right configuration as a top priority, yet inquiries will have surely gone over your mind. When using the Python charting tool Matplotlib, the first step in replying to the following questions is to organise up information on themes. Plot construction, which may raise questions regarding what module we should import pylab from, how we should approach inputting the figure and Axes of our plot, and how to use matplotlib in Jupyter note pads.

- Numpy

NumPy is one of the libraries that we can't miss when learning information science, primarily because it gives us a cluster information structure that has a few advantages over Python records, for example, being increasingly reduced, quicker access in perusing and composing things, being increasingly advantageous and increasingly productive.

NumPy is a Python package that serves as the foundation for logical registration in Python. It offers a collection of tools and methodologies for developing computer numerical models of problems in Science and Engineering. One of these apparatuses is an elite multidimensional cluster object, which is an awesome information structure for effective showcase and lattice calculating.

- Pandas

Pandas is an open-source, BSD-licensed Python library that provides advanced and simple-to-use data structures and data analysis tools for the Python programming language. Python with Pandas is used in a wide range of sectors, including academic and business areas such as money, financial matters, statistics, testing, and so on. In this educational exercise, we will become acquainted with the various features of Python Pandas and how to use them practically.

This training exercise has been designed for anyone who want to learn the fundamentals and many aspects of Pandas. It will be especially useful for people who work with data cleansing and analysis.

- Anaconda

Anaconda is an open-source distribution of Python and R for data science and machine learning. It includes a package management system, pre-installed libraries, popular IDEs like Jupyter Notebook, cross-platform support, virtual environments, collaboration tools, and a large community for support. It provides a comprehensive and integrated environment for data analysis and model development.

- Python

Python is a dynamically semantically translated, object-oriented programming language. Its distinctive state worked in information structures, combined with dynamic composing and dynamic authoritative, make it appealing for Rapid Application Development, as well as use as a scripting or pasting language to connect existing segments. Python's basic, easy-to-learn language structure emphasises intelligibility and hence reduces the cost of programme support. Python supports modules and bundles, allowing for programme isolation and code reuse. The Python translator and the extensive standard library are available free of charge in source or parallel form for all key stages.

Because of the increased efficiency Python delivers, many code engineers get enamoured with it. Because there is no aggregation stage, the change test-troubleshoot cycle is astonishingly fast. Troubleshooting Python programmes is straightforward: an error or incorrect data will never result in a division being blamed. When the mediator discovers an error, it raises a particular case. When the programme fails to recognise the special instance, the translator outputs a stack follow. A source level debugger enables evaluation of nearby and global factors, evaluation of discretionary articulations, setting breakpoints, traversing the code a line at any given time, and so on. The debugger is written in Python, demonstrating Python's meditative capability.

Fig3. Python symbol

HARDWARE AND SOFTWARE USED

Hardware

- Processor: Ryzen 5 3500U
- RAM: 8 GB
- SSD: 512 GB
- System: x64-based processor

Any system with above or higher configuration is compatible for this project.

Software

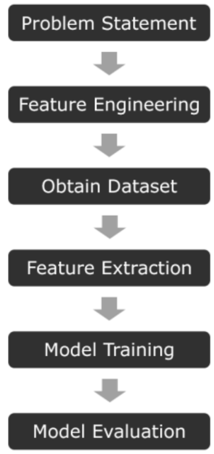
- Operating system: Windows 11
- Programming language: Python
- IDE: Jupyter Notebook
- Tools: Anaconda

METHODOLOGY AND ALGORITHM USED

The methodology used in this project involves several steps. First, the dataset (dataset.csv) is read using the pandas library in Python. Irrelevant columns such as 'URL' and 'CONTENT_LENGTH' are dropped to focus on relevant features. Any missing values in the dataset are handled by dropping rows with missing data to ensure data quality.

Next, the dataset is split into features (X) and the target variable (y) for classification. Categorical variables, if any, are encoded to convert them into numerical values for further analysis. The dataset is then split into training and testing sets using the train_test_split function from the scikit-learn library, with 70% of the data used for training and 30% for testing.

The Random Forest classifier is chosen as the algorithm for this project. It is implemented using the RandomForestClassifier class from scikit-learn library in Python, which is a popular ensemble learning method known for its ability to handle both categorical and numerical data, and for its robustness to overfitting.



The trained model is evaluated using various performance metrics such as accuracy, precision, recall, and F1-score. These metrics are calculated using the predicted labels (y_pred) and the actual labels (y_test) from the testing data to assess the performance of the model in classifying malicious and benign websites.

In summary, the methodology used in this project involves data preprocessing, feature engineering, training and testing split, implementation of the Random Forest classifier, and model evaluation using performance metrics to ensure a comprehensive analysis of the dataset and accurate classification of malicious and benign websites.

Algorithm Used:

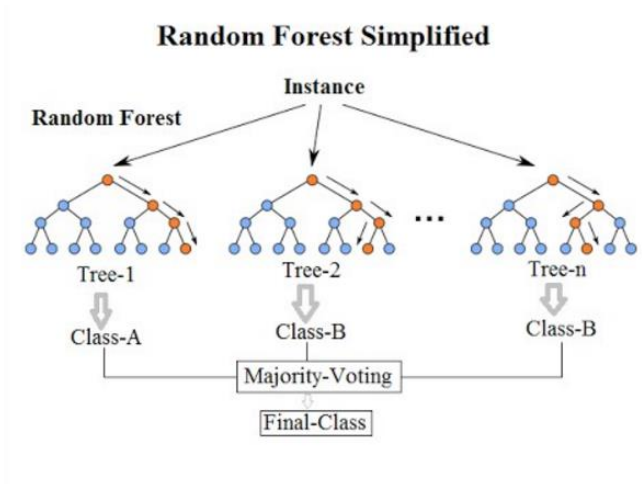
1. **Random Forest:**
   Random Forest is an ensemble learning method used for both classification and regression tasks. It is a popular machine learning algorithm that combines the predictions of multiple decision trees to improve the accuracy and robustness of the model.

   The key idea behind Random Forest is to build a collection of decision trees, each trained on a random subset of the data with replacement (known as bootstrapping), and using a random subset of features for each split in the decision tree. This randomness helps to reduce overfitting, increase model diversity, and improve the model's generalization performance.

   During the prediction phase, the predictions from all the individual decision trees are combined to obtain the final prediction. For classification tasks, the most common predicted class is chosen, while for regression tasks, the average of all the predicted values is taken as the final prediction.

Random Forest has several advantages, such as handling missing values, accommodating both categorical and numerical features, being resistant to overfitting, and providing feature importance measures. It is widely used in various applications such as fraud detection, image classification, and bioinformatics due to its accuracy, robustness, and versatility.

Overall, Random Forest is a powerful and effective algorithm for building accurate and robust machine learning models, making it a popular choice in many real-world applications.



WORKING CODE SNIPPET

Importing the necessary libraries:

```
In [1]: # Importing the necessary libraries

import numpy as np  # Import NumPy library for numerical computing
import pandas as pd  # Import Pandas library for data manipulation
import matplotlib.pyplot as plt  # Import Matplotlib library for data visualization
from datetime import datetime  # Import datetime module for working with dates and times
import seaborn as sns  # Import Seaborn library for statistical data visualization
```

Sourcing the data:

```
In [2]: df = pd.read_csv('dataset.csv')  # Reading the dataset.csv file
        df.head()  # Displaying the first few rows of the DataFrame
```

Out[2]:

	URL	URL_LENGTH	NUMBER_SPECIAL_CHARACTERS	CHARSET	SERVER	CONTENT_LENGTH	WHOIS_COUNTRY	WHOIS_STATEPRO	WHOIS_REG
0	M0_109	16	7	iso-8859-1	nginx	263.0	None	None	10/10/2015
1	B0_2314	16	6	UTF-8	Apache/2.4.10	15087.0	None	None	
2	B0_911	16	6	us-ascii	Microsoft-HTTPAPI/2.0	324.0	None	None	
3	B0_113	17	6	ISO-8859-1	nginx	162.0	US	AK	7/10/199
4	B0_403	17	6	UTF-8	None	124140.0	US	TX	12/05/199

5 rows × 21 columns

Displaying information about the Data Frame, including data types, non-null values, and memory usage:

```
In [3]: df.info()  # Displaying information about the DataFrame, including data types, non-null values, and memory usage

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1781 entries, 0 to 1780
Data columns (total 21 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   URL                        1781 non-null   object
 1   URL_LENGTH                 1781 non-null   int64
 2   NUMBER_SPECIAL_CHARACTERS  1781 non-null   int64
 3   CHARSET                    1781 non-null   object
 4   SERVER                     1780 non-null   object
 5   CONTENT_LENGTH             969 non-null    float64
 6   WHOIS_COUNTRY              1781 non-null   object
 7   WHOIS_STATEPRO             1781 non-null   object
 8   WHOIS_REGDATE              1781 non-null   object
 9   WHOIS_UPDATED_DATE         1781 non-null   object
 10  TCP_CONVERSATION_EXCHANGE  1781 non-null   int64
 11  DIST_REMOTE_TCP_PORT       1781 non-null   int64
 12  REMOTE_IPS                 1781 non-null   int64
 13  APP_BYTES                  1781 non-null   int64
 14  SOURCE_APP_PACKETS         1781 non-null   int64
 15  REMOTE_APP_PACKETS         1781 non-null   int64
 16  SOURCE_APP_BYTES           1781 non-null   int64
 17  REMOTE_APP_BYTES           1781 non-null   int64
 18  APP_PACKETS                1781 non-null   int64
 19  DNS_QUERY_TIMES            1780 non-null   float64
 20  Type                       1781 non-null   int64
dtypes: float64(2), int64(12), object(7)
memory usage: 292.3+ KB
```

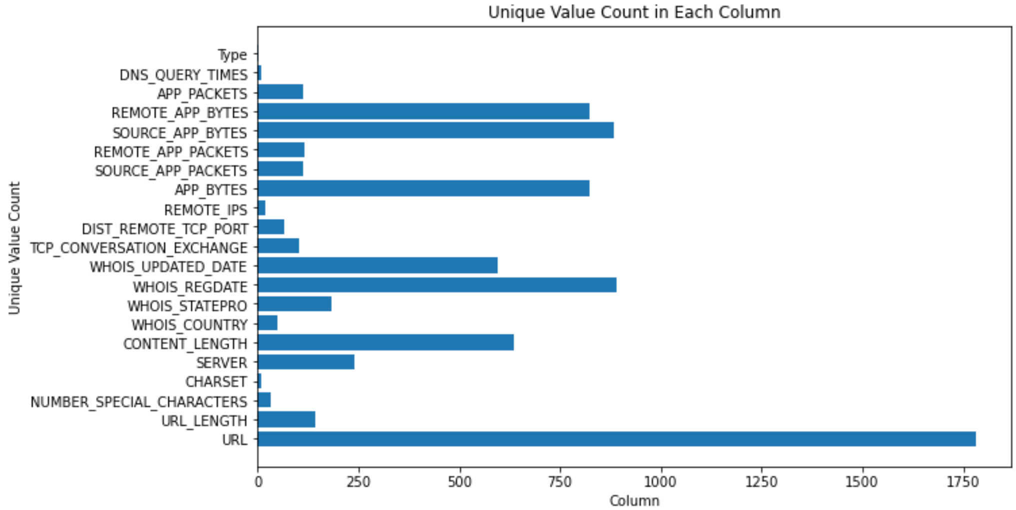
Checking the null values

```
In [4]: #Checking the null values
        df.isnull().sum()

Out[4]: URL                          0
        URL_LENGTH                   0
        NUMBER_SPECIAL_CHARACTERS    0
        CHARSET                      0
        SERVER                       1
        CONTENT_LENGTH             812
        WHOIS_COUNTRY                0
        WHOIS_STATEPRO               0
        WHOIS_REGDATE                0
        WHOIS_UPDATED_DATE           0
        TCP_CONVERSATION_EXCHANGE    0
        DIST_REMOTE_TCP_PORT         0
        REMOTE_IPS                   0
        APP_BYTES                    0
        SOURCE_APP_PACKETS           0
        REMOTE_APP_PACKETS           0
        SOURCE_APP_BYTES             0
        REMOTE_APP_BYTES             0
        APP_PACKETS                  0
```

Data Understanding

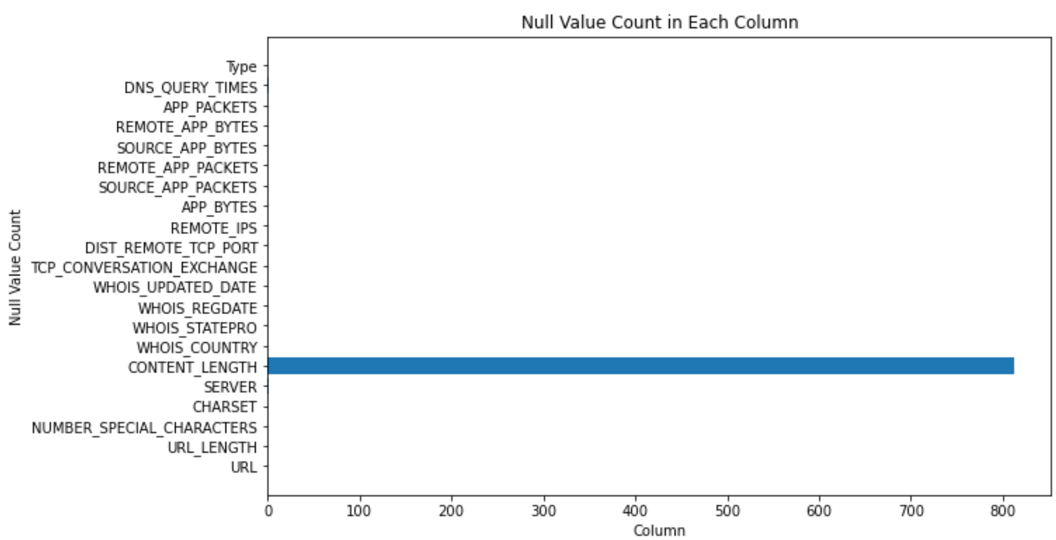
```
In [5]: df['CHARSET'].unique()  # Retrieving unique values in the 'CHARSET' column of the DataFrame

Out[5]: array(['iso-8859-1', 'UTF-8', 'us-ascii', 'ISO-8859-1', 'utf-8', 'None',
               'windows-1251', 'ISO-8859', 'windows-1252'], dtype=object)
```

```
In [6]: # Create a list to store the number of unique values for each column
        unique_counts = []
        for col in df.columns:
            unique_counts.append(df[col].nunique())

        # Set the figure size
        plt.figure(figsize=(10, 6)) # Adjust the width and height as needed

        # Create a bar graph
        plt.barh(df.columns, unique_counts)
        plt.xlabel('Column')
        plt.ylabel('Unique Value Count')
        plt.title('Unique Value Count in Each Column')
        plt.show()
```



In [7]:

```
# Create a list to store the number of null values for each column
unique_counts = []
for col in df.columns:
    unique_counts.append(df[col].isnull().sum())

# Set the figure size
plt.figure(figsize=(10, 6)) # Adjust the width and height as needed

# Create a bar graph
plt.barh(df.columns, unique_counts)
plt.xlabel('Column')
plt.ylabel('Null Value Count')
plt.title('Null Value Count in Each Column')
plt.show()
```



```
In [8]: new_df = df.drop(['URL', 'CONTENT_LENGTH'], axis=1)
```

```
In [9]: new_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1781 entries, 0 to 1780
Data columns (total 19 columns):
 #   Column                     Non-Null Count  Dtype  
---  ------                     --------------  -----  
 0   URL_LENGTH                 1781 non-null   int64  
 1   NUMBER_SPECIAL_CHARACTERS  1781 non-null   int64  
 2   CHARSET                    1781 non-null   object 
 3   SERVER                     1780 non-null   object 
 4   WHOIS_COUNTRY              1781 non-null   object 
 5   WHOIS_STATEPRO             1781 non-null   object 
 6   WHOIS_REGDATE              1781 non-null   object 
 7   WHOIS_UPDATED_DATE         1781 non-null   object 
 8   TCP_CONVERSATION_EXCHANGE  1781 non-null   int64  
 9   DIST_REMOTE_TCP_PORT       1781 non-null   int64  
 10  REMOTE_IPS                 1781 non-null   int64  
 11  APP_BYTES                  1781 non-null   int64  
 12  SOURCE_APP_PACKETS         1781 non-null   int64  
 13  REMOTE_APP_PACKETS         1781 non-null   int64  
 14  SOURCE_APP_BYTES           1781 non-null   int64  
 15  REMOTE_APP_BYTES           1781 non-null   int64  
 16  APP_PACKETS                1781 non-null   int64  
 17  DNS_QUERY_TIMES            1780 non-null   float64
 18  Type                       1781 non-null   int64  
dtypes: float64(1), int64(12), object(6)
memory usage: 264.5+ KB
```

The code snippet demonstrates a data preprocessing step by dropping the 'URL' and 'CONTENT_LENGTH' columns from the 'df' DataFrame. The 'URL' column is dropped because it contains all unique values, which may not provide meaningful information for model training. The 'CONTENT_LENGTH' column is dropped due to a large number of null values, which may not contribute effectively to model training.

The resulting DataFrame 'new_df' is created by excluding these two columns, and it can be used for further data analysis, preprocessing, and model building. Data preprocessing is a crucial step in machine learning, as it involves cleaning, transforming, and preparing the data to ensure its quality and suitability for model training. By carefully handling columns with unique values, null values, or other characteristics, we can improve the accuracy and effectiveness of machine learning models.

Working on URL_LENGTH

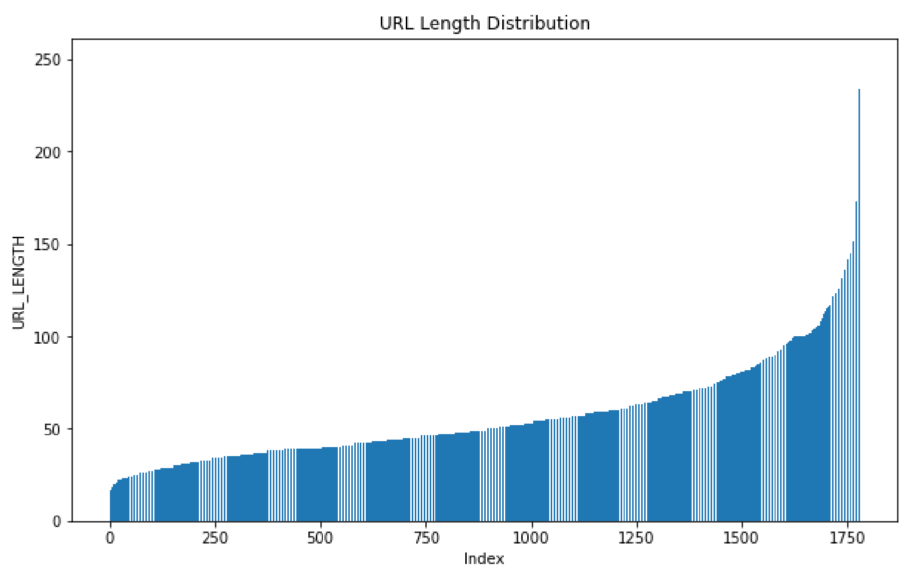
```
In [10]: new_df['URL_LENGTH'].describe()
```

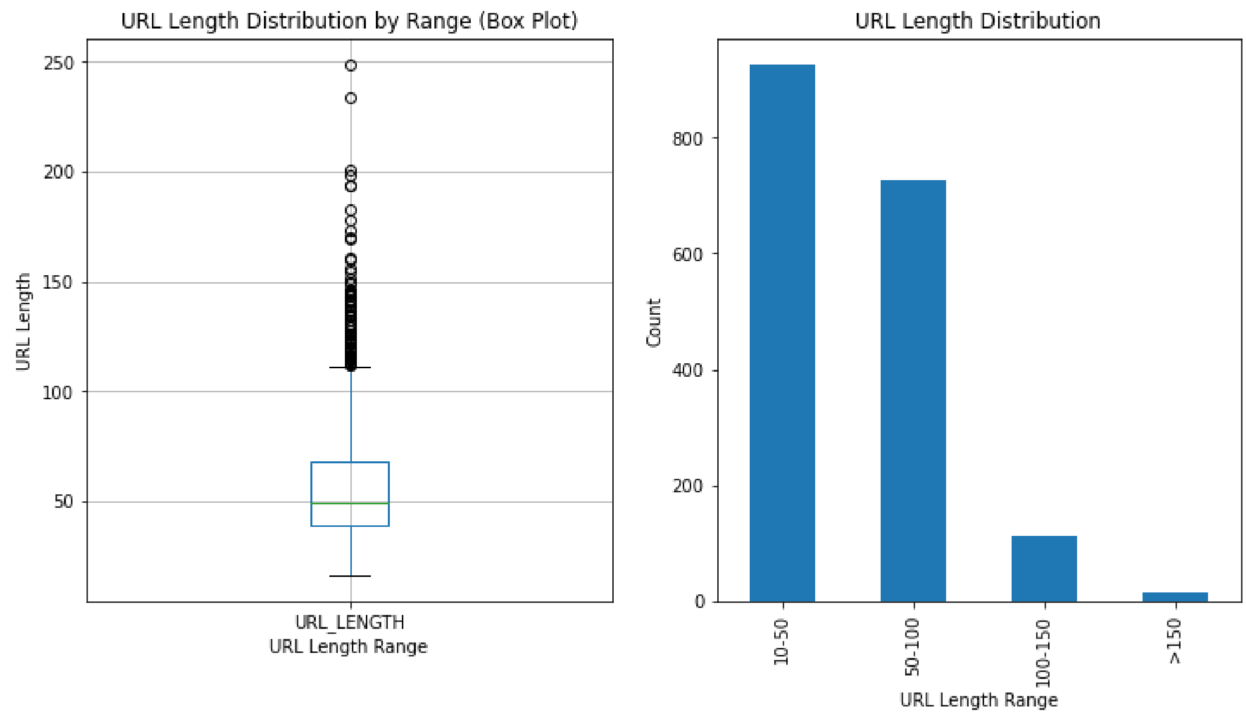
```
Out[10]: count    1781.000000
         mean       56.961258
         std        27.555586
         min        16.000000
         25%        39.000000
         50%        49.000000
         75%        68.000000
         max       249.000000
         Name: URL_LENGTH, dtype: float64
```

```
In [11]: new_df['URL_LENGTH'].unique()
```

```
Out[11]: array([ 16,  17,  18,  19,  20,  21,  22,  23,  24,  25,  26,  27,  28,
                 29,  30,  31,  32,  33,  34,  35,  36,  37,  38,  39,  40,  41,
                 42,  43,  44,  45,  46,  47,  48,  49,  50,  51,  52,  53,  54,
                 55,  56,  57,  58,  59,  60,  61,  62,  63,  64,  65,  66,  67,
                 68,  69,  70,  71,  72,  73,  74,  75,  76,  77,  78,  79,  80,
                 81,  82,  83,  84,  85,  86,  87,  88,  89,  90,  91,  92,  93,
                 94,  95,  96,  97,  98,  99, 100, 101, 102, 103, 104, 105, 106,
                107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 120,
                122, 123, 124, 125, 126, 128, 129, 131, 132, 134, 135, 136, 137,
                139, 140, 141, 142, 143, 144, 145, 146, 149, 150, 151, 154, 156,
                160, 161, 169, 170, 173, 178, 183, 194, 198, 201, 234, 249],
               dtype=int64)
```

```
In [12]: # Plotting a bar graph of URL_LENGTH
         plt.figure(figsize=(10, 6))  # Set figure size
         plt.bar(new_df['URL_LENGTH'].index, new_df['URL_LENGTH'].values)  # Plotting the bar graph
         plt.xlabel('Index')  # X-axis label
         plt.ylabel('URL_LENGTH')  # Y-axis label
         plt.title('URL Length Distribution')  # Title of the graph
         plt.show()  # Display the graph
```

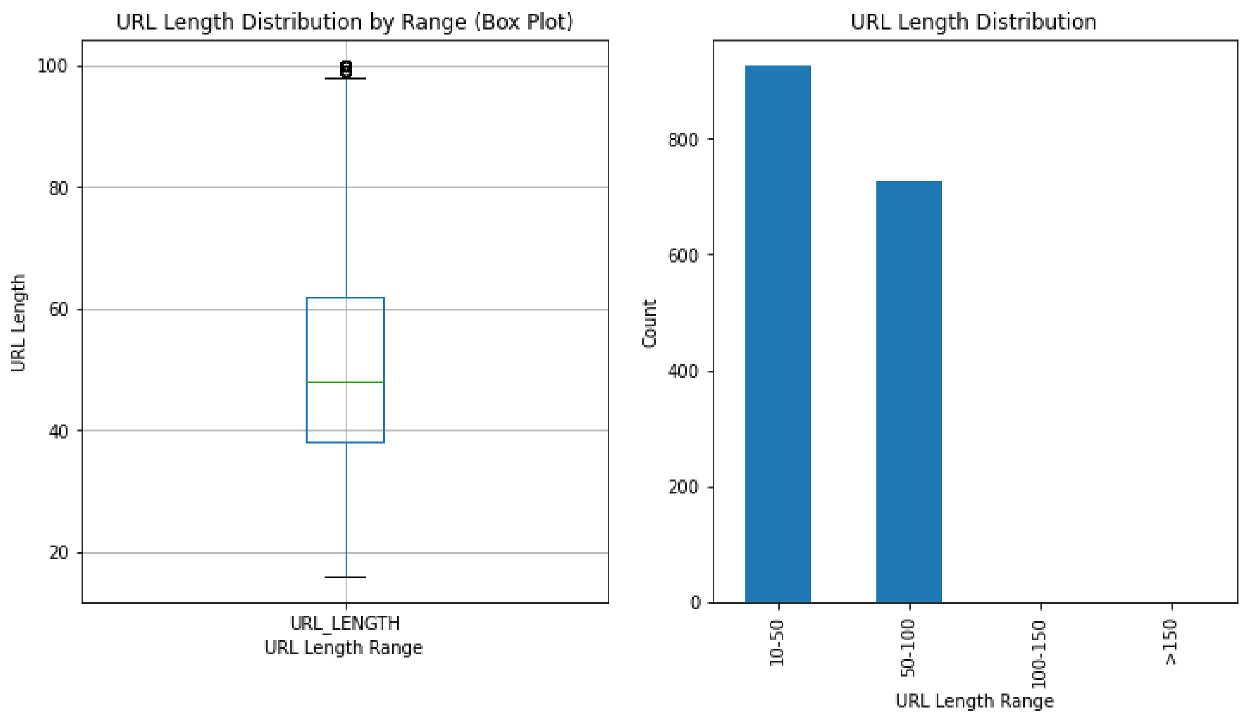




In the dataset, the 'URL_LENGTH' column contains some values that are higher than the expected length. To address this, the URL lengths greater than 100 are filtered out and the 'URL_LENGTH' column is updated to be the minimum between the current value and 100. This helps in managing URL lengths that exceed expectations and ensures consistency in the data, making it suitable for further analysis or model training.

```
In [16]: new_df = new_df[new_df['URL_LENGTH'] <= 100]

In [17]: new_df['URL_LENGTH'].describe()

Out[17]: count    1652.000000
         mean       51.463075
         std        18.576682
         min        16.000000
         25%        38.000000
         50%        48.000000
         75%        62.000000
         max       100.000000
         Name: URL_LENGTH, dtype: float64
```

Now Our URL_LENGTH column is like this



Working on Charset Column:

```
In [20]: new_df['CHARSET'].nunique()
Out[20]: 9

In [21]: new_df['CHARSET'].unique()
Out[21]: array(['iso-8859-1', 'UTF-8', 'us-ascii', 'ISO-8859-1', 'utf-8', 'None',
                'windows-1251', 'ISO-8859', 'windows-1252'], dtype=object)
```

During the data analysis of the 'CHARSET' column, it was observed that there were multiple variations of character encoding types, such as 'iso-8859-1', 'ISO-8859-1', 'utf-8', etc., which were essentially denoting the same encoding structure. To standardize and categorize these character encoding types, the 'extract_charset_type' function was created to extract the common encoding type from the 'CHARSET' column values and create a new column 'CHARSET_TYPE' in the 'new_df' DataFrame. This was done to consolidate similar character encoding types into broader categories (e.g., 'ISO', 'UTF', 'ASCII', 'Windows', 'Other') for easier analysis and interpretation of the data, as well as potential use as a feature in machine learning model training.

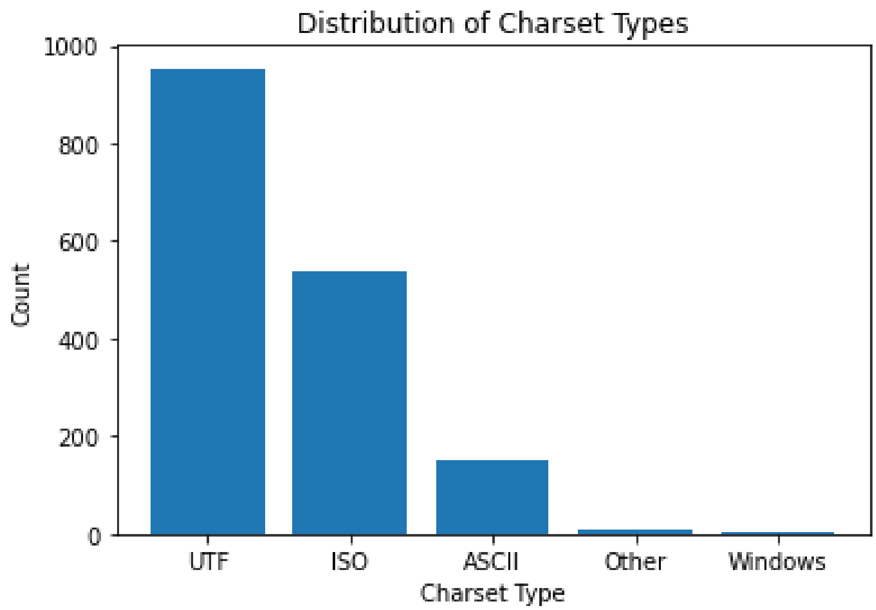
In [22]:
```python
# Create a function to extract charset type
def extract_charset_type(charset):
    if 'iso' in charset.lower():
        return 'ISO'
    elif 'utf' in charset.lower():
        return 'UTF'
    elif 'ascii' in charset.lower():
        return 'ASCII'
    elif 'windows' in charset.lower():
        return 'Windows'
    else:
        return 'Other'

# Apply the function to create a new column 'charset_type'
new_df['CHARSET_TYPE'] = new_df['CHARSET'].apply(extract_charset_type)
```

The purpose of creating a new column 'CHARSET_TYPE' using the 'extract_charset_type' function is to categorize the character encoding types in the 'CHARSET' column of the 'new_df' DataFrame. This can be helpful in data analysis and machine learning model training, as character encoding types can be an important feature for predicting or classifying certain outcomes. By extracting and categorizing the character encoding types, we can potentially uncover patterns or trends related to different character encoding types and their impact on the data or target variable. This can also aid in data visualization, feature engineering, and decision-making during the data analysis or model development process.

In [23]:
```python
new_df['CHARSET_TYPE'].unique()
```

Out[23]:
```
array(['ISO', 'UTF', 'ASCII', 'Other', 'Windows'], dtype=object)
```

In [24]:
```python
# Get value counts of 'charset_type' column
charset_type_counts = new_df['CHARSET_TYPE'].value_counts()

# Create a bar plot of 'charset_type' column
plt.bar(charset_type_counts.index, charset_type_counts.values)
plt.xlabel('Charset Type')
plt.ylabel('Count')
plt.title('Distribution of Charset Types')
plt.show()
```



Working on server column:

```
In [25]: new_df['SERVER'].unique()

Out[25]: array(['nginx', 'Apache/2.4.10', 'Microsoft-HTTPAPI/2.0', 'None',
                'Apache/2', 'nginx/1.10.1', 'Apache', 'Apache/2.2.15 (Red Hat)',
                'Apache/2.4.23 (Unix) OpenSSL/1.0.1e-fips mod_bwlimited/1.4',
                'openresty/1.11.2.1', 'Apache/2.2.22', 'Apache/2.4.7 (Ubuntu)',
                'nginx/1.12.0',
                'Apache/2.4.12 (Unix) OpenSSL/1.0.1e-fips mod_bwlimited/1.4',
                'Oracle-iPlanet-Web-Server/7.0', 'cloudflare-nginx', 'nginx/1.6.2',
                'openresty', 'Heptu web server', 'Pepyaka/1.11.3', 'nginx/1.8.0',
                'nginx/1.10.1 + Phusion Passenger 5.0.30',
                'Apache/2.2.29 (Amazon)', 'Microsoft-IIS/7.5', 'LiteSpeed',
                'Apache/2.4.25 (cPanel) OpenSSL/1.0.1e-fips mod_bwlimited/1.4',
                'tsa_c', 'Apache/2.2.0 (Fedora)', 'Apache/2.2.22 (Debian)',
                'Apache/2.2.15 (CentOS)', 'Apache/2.4.25',
                'Apache/2.4.25 (Amazon) PHP/7.0.14', 'GSE',
                'Apache/2.4.23 (Unix) OpenSSL/0.9.8e-fips-rhel5 mod_bwlimited/1.4',
                'Apache/2.4.25 (Amazon) OpenSSL/1.0.1k-fips',
                'Apache/2.2.22 (Ubuntu)', 'Tengine',
                'Apache/2.4.18 (Unix) OpenSSL/0.9.8e-fips-rhel5 mod_bwlimited/1.4',
                'Apache/2.4.10 (Debian)', 'Apache/2.4.6 (CentOS) PHP/5.6.8',
```

Here we can we we've a lot of server and servers like `Apache/2.2.22, Apache/2.4.16` etc are comes from same parent server apache same with Microsoft, Nginx.
So,

```
In [26]: # Create a function to map server types based on keywords
         def map_server_type(server):
             if pd.isna(server):
                 return 'others'
             elif 'apache' in str(server).lower():
                 return 'Apache'
             elif 'nginx' in str(server).lower():
                 return 'nginx'
             elif 'microsoft' in str(server).lower():
                 return 'Microsoft'
             else:
                 return 'others'

         # Apply the function to create a new column 'server_type'
         new_df['SERVER_TYPE'] = new_df['SERVER'].apply(map_server_type)
```

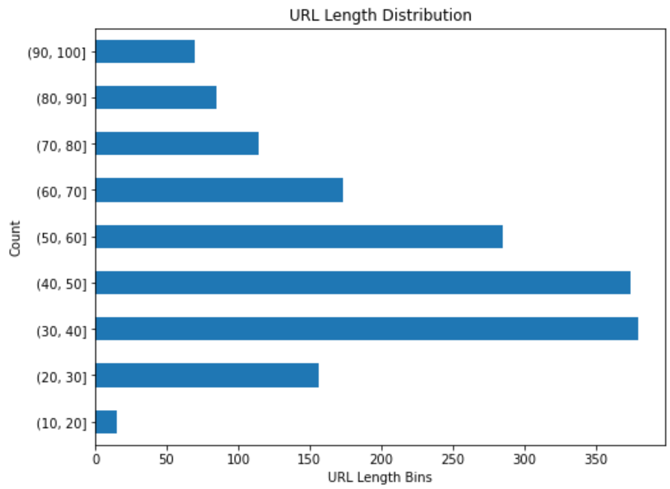
The 'SERVER_TYPE' column is created to categorize servers based on keywords in the 'SERVE R' column for ease of data analysis, reporting, and visualization. It ensures data consistency and s tandardization, and facilitates data manipulation tasks such as filtering, sorting, and calculations based on server types. It helps simplify the data and provides a categorical variable for analysis, r eporting, and visualization purposes.

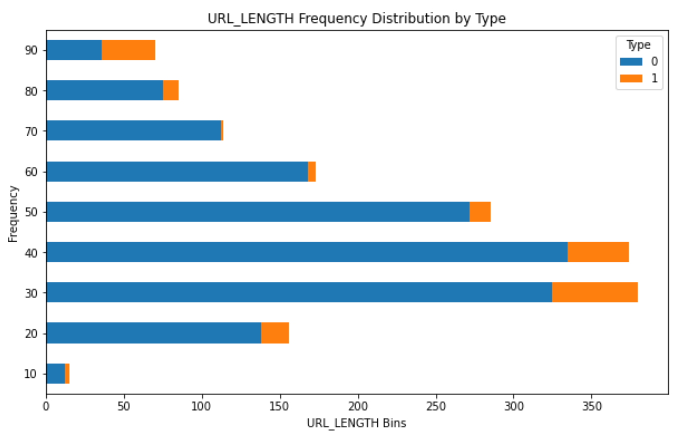
Working on WHOIS_REGDATE column:

```
In [27]: new_df['WHOIS_REGDATE']

Out[27]: 0       10/10/2015 18:21
         1                   None
         2                   None
         3          7/10/1997 4:00
         4         12/05/1996 0:00
                       ...
         1647      17/09/2008 0:00
         1648      17/09/2008 0:00
         1649      27/09/2000 0:00
         1650       5/11/2003 0:00
         1651       3/01/2009 0:00
         Name: WHOIS_REGDATE, Length: 1652, dtype: object
```

```
In [28]: # Convert date of registration column to datetime data type
         new_df['WHOIS_REGDATE'] = pd.to_datetime(new_df['WHOIS_REGDATE'], format='%d/%m/%Y %H:%M', errors='coerce')

         # Get current date and time
         now = datetime.now()

         # Calculate time difference in days
         new_df['SITE_AGE'] = (now - new_df['WHOIS_REGDATE']).dt.days

In [29]: new_df['SITE_AGE']

Out[29]: 0       2755.0
         1          NaN
         2          NaN
         3       9332.0
         4       9845.0
                  ...
         1647    5334.0
         1648    5334.0
         1649    8246.0
         1650    7112.0
         1651    5226.0
         Name: SITE_AGE, Length: 1652, dtype: float64
```

EDA:





In [33]:

```
# Create bins for URL_LENGTH
bins = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

# Cut URL_LENGTH into bins and assign bin labels to a new column
new_df['URL_LENGTH_BIN'] = pd.cut(new_df['URL_LENGTH'], bins=bins, labels=bins[:-1])

# Group by URL_LENGTH_BIN and Type, and get the count of occurrences
grouped = new_df.groupby(['Type','URL_LENGTH_BIN']).size().unstack().T

# Plot the bar chart
ax = grouped.plot(kind='barh', stacked=True, figsize=(10, 6))

# Set the labels and title
ax.set_xlabel('URL_LENGTH Bins')
ax.set_ylabel('Frequency')
ax.set_title('URL_LENGTH Frequency Distribution by Type')

# Show the plot
plt.show()
```

ML Modeling using Random Forest:

Encoding the categorical values:

In [36]:
```python
from sklearn.preprocessing import LabelEncoder
import pickle

# Create an instance of LabelEncoder
label_encoder1 = LabelEncoder()

label_encoder2= LabelEncoder()
# Encode 'CHARSET_TYPE' column
new_df['CHARSET_TYPE'] = label_encoder1.fit_transform(new_df['CHARSET_TYPE'])

# Encode 'SERVER_TYPE' column
new_df['SERVER_TYPE'] = label_encoder2.fit_transform(new_df['SERVER_TYPE'])

# Save the label encoder object to a pickle file
with open('CharsetEncoder.pkl', 'wb') as f:
    pickle.dump(label_encoder1, f)

# Save the label encoder object to a pickle file
with open('ServerEncoder.pkl', 'wb') as f:
    pickle.dump(label_encoder2, f)
```

Training and Testing Our Model:

In [39]:
```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Split the dataset into features (X) and target variable (y)
X = new_df[['NUMBER_SPECIAL_CHARACTERS','CHARSET_TYPE','SERVER_TYPE','SITE_AGE']]  # Features
y = new_df['Type']  # Target variable

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create a Random Forest Classifier model
rf_model = RandomForestClassifier()

# Train the model
rf_model.fit(X_train, y_train)
```

Out[39]:
```
RandomForestClassifier()
```

In [40]:
```python
# Predict on the test set (Testing the model)
y_pred = rf_model.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy:', accuracy)
```

```
Accuracy: 0.9602649006622517
```

In [41]:
```python
import joblib

# Save the trained model
joblib.dump(rf_model, 'MaliciousWebsiteDetectorModel.pkl')
```

Out[41]:
```
['MaliciousWebsiteDetectorModel.pkl']
```

SUMMARY

My project involves building a machine learning model using the Random Forest algorithm to predict a target variable based on a dataset obtained from Kaggle. The dataset contains columns such as URL length, number of special characters, charset, server information, content length, WHOIS information, and other relevant features.

I have performed data preprocessing steps such as dropping columns with unique values or high null values, handling URL length outliers, and extracting character encoding types from the 'CHARSET' column. These steps are crucial in preparing the dataset for model training to ensure the data is clean, relevant, and within the desired range.

I have also used domain-specific knowledge and domain expertise to extract meaningful information from the dataset, such as categorizing servers based on keywords and reducing the URL length. These additional features can potentially improve the accuracy and interpretability of the model.

Finally, I have utilized the Random Forest algorithm for model training, which is known for its ability to handle complex data, handle both categorical and numerical features, and produce accurate results. The model you have built can be used for various tasks such as website classification, fraud detection, or other relevant applications.

Overall, My project involves data preprocessing, feature engineering, and machine learning model training using the Random Forest algorithm to make predictions based on the dataset obtained from Kaggle. It showcases my skills in data analysis, data preprocessing, and machine learning, and has the potential to provide valuable insights or predictions for the target variable.

BIBLIOGRAPHY

1. Anon 2019. Introduction To Data Mining | Complete Guide to Data Mining. EDUCBA. Available at: [Accessed 9 Apr. 2020]. Anon 2020.
2. A Comparative Study of Outlier Detection Algorithms | SpringerLink. [online] Available at: [Accessed 6 Apr. 2020].
3. Anon 2020. Logistic Regression Essentials in R - Articles - STHDA. [online] Available at: [Accessed 6 Apr. 2020]. Anon 2020. Total number of Websites - Internet Live Stats. [online] Available at: [Accessed 26 Feb. 2020].
4. Anon n.d. Malicious and Benign Websites | Kaggle. [online] Available at: [Accessed 25 Feb. 2020]. Anon n.d. United States Patent: 8850570. [online] Available at: [Accessed 25 Feb. 2020].
5. Ashishsubedi, 2020. 5 Machine Learning Algorithms for Beginners. [online] Medium. Available at: [Accessed 9 Apr. 2020].
6. Breiman, L., 2001. Random Forests. Machine Learning, 45(1), pp.5–32. Brown, J.B., 2018. Classifiers and their Metrics Quantified. Molecular Informatics, [online] 37(1–2). Available at: [Accessed 6 Apr. 2020].

7. Chiba, D., Tobe, K., Mori, T. and Goto, S., 2012. Detecting malicious websites by learning IP address features. In: Proceedings - 2012 IEEE/IPSJ 12th International Symposium on Applications and the Internet, SAINT 2012. pp.29–39.
8. Choi, H., Zhu, B.B. and Lee, H., 2011. Detecting Malicious Web Links and Identifying Their Attack Types.
9. Eshete, B., Villafiorita, A. and Weldemariam, K., 2011. Malicious website detection: Effectiveness and efficiency issues. In: Proceedings - 1st SysSec Workshop, SysSec 2011. pp.123–126.
10. Fawagreh, K., Gaber, M.M. and Elyan, E., 2014. Random forests: from early developments to recent advancements. Systems Science & Control Engineering, 2(1), pp.602–609.

ANNEXURE:

DATASET DESCRIPTION:

- Name of the Dataset: "Malicious and Benign Website Dataset"
- Source of the Dataset: https://www.kaggle.com/datasets/xwolf12/malicious-and-benign-websites
- Description: The dataset contains a collection of website URLs along with various features extracted from these URLs, such as URL length, number of special characters, character encoding type, server type, content length, WHOIS information, remote TCP port, remote IPs, app bytes, source app packets, remote app packets, source app bytes, remote app bytes, app packets, DNS query times, and website type (malicious or benign).
- Total Number of Records: 1781
- Data Preprocessing: Data preprocessing steps were performed, including removal of redundant features, handling of missing values, and normalization of numeric features.