St. Francis
College for Women
Begumpet, Hyderabad-500016
(Autonomous & Affiliated to Osmania University)

Detecting Phishing Websites Using Deep Learning Technolgies

M.Sc (Data Science) Dissertation

Submitted to St. Francis College for Women in partial fulfillment of the requirements for the award of the Degree of

Masters of Science
By

Ailneni Banu Harshini
121323030026

Under the guidance of
Dr. Sr. Sujatha Yeruva

Department of Computer Science

St. Francis College for Women

Begumpet, Hyderabad – 500 016

(Autonomous and affiliated to Osmania University)

MARCH – 2025

St. Francis
College for Women
Begumpet, Hyderabad-500016
(Autonomous & Affiliated to Osmania University)

Detecting Phishing Websites Using Deep Learning Technolgies

MSc Dissertation

Submitted to St. Francis College for Women in partial fulfillment of the requirements for the award of the Degree of

Masters of Science
By

Ailneni Banu Harshini
121323030026

Under the guidance of
Dr. Sr. Sujatha Yeruva

Department of Computer Science

St. Francis College for Women

Begumpet, Hyderabad – 500 016

(Autonomous and affiliated to Osmania University)

MARCH – 2025

St. Francis

College for Women

Begumpet, Hyderabad-500016

(Autonomous & Affiliated to Osmania University)

CERTIFICATE

This is to certify that this bonafide project work titled "**Detecting Phishing Websites Using Deep Learning Technolgies**" has been carried out by **Ailneni Banu Harshini** bearing Roll No: **121323030026** towards partial fulfillment of the requirements for the award of Degree of Masters in Data Science from St. Francis College for Women,Begumpet for the academic year **2024-2025**.

[illegible]

Supervisor

[illegible]

Head of the Department

Computer Science Dept
ST FRANCIS COLLEGE FOR WOMEN
Begumpet, Hyderabad - 500 016

[illegible]

External Examiner

[illegible]

Controller of Examinations

Controller of Examinations
St. Francis College for Women
(Autonomous)
Begumpet, Hyderabad-16.

DECLARATION

The current study "**Detecting Phishing Websites Using Deep Learning Technolgies**" has been carried out under the supervision of **Dr. Sr. Sujatha Yeruva**, Assistant Professor, Department of Computer Science, St. Francis College for Women. We hereby declare that the present study that has been carried out by me, **Ailneni Banu Harshini**, during March 2024-2025 is original and no part of it has been carried out prior to this date.

Date: 10/03/2025

Signature of Candidate: Banu

ABSTRACT

The rapidly growing phishing sites are one of the most alarming cybersecurity threats where users are cheated to provide their login credentials, financial information, and personal data. Traditional rule-based and machine learning methods find it difficult to cope with the ever-changing phishing tactics of the cybercriminals. To this end, the present work aims to propose a deep learning-based phishing detection model using a Fully Connected Neural Network (FCNN). The model trains on a set of datasets having various URL-based features for categorizing the classification of websites into legitimate and phishing. The results of the experimental results indicate the proposed model classifies with accuracy of 85% in training set and with 84% in test sets, hence validating its practical suitability.

To achieve better accessibility, the model is integrated into an easy-to-use Graphical User Interface (GUI) using Tkinter, through which r URL analysis and phishing detection take place. It is an efficient and scalable method for detecting phishing attempts, and thus the chance of cyber fraud is reduced to a great extent. This paper shows the effective application of deep learning in the field of cybersecurity and opens ways for future developments in phishing prevention strategies.

ACKNOWLEDGEMENT

The successful completion of my project was made possible by the invaluable guidance and support of numerous individuals, for which I am truly grateful.

I extend my heartfelt thanks to **Dr. Sr. Sujatha Yeruva**, the Head of the Department, for granting us the opportunity to undertake this project and for her continuous support and encouragement.

Special appreciation goes to my project guide, **Dr. Sr. Sujatha Yeruva**, whose unwavering support and guidance were instrumental in navigating the complexities of the project. Her expertise and insights were invaluable in ensuring the project's success, and I am deeply grateful for her mentorship.

I also want to express my gratitude to our family, friends, and teachers, whose encouragement and motivation served as constant pillars of support throughout the project. Their belief in my abilities kept me motivated and focused, and for that, I am truly thankful.

Lastly, I extend my appreciation to all the computer lab assistants for their assistance and cooperation, allowing us access to the necessary equipment and resources essential for the project's execution.

Together, the collective efforts and support of these individuals played a significant role in the successful completion of my project, and I am sincerely grateful for their contributions.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

CHAPTER 1

INTRODUCTION

In the rapidly evolving digital era, cyber threats have grown exponentially, with phishing attacks emerging as one of the most prevalent and damaging forms of cybercrime. Phishing websites are malicious platforms designed to deceive users into divulging sensitive information, such as passwords, credit card details, and personal identification data. These attacks exploit human vulnerabilities and technical loopholes, causing significant financial and reputational losses globally.

Detecting Phishing Websites Using Deep Learning Technologies aims to address this pressing challenge by leveraging the power of artificial intelligence. Deep learning, a subset of machine learning, has shown remarkable success in identifying complex patterns within large datasets, making it an ideal candidate for tackling the intricacies of phishing detection. This project focuses on building a robust, automated system capable of distinguishing between legitimate and phishing websites with high accuracy.

The dataset used in this project encapsulates a variety of URL-based features, such as the presence of IP addresses, URL length, redirection behavior, and the use of HTTPS. These attributes are critical indicators of phishing activity, as attackers often employ specific tactics to disguise malicious URLs. For instance, excessively long URLs, tiny URLs, or the use of special characters like "@" are common in phishing attempts. By analyzing these features, the project seeks to uncover patterns that differentiate phishing websites from legitimate ones.

The proposed system employs advanced deep learning models, such as Fully connected Neural Network(FCNNs),Wide and Deep Model and convolutional neural networks (CNNs) to analyze the dataset. These models are well-suited for handling structured data and extracting meaningful insights. The ultimate goal is to develop a phishing detection mechanism that not only identifies threats but also adapts to emerging attack patterns through continuous learning.

This project holds immense significance in enhancing cybersecurity frameworks. By automating the detection process, it reduces reliance on manual interventions and minimizes the time required to identify and neutralize threats. Moreover, it empowers organizations and individuals to navigate the digital landscape safely, fostering trust in online interactions.

1.1 OBJECTIVE:

The primary objective of this project is to develop an **intelligent and automated system** for detecting phishing websites using deep learning technologies. Phishing attacks continue to be a significant cybersecurity concern, with attackers deploying deceptive websites to steal sensitive user information, including login credentials, financial details, and personal data. This project aims to provide a **highly accurate and efficient phishing detection mechanism** by leveraging advanced machine learning models.

To achieve this, the system will analyze **key URL-based features**, identifying patterns that distinguish legitimate websites from phishing sites. Traditional phishing detection approaches, such as blacklist-based filtering and heuristic rule-based methods, often struggle to keep pace with rapidly evolving cyber threats. This project seeks to **overcome these limitations** by utilizing deep learning techniques that can learn from vast amounts of data and generalize well to new, unseen phishing attacks.

The system will employ **advanced models**, including:

- **Fully Connected Neural Networks (FCNNs)** – To capture intricate relationships between URL-based features.
- **Wide & Deep Model** – To balance memorization of known phishing patterns with generalization to emerging threats.
- **Convolutional Neural Networks (CNNs)** – To detect complex structures and patterns in URL data efficiently.

By combining these deep learning architectures, the project aims to improve the detection **accuracy, speed, and adaptability** of phishing classification systems. The model will be trained on a dataset of URLs, learning to distinguish between malicious and legitimate sites based on key characteristics such as domain features, presence of suspicious keywords, and other structural patterns.

Additionally, the project seeks to **enhance cybersecurity frameworks** by providing a scalable solution that can be integrated into various applications, such as:

- **Web browsers** – To warn users before they access suspicious sites.
- **Antivirus software** – To strengthen existing security solutions.
- **Enterprise cybersecurity systems** – To protect organizations from phishing-based cyber threats.

1.2 PURPOSE:

Phishing attacks have become one of the most prevalent cybersecurity threats, targeting individuals and organizations worldwide. Cybercriminals create deceptive websites that mimic legitimate platforms to steal sensitive information, such as login credentials, financial details, and personal data. Traditional phishing detection methods, including blacklists and rule-based systems, often fail to keep up with the rapid evolution of phishing tactics, leaving users vulnerable to sophisticated attacks.

This project aims to address this critical issue by developing an **automated phishing detection system** that leverages deep learning technologies. By analyzing URL-based features, the system can accurately classify websites as phishing or legitimate in real time. Unlike conventional detection approaches, deep learning models can continuously learn and adapt to emerging threats, improving their detection accuracy over time.

The primary objective of this project is to enhance online security by offering a more reliable and efficient solution for phishing detection. By integrating advanced machine learning techniques, the system provides **proactive protection**, reducing the likelihood of users falling victim to fraudulent websites. It is designed to assist both individual users and organizations in safeguarding their data, thereby minimizing financial losses, identity theft, and reputational harm caused by phishing attacks.

Additionally, this project contributes to the broader field of cybersecurity by advancing research in phishing detection using deep learning. The insights gained from this study can serve as a foundation for future developments in AI-driven cybersecurity frameworks. Ultimately, the project aspires to create a safer digital ecosystem, empowering users with a robust defense mechanism against cyber threats.

1.3 Scope and Applicability:

This project is centered on detecting phishing websites by analyzing various URL-based features using deep learning techniques. It is designed to operate in real time, allowing for the immediate identification of malicious websites before users can interact with them. The system aims to detect phishing attempts across multiple online environments, making it highly relevant to sectors where cybersecurity is crucial, such as:

- **E-commerce platforms** – Preventing fraudulent websites from deceiving online shoppers.
- **Banking and financial services** – Protecting users from phishing scams that attempt to steal banking credentials.
- **Social media platforms** – Identifying phishing links that spread through messages, posts, or advertisements.
- **Enterprise networks** – Enhancing corporate security by preventing employees from accessing malicious websites.
- **Email security** – Assisting email filtering systems in detecting phishing links embedded in emails.

The **applicability** of this project extends beyond individual users to enterprises, cybersecurity firms, and regulatory bodies. The system can be integrated into various cybersecurity tools, such as:

- **Web browsers** – Providing real-time alerts or blocking access to suspected phishing sites.
- **Antivirus software** – Enhancing existing security solutions with AI-driven phishing detection.
- **Enterprise security solutions** – Strengthening organizational defenses against cyber threats.
- **Government and law enforcement agencies** – Assisting in cybercrime investigations by identifying phishing patterns.

Furthermore, the system's **adaptive learning capabilities** allow it to evolve alongside emerging phishing techniques. As cybercriminals develop new methods to bypass traditional security measures, the deep learning model continuously refines its detection accuracy, ensuring long-term effectiveness. This scalability makes it a **future-proof** solution that remains relevant in an ever-changing digital landscape.

By implementing this system, organizations can **reduce financial losses**, prevent **data breaches**, and **protect brand reputation** from the impact of phishing attacks. Individuals, in turn, gain a more secure browsing experience, minimizing their risk of identity theft and fraud.

Ultimately, this project serves as a **comprehensive cybersecurity measure**, reinforcing trust in digital interactions while advancing the capabilities of AI-driven phishing detection systems.

CHAPTER 2
SURVEY OF TECHNOLOGIES

CHAPTER 2

SURVEY OF TECHNOLOGIES

2.1 PROGRAMMING LANGUAGES:

• **Python:** Python is a versatile programming language known for its simplicity and readability, making it ideal for rapid development. It offers a vast ecosystem of libraries and frameworks for various tasks such as web development, data analysis, and machine learning. In this project, Python served as the primary programming language for backend development, data preprocessing, model building, and scripting. Its extensive libraries such as pandas, NumPy, sci-kit-learn, and TensorFlow/Keras provided powerful tools for data manipulation, machine learning, and deep learning tasks.

2.2 FRAMEWORKS AND LIBRARIES:

- **tkinter:**Tkinter is a standard Python library used for creating graphical user interfaces (GUIs). It provides a wide range of tools and widgets, such as buttons, labels, text boxes, and menus, to design interactive applications. Tkinter is simple to use, making it ideal for beginners, while also being powerful enough for more complex GUI projects. It is platform-independent, allowing applications to run on Windows, macOS, and Linux. As part of Python's standard library, Tkinter requires no additional installation, making it a convenient choice for GUI development.
- **Scikit-learn:** scikit-learn is a popular machine learning library for Python, offering a wide range of algorithms for classification, regression, clustering, and dimensionality reduction. It provides simple and efficient tools for data mining and analysis and is widely used for building predictive models. In this project, scikit-learn was utilized for implementing machine learning algorithms for sentiment analysis, such as support vector machines (SVM), random forests, and logistic regression.
- **TensorFlow or Keras:** TensorFlow and Keras are deep learning frameworks for Python, widely used for building and training neural network models. TensorFlow provides a flexible and scalable platform for constructing complex deep learning architectures, while Keras offers a high-level API for building neural networks with minimal code. In this project, TensorFlow or Keras was employed for implementing deep learning models, such as recurrent neural networks (RNNs) or convolutional neural networks (CNNs), for analyzing textual data and predicting sentiment.
- **NLTK (Natural Language Toolkit):** NLTK is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources, such as WordNet, along with a suite of text-processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning. In this project, NLTK was utilized for natural language processing tasks such as tokenization, lemmatization, and removing stop words to preprocess the textual data before feeding it into the sentiment analysis models.
- **NumPy and Pandas:** Utilized for data manipulation, analysis, and numerical computations.
- **Matplotlib or Seaborn:** Employed for data visualization and plotting.

2.3 DEVELOPMENT TOOLS:

Jupyter Notebook

Jupyter Notebook is an open-source web-based interactive development environment widely used for data science, machine learning, and research projects. It allows users to create and share documents that combine live code, equations, visualizations, and narrative text. Supporting multiple programming languages, including Python, R, and Julia, Jupyter is ideal for exploratory data analysis and prototyping. Its inline visualization support makes it a favorite among data scientists for presenting and analyzing data interactively. Jupyter Notebook is highly extensible and integrates well with popular libraries like NumPy, pandas, and Matplotlib.

PyCharm

PyCharm is a powerful integrated development environment (IDE) designed specifically for Python programming. Developed by JetBrains, it provides advanced features like intelligent code completion, debugging tools, and seamless integration with version control systems. PyCharm supports frameworks like Django, Flask, and data science tools, making it suitable for a wide range of projects. Its user-friendly interface and robust code analysis capabilities enhance productivity and ensure code quality. Available in both free (Community) and paid (Professional) editions, PyCharm is widely used by developers for efficient Python development.

2.4 WEB TECHNOLOGIES:

HTML/CSS: This is referred to as the core web development technology for providing interactive and aesthetically pleasing web pages. HTML, or Hypertext Markup Language, forms the basis of online content that supports providing semantics and structure in a simple manner to allow data to be adequately organized and presented on the screen. It uses several elements and tags to specify different elements on a web page, such as headers, paragraphs, pictures, links, and forms. Browsers make use of the Document Object Model (DOM), a hierarchical framework consisting of these components, to render web pages correctly.CSS, in contrast, develops HTML by governing the presentation of web pages, which enables the programmer to decide the styles that will be attached to HTML elements, including their fonts, colors, margins, and location. This enables developers to come up with attractive designs that work through various devices and screen sizes using CSS in the association of rules for style to HTML elements. HTML and CSS together give the necessary tools and methods to build and style web pages that benefit the production of interesting and intuitive online experiences.

CHAPTER 3
REQUIREMENTS AND ANALYSIS

CHAPTER 3
REQUIREMENTS AND ANALYSIS

3.1 PROBLEM DEFINITION

In the digital era, phishing attacks have emerged as a significant cybersecurity threat, deceiving users into disclosing sensitive information such as login credentials, banking details, and personal data. Traditional phishing detection methods, which rely on blacklists, heuristic rules, or manual analysis, have several limitations:

- **Evolving Threats:** Phishing techniques constantly evolve, making it difficult for rule-based systems to keep up with newly emerging attack patterns.
- **High False Positives:** Many existing detection methods incorrectly flag legitimate websites, causing unnecessary disruptions and reducing user trust.
- **Limited Generalization:** Traditional approaches may struggle to generalize across different types of phishing websites, leading to inconsistencies in detection accuracy.
- **Time-Consuming Detection:** Manual review and heuristic-based detection methods are slow, making real-time detection challenging.
- **Lack of Adaptability:** Many detection systems do not leverage learning-based models, making it difficult to adapt to new phishing strategies effectively.

To address these challenges, our project aims to develop a **deep learning-based phishing detection system** that can classify websites as phishing or legitimate based on extracted URL-based features. By leveraging **Fully Connected Neural Networks (FCNN), Wide & Deep models, and Convolutional Neural Networks (CNN)**, the system will enhance accuracy, adaptability, and threat detection capabilities. This solution will help organizations, cybersecurity firms, and individual users strengthen their defenses against phishing attacks by providing an automated and intelligent detection mechanism.

3.2 DESCRIPTION:

3.2.1Single Domain Prediction Interface

- **Input Box:** Users can manually enter a domain name to check if it is a phishing website.
- **Predict Button:** Clicking this button runs the deep learning model to classify the entered domain as either "Safe" or "Phishing."
- **Prediction Display:** Once the analysis is complete, the result is displayed on the interface. The output is color-coded:
  1. **Green** for "Safe" websites
  2. **Red** for "Phishing" websites

- **Reset Button:** Clears the input field and result display, allowing users to test another domain.

3.2.2 Dataset Upload & Processing Interface

- **Upload Button:** Users can upload a CSV file containing website data for analysis.
- **Automatic Preprocessing:** After uploading, the dataset undergoes preprocessing, including feature extraction, scaling, and splitting into training and testing sets.
- **Success Notification:** Users receive a confirmation message once the dataset is successfully uploaded and processed.

3.2.3 Data Visualization Interface

- **Dataset Preview:** Displays a preview of the uploaded dataset, showing the first and last few rows.
- **Graphical Analysis:**
    - A **pie chart** visualizes the proportion of phishing vs. safe websites.
    - A **bar chart** displays the distribution of key website features.
- **Visualization Display:** The generated graphs are displayed within the interface to help users analyze website characteristics.

3.2.4 How It Works Section

- Provides a step-by-step guide explaining how the system operates, including:
    1. **Dataset Processing:** Upload, clean, and preprocess data.
    2. **Model Training:** Uses a deep learning-based Fully Connected Neural Network (FCNN) to classify websites.
    3. **Prediction:** Classifies entered domains as either "Safe" or "Phishing."
    4. **Visualization:** Displays insights from the dataset using interactive graphs.

3.2.5 User Navigation & Interaction

- **Homepage Menu:** Users can navigate between different sections such as dataset upload, testing, visualization, and learning about the model.
- **Background Customization:** A background image is used to enhance the interface aesthetics, ensuring a user-friendly experience.
- **Error Handling:** If a domain is not found in the dataset or if the background image is missing, the interface displays an appropriate error message.

3.3 SYSTEM SPECIFICATIONS

3.3.1 HARDWARE SPECIFICATIONS:

• Computer or Laptop: Standard hardware with sufficient processing power to handle
data preprocessing, model building, and interface operations smoothly.
• Storage Space: Adequate storage capacity to store the sentiment analysis interface
application, datasets, and any additional files required for the project.
• Internet Connectivity: Stable internet connection for accessing external resources,
libraries, and updates during the development and deployment phases.

3.3.2 SOFTWARE SPECIFICATIONS:

• Operating System: Compatibility with major operating systems such as Windows,
macOS, or Linux distributions to ensure broad accessibility.
• Web Browser: Support for modern web browsers like Google Chrome, Mozilla Firefox, or Microsoft Edge for accessing the web-based interface.
• Python Environment: Installation of Python 3.x to execute the backend scripts, perform data preprocessing, and implement deep learning algorithms.
• Integrated Development Environment (IDE): Optional but recommended for code
development and debugging purposes. Popular IDEs include PyCharm, Jupyter Notebook, or Visual Studio Code.
• Dependencies and Libraries: Installation of required Python libraries such as NumPy,
pandas, scikit-learn, TensorFlow/Keras, Flask, NLTK, Matplotlib, tkinter and Seaborn for data manipulation, analysis, machine learning, natural language processing, and web development tasks.
• Additional Tools: Any specific tools or frameworks necessary for integrating the
sentiment analysis interface with e-commerce platforms or handling file uploads, if
applicable.

3. 4 MODEL ARCHITECTURE

This project utilizes three deep learning architectures for phishing website detection: **Fully Connected Neural Network (FCNN), Wide & Deep Model, and Convolutional Neural Network (CNN)**. Each model is designed to effectively analyze URL-based features and classify websites as phishing or legitimate with high accuracy.

1. Fully Connected Neural Network (FCNN)

The **Fully Connected Neural Network (FCNN)** serves as a foundational deep learning model that processes numerical website features extracted from URLs.

It learns complex relationships between input features through multiple densely connected layers.

Architecture Details:

Input Layer:

- Accepts feature-scaled numerical data representing website attributes.
- The input features are normalized to improve learning efficiency.

Hidden Layers:

Dense Layer 1:

- **128 neurons** with **ReLU activation** (Rectified Linear Unit).
- This layer captures high-level patterns in the input data.

Dropout Layer 1:

- **30% dropout rate** to prevent overfitting by randomly disabling 30% of neurons during training.

Dense Layer 2:

- **64 neurons** with **ReLU activation** to refine feature learning.

Dropout Layer 2:

- Another **30% dropout rate** for regularization.

Dense Layer 3:

- **32 neurons** with **ReLU activation**, further refining the feature representations.

Output Layer:

- **1 neuron with Sigmoid activation** to output a probability score between 0 and 1.
- A threshold (e.g., 0.5) is applied to classify the website as either phishing or legitimate.

2. Wide & Deep Model

The **Wide & Deep Model** combines **a linear model (wide component)** and **a deep neural network (deep component)** to leverage both memorization and generalization.

Architecture Details:

Wide Component (Linear Model):

- Directly processes raw input features.
- Applies a **linear transformation**, making it effective for memorizing simple relationships in the dataset.
- Helps detect phishing websites based on pre-existing patterns in historical data.

Deep Component (Neural Network):

Dense Layer 1:

- **128 neurons** with **ReLU activation** for initial feature extraction.

Dropout Layer 1:

- **30% dropout rate** to prevent overfitting.

Dense Layer 2:

- **64 neurons** with **ReLU activation** for deeper pattern recognition.

Dropout Layer 2:

- Another **30% dropout rate** for regularization.

Dense Layer 3:

- **1 neuron with Sigmoid activation**, producing a probability score for classification.

Merge Layer:

- Combines outputs from both the **wide** and **deep** components.
- The **final prediction** is made based on the combined representation of both models.

Final Output Layer:

- **1 neuron with Sigmoid activation** for binary classification (phishing or legitimate).

3. Convolutional Neural Network (CNN)

The **Convolutional Neural Network (CNN)** is used to extract spatial and sequential relationships between features, making it highly effective for detecting complex phishing patterns. Instead of treating URLs as simple text-based data, CNNs analyze their structural patterns and relationships.

Architecture Details:

Input Layer:

- Accepts website features in a reshaped format suitable for convolutional processing.
- Input features are structured to represent URL characteristics such as length, special characters, domain reputation, and more.

Convolutional Layers:

Conv1D Layer 1:

- **64 filters**, **kernel size = 3**, with **ReLU activation**.
- Detects local feature patterns in the input data.

Batch Normalization & Dropout Layer 1:

- **Batch normalization** normalizes activations for stable training.
- **30% dropout rate** to prevent overfitting.

Conv1D Layer 2:

- **32 filters**, **kernel size = 3**, with **ReLU activation**.
- Further extracts intricate phishing-related patterns.

Batch Normalization & Dropout Layer 2:

- Another **batch normalization** and **dropout (30%)** to improve generalization.

Flatten Layer:

- Converts the extracted feature maps into a **1D vector** for dense layers.

Fully Connected Layers:

Dense Layer 1:

- **64 neurons** with **ReLU activation** to interpret extracted patterns.

Dropout Layer 3:

- **30% dropout rate** to improve model robustness.

Dense Layer 2:

- **32 neurons** with **ReLU activation** for refined feature learning.

Output Layer:

- **1 neuron with Sigmoid activation**, providing a probability score for phishing detection.

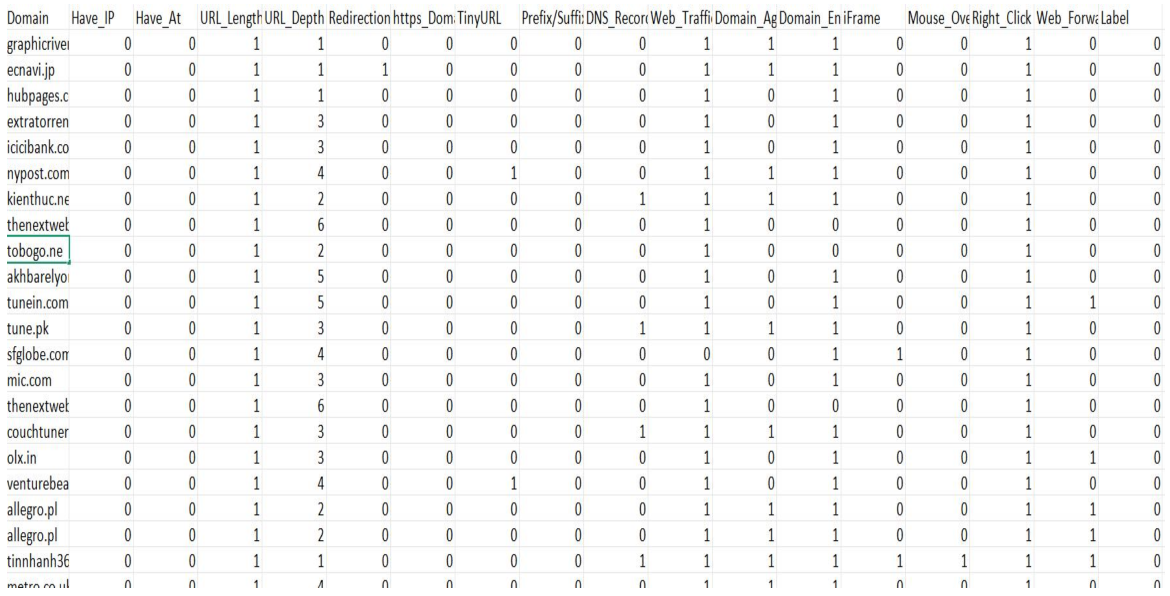
CHAPTER 4

IMPLEMENTATIONS AND RESULTS

CHAPTER 4

IMPLEMENTATIONS AND RESULTS

4.1 DATASET



Domain	Have_IP	Have_At	URL_Length	URL_Depth	Redirection	https_Dom	TinyURL	Prefix/Suffi	DNS_Recor	Web_Traffi	Domain_Ag	Domain_En	iFrame	Mouse_Ove	Right_Click	Web_Forwa	Label
graphicrivei	0	0	1	1	0	0	0	0	0	1	1	1	0	0	1	0	0
ecnavi.jp	0	0	1	1	1	0	0	0	0	1	1	1	0	0	1	0	0
hubpages.c	0	0	1	1	0	0	0	0	0	1	0	1	0	0	1	0	0
extratorren	0	0	1	3	0	0	0	0	0	1	0	1	0	0	1	0	0
icicibank.co	0	0	1	3	0	0	0	0	0	1	0	1	0	0	1	0	0
nypost.com	0	0	1	4	0	0	1	0	0	1	1	1	0	0	1	0	0
kienthuc.ne	0	0	1	2	0	0	0	0	1	1	1	1	0	0	1	0	0
thenextwet	0	0	1	6	0	0	0	0	0	1	0	0	0	0	1	0	0
tobogo.ne	0	0	1	2	0	0	0	0	0	1	0	0	0	0	1	0	0
akhbarelyo	0	0	1	5	0	0	0	0	0	1	0	1	0	0	1	0	0
tunein.com	0	0	1	5	0	0	0	0	0	1	0	1	0	0	1	1	0
tune.pk	0	0	1	3	0	0	0	0	1	1	1	1	0	0	1	0	0
sfglobe.com	0	0	1	4	0	0	0	0	0	0	0	1	1	0	1	0	0
mic.com	0	0	1	3	0	0	0	0	0	1	0	1	0	0	1	0	0
thenextwet	0	0	1	6	0	0	0	0	0	1	0	0	0	0	1	0	0
couchtuner	0	0	1	3	0	0	0	0	1	1	1	1	0	0	1	0	0
olx.in	0	0	1	3	0	0	0	0	0	1	0	1	0	0	1	1	0
venturebea	0	0	1	4	0	0	1	0	0	1	0	1	0	0	1	0	0
allegro.pl	0	0	1	2	0	0	0	0	0	1	1	1	0	0	1	1	0
allegro.pl	0	0	1	2	0	0	0	0	0	1	1	1	0	0	1	1	0
tinnhanh36	0	0	1	1	0	0	0	0	1	1	1	1	1	1	1	1	0
metro.co.ul	0	0	1	4	0	0	0	0	0	1	1	1	0	0	1	0	0

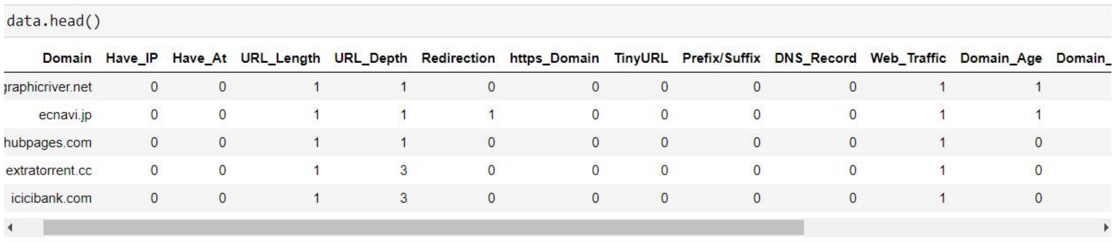
4.2 CODING DETAILS

Data Preprocessing: Imports necessary libraries for data preprocessing, including
pandas, numpy, NLTK, and scikit-learn. It reads the dataset from a CSV file and handles duplicates and null values. Text cleaning and normalization techniques such as converting to lowercase, handling contractions, and removing non-alphabetic
characters are applied to the text data.

4.2.1Importing Required Libraries

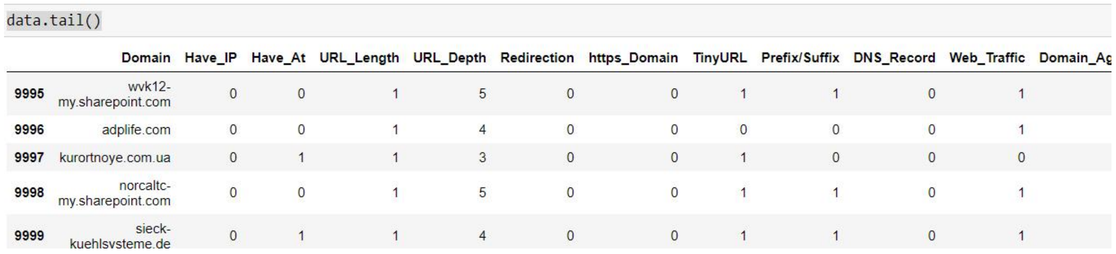
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

4.2.2 Loading and Exploring the Dataset



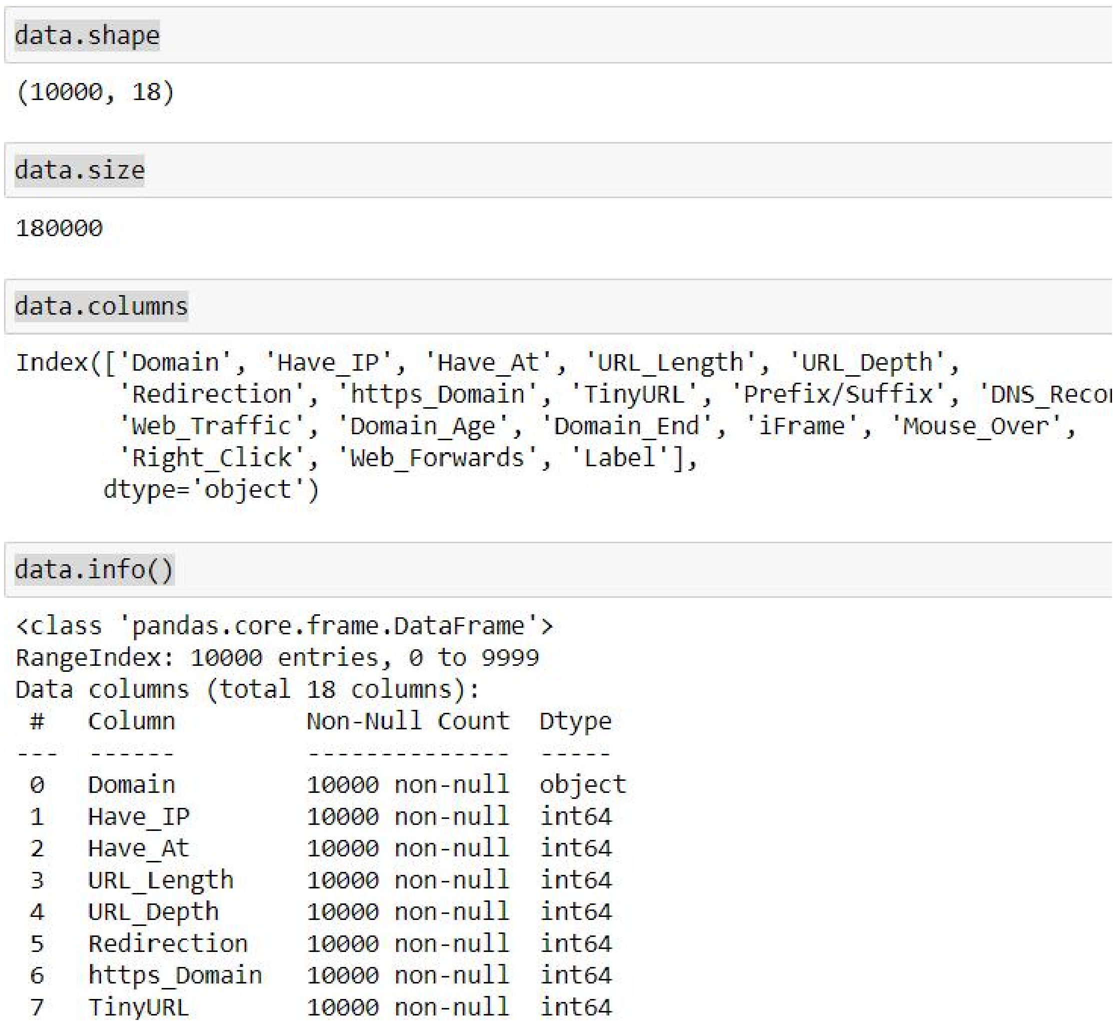
```
data.head()
```

Domain	Have_IP	Have_At	URL_Length	URL_Depth	Redirection	https_Domain	TinyURL	Prefix/Suffix	DNS_Record	Web_Traffic	Domain_Age	Domain_
graphicriver.net	0	0	1	1	0	0	0	0	0	1	1	
ecnavi.jp	0	0	1	1	1	0	0	0	0	1	1	
hubpages.com	0	0	1	1	0	0	0	0	0	1	0	
extratorrent.cc	0	0	1	3	0	0	0	0	0	1	0	
icicibank.com	0	0	1	3	0	0	0	0	0	1	0	



```
data.tail()
```

	Domain	Have_IP	Have_At	URL_Length	URL_Depth	Redirection	https_Domain	TinyURL	Prefix/Suffix	DNS_Record	Web_Traffic	Domain_Ag
9995	wvk12-my.sharepoint.com	0	0	1	5	0	0	1	1	0	1	
9996	adplife.com	0	0	1	4	0	0	0	0	0	1	
9997	kurortnoye.com.ua	0	1	1	3	0	0	1	0	0	0	
9998	norcaltc-my.sharepoint.com	0	0	1	5	0	0	1	1	0	1	
9999	sieck-kuehlsvsteme.de	0	1	1	4	0	0	1	1	0	1	



```
data.shape
```

```
(10000, 18)
```

```
data.size
```

```
180000
```

```
data.columns
```

```
Index(['Domain', 'Have_IP', 'Have_At', 'URL_Length', 'URL_Depth',
       'Redirection', 'https_Domain', 'TinyURL', 'Prefix/Suffix', 'DNS_Recoi
       'Web_Traffic', 'Domain_Age', 'Domain_End', 'iFrame', 'Mouse_Over',
       'Right_Click', 'Web_Forwards', 'Label'],
      dtype='object')
```

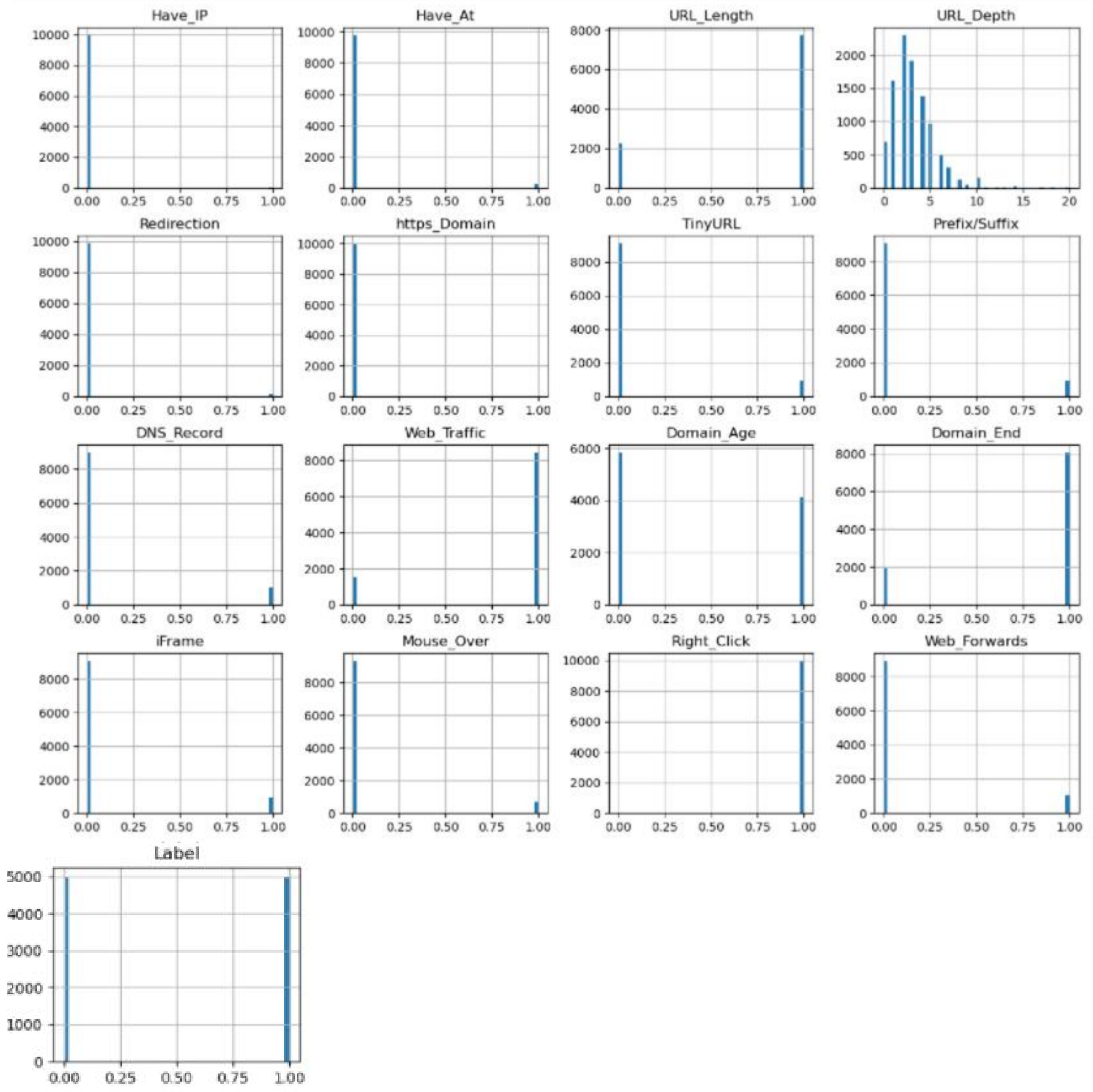
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 18 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Domain         10000 non-null  object
 1   Have_IP        10000 non-null  int64
 2   Have_At        10000 non-null  int64
 3   URL_Length     10000 non-null  int64
 4   URL_Depth      10000 non-null  int64
 5   Redirection    10000 non-null  int64
 6   https_Domain   10000 non-null  int64
 7   TinyURL        10000 non-null  int64
```

4.2.3 Data Visualization

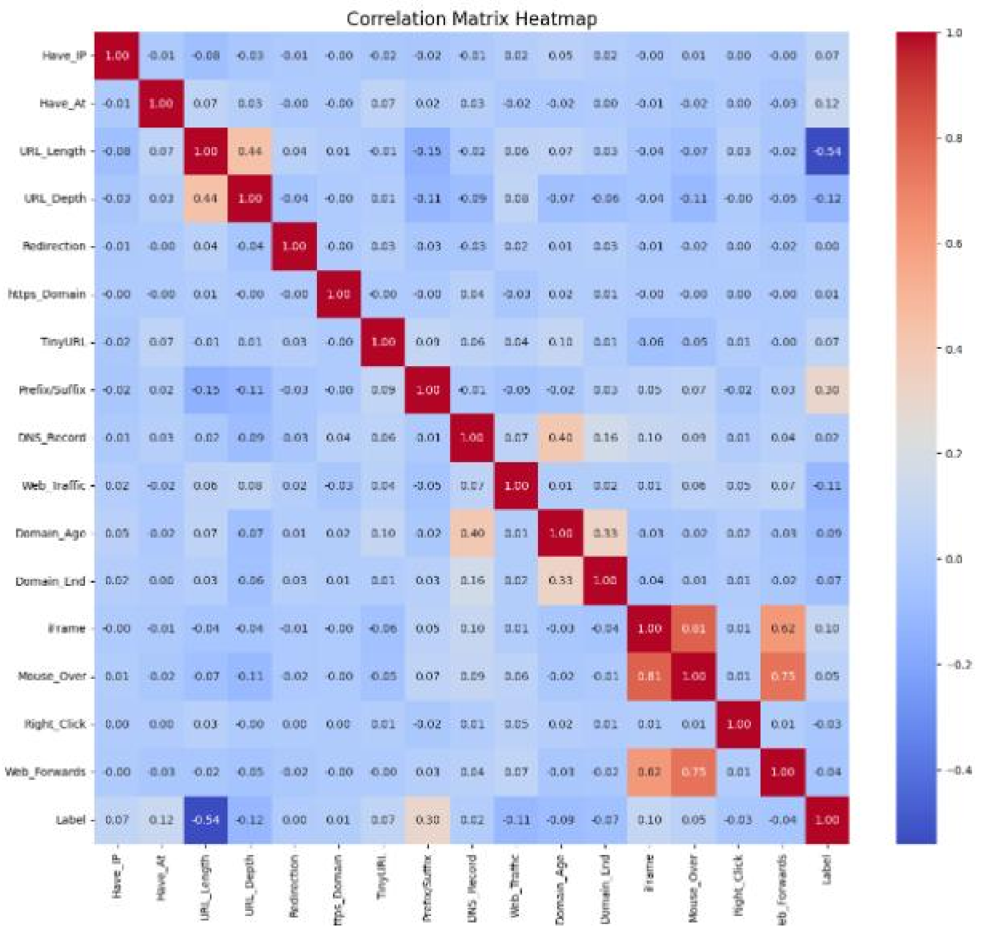
Histogram of Features

```
data.hist(bins = 50,figsize = (15,15))
plt.show()
```

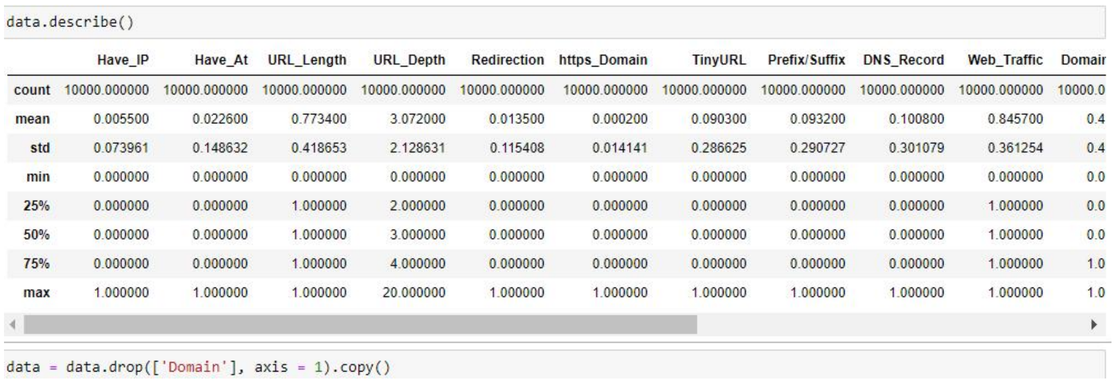


4.2.4 Correlation Heatmap

```
numeric_data = data.select_dtypes(include=["number"])
correlation_matrix = numeric_data.corr()
plt.figure(figsize=(15, 13))
sns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap="coolwarm", cbar=True)
plt.title("Correlation Matrix Heatmap", fontsize=16)
plt.show()
```



4.2.5 Removing the 'Domain' Column

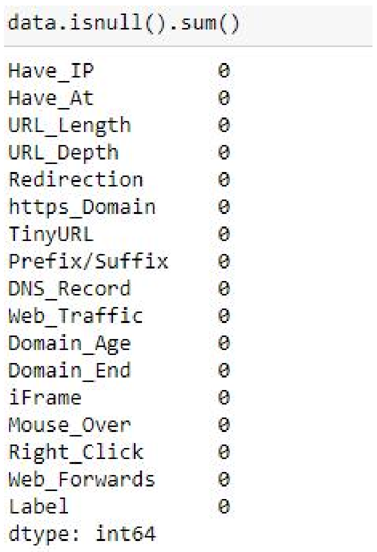


```
data.describe()
```

	Have_IP	Have_At	URL_Length	URL_Depth	Redirection	https_Domain	TinyURL	Prefix/Suffix	DNS_Record	Web_Traffic	Domair
count	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.0
mean	0.005500	0.022600	0.773400	3.072000	0.013500	0.000200	0.090300	0.093200	0.100800	0.845700	0.4
std	0.073961	0.148632	0.418653	2.128631	0.115408	0.014141	0.286625	0.290727	0.301079	0.361254	0.4
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0
25%	0.000000	0.000000	1.000000	2.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	0.0
50%	0.000000	0.000000	1.000000	3.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	0.0
75%	0.000000	0.000000	1.000000	4.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	1.0
max	1.000000	1.000000	1.000000	20.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.0

```
data = data.drop(['Domain'], axis = 1).copy()
```

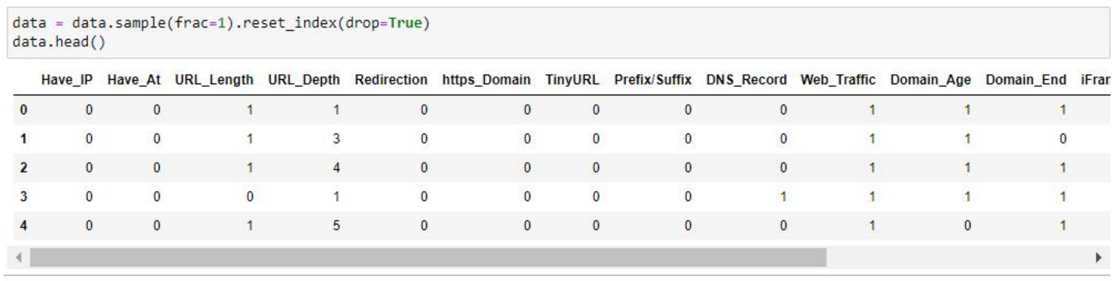
4.2.6 Checking for Missing Values



```
data.isnull().sum()
```

```
Have_IP          0
Have_At          0
URL_Length       0
URL_Depth        0
Redirection      0
https_Domain     0
TinyURL          0
Prefix/Suffix    0
DNS_Record       0
Web_Traffic      0
Domain_Age       0
Domain_End       0
iFrame           0
Mouse_Over       0
Right_Click      0
Web_Forwards     0
Label            0
dtype: int64
```

4.2.7 Shuffling the Data



```
data = data.sample(frac=1).reset_index(drop=True)
data.head()
```

	Have_IP	Have_At	URL_Length	URL_Depth	Redirection	https_Domain	TinyURL	Prefix/Suffix	DNS_Record	Web_Traffic	Domain_Age	Domain_End	iFrar
0	0	0	1	1	0	0	0	0	0	1	1	1	
1	0	0	1	3	0	0	0	0	0	1	1	0	
2	0	0	1	4	0	0	0	0	0	1	1	1	
3	0	0	0	1	0	0	0	0	1	1	1	1	
4	0	0	1	5	0	0	0	0	0	1	0	1	

4.2.8 Splitting Data into Features and Labels

```
y = data['Label']
X = data.drop('Label',axis=1)
X.shape, y.shape
```

```
((10000, 16), (10000,))
```

- `y` stores the **target labels** (0 = legitimate, 1 = phishing).
- `X` stores the **features** used for training.

4.2.9 Splitting 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 = 12)
X_train.shape, X_test.shape
```

```
((8000, 16), (2000, 16))
```

```
from sklearn.metrics import accuracy_score
```

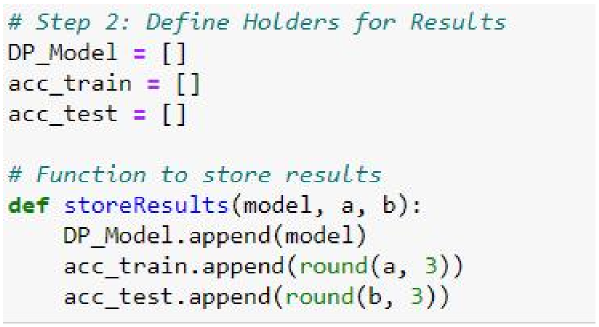
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)
```

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from tensorflow.keras import Sequential, Model
from tensorflow.keras.layers import Dense, Dropout, Input, Concatenate, Conv1D, Flatten, BatchNormalization
from pytorch_tabnet.tab_model import TabNetClassifier
import torch
import matplotlib.pyplot as plt
```

4.2.10Standardizing the Data

```
# Standardize features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

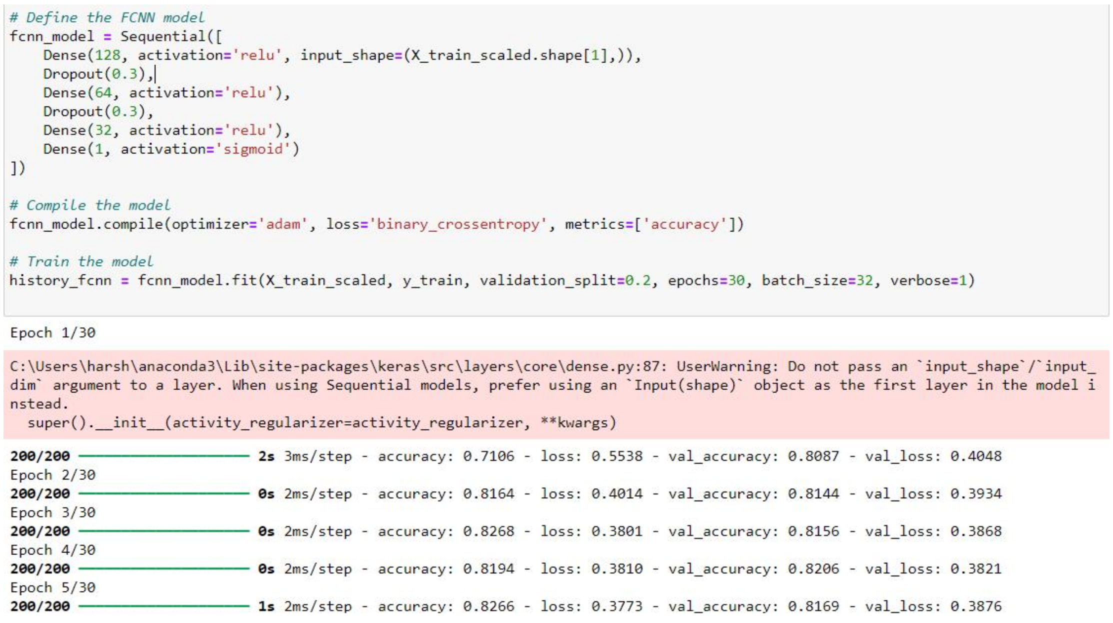
4.2.11 Function to Store Model Results



```
# Step 2: Define Holders for Results
DP_Model = []
acc_train = []
acc_test = []

# Function to store results
def storeResults(model, a, b):
    DP_Model.append(model)
    acc_train.append(round(a, 3))
    acc_test.append(round(b, 3))
```

4.3 Fully Connected Neural Network (FCNN)

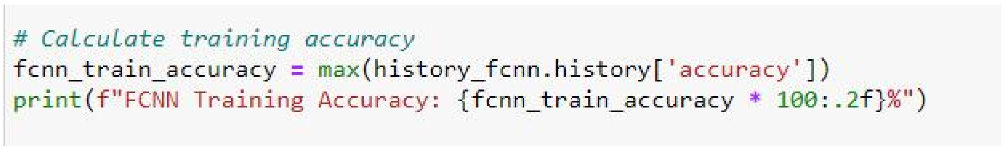
4.3.1 Defining the FCNN Model



```
# Define the FCNN model
fcnn_model = Sequential([
    Dense(128, activation='relu', input_shape=(X_train_scaled.shape[1],)),
    Dropout(0.3),
    Dense(64, activation='relu'),
    Dropout(0.3),
    Dense(32, activation='relu'),
    Dense(1, activation='sigmoid')
])

# Compile the model
fcnn_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
history_fcnn = fcnn_model.fit(X_train_scaled, y_train, validation_split=0.2, epochs=30, batch_size=32, verbose=1)
```

```
Epoch 1/30
C:\Users\harsh\anaconda3\Lib\site-packages\keras\src\layers\core\dense.py:87: UserWarning: Do not pass an `input_shape`/`input_
dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model i
nstead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
200/200 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.7106 - loss: 0.5538 - val_accuracy: 0.8087 - val_loss: 0.4048
Epoch 2/30
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8164 - loss: 0.4014 - val_accuracy: 0.8144 - val_loss: 0.3934
Epoch 3/30
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8268 - loss: 0.3801 - val_accuracy: 0.8156 - val_loss: 0.3868
Epoch 4/30
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8194 - loss: 0.3810 - val_accuracy: 0.8206 - val_loss: 0.3821
Epoch 5/30
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8266 - loss: 0.3773 - val_accuracy: 0.8169 - val_loss: 0.3876
```

4.3.2 Training the Model



```
# Calculate training accuracy
fcnn_train_accuracy = max(history_fcnn.history['accuracy'])
print(f"FCNN Training Accuracy: {fcnn_train_accuracy * 100:.2f}%")
```

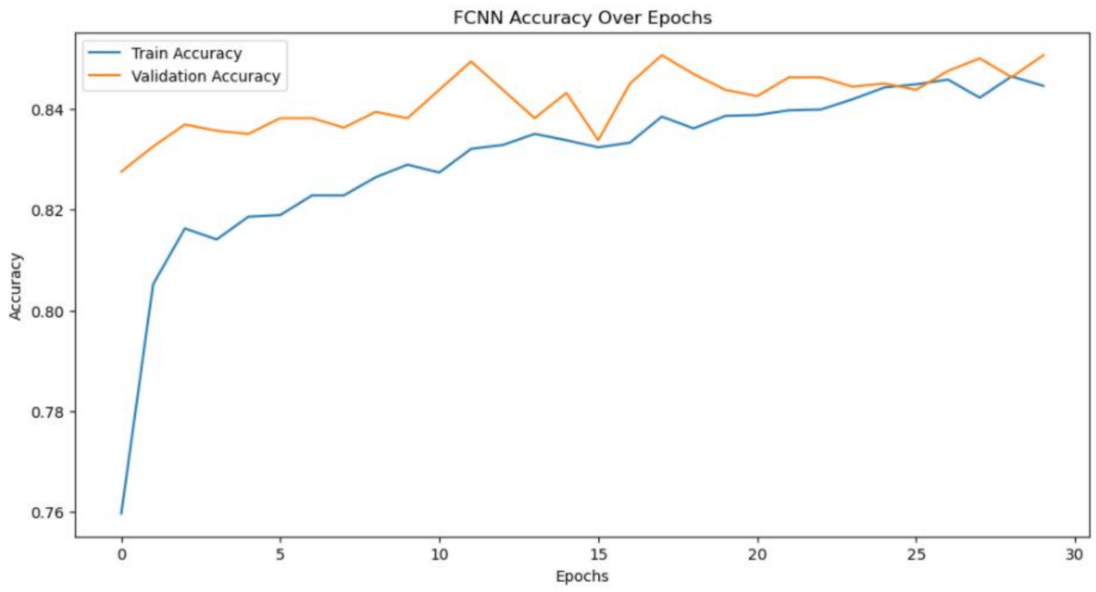
4.3.3 Evaluating the Model

```
# Calculate training accuracy
fcnn_train_accuracy = max(history_fcnn.history['accuracy'])
print(f"FCNN Training Accuracy: {fcnn_train_accuracy * 100:.2f}%")

# Evaluate the model on the test set
fcnn_test_loss, fcnn_test_accuracy = fcnn_model.evaluate(X_test_scaled, y_test, verbose=0)
print(f"FCNN Test Accuracy: {fcnn_test_accuracy * 100:.2f}%")
```

```
FCNN Training Accuracy: 84.64%
FCNN Test Accuracy: 86.20%
```

```
# Store results
storeResults("FCNN", fcnn_train_accuracy, fcnn_test_accuracy)
```

```
# Plot training history for FCNN
plt.figure(figsize=(12, 6))
plt.plot(history_fcnn.history['accuracy'], label='Train Accuracy')
plt.plot(history_fcnn.history['val_accuracy'], label='Validation Accuracy')
plt.title('FCNN Accuracy Over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



4.4 Wide & Deep Model

4.4.1 Defining the Model

```
# Define the Wide & Deep model
wide_input = Input(shape=(X_train_scaled.shape[1],), name="wide_input")
wide_output = Dense(1, activation="sigmoid")(wide_input)
deep_input = Input(shape=(X_train_scaled.shape[1],), name="deep_input")
x = Dense(128, activation='relu')(deep_input)
x = Dropout(0.3)(x)
x = Dense(64, activation='relu')(x)
x = Dropout(0.3)(x)
deep_output = Dense(1, activation="sigmoid")(x)
merged = Concatenate()([wide_output, deep_output])
final_output = Dense(1, activation="sigmoid")(merged)
wide_deep_model = Model(inputs=[wide_input, deep_input], outputs=final_output)

# Compile the Wide & Deep model
wide_deep_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the Wide & Deep model
history_wide_deep = wide_deep_model.fit([X_train_scaled, X_train_scaled], y_train, validation_split=0.2,
                                        epochs=30, batch_size=32, verbose=1)
```

```
Epoch 1/30
200/200 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.7113 - loss: 0.6215 - val_accuracy: 0.7919 - val_loss: 0.5330
Epoch 2/30
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8071 - loss: 0.5263 - val_accuracy: 0.8025 - val_loss: 0.5069
Epoch 3/30
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8220 - loss: 0.4967 - val_accuracy: 0.8062 - val_loss: 0.4844
Epoch 4/30
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8125 - loss: 0.4843 - val_accuracy: 0.8075 - val_loss: 0.4679
Epoch 5/30
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8175 - loss: 0.4600 - val_accuracy: 0.8163 - val_loss: 0.4549
Epoch 6/30
```

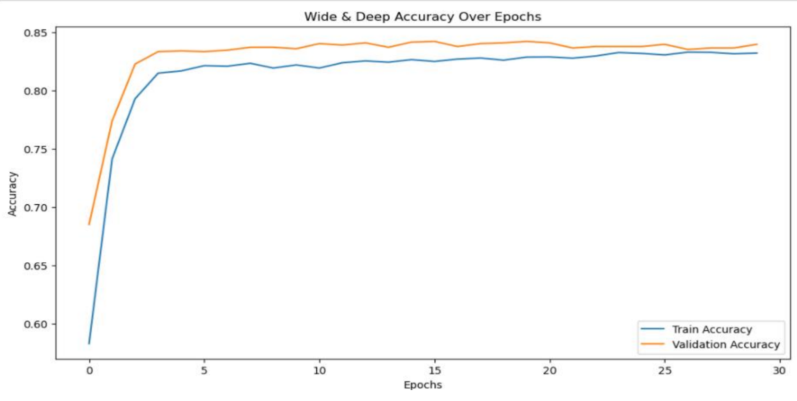
4.4.2 TRAINING AND MODEL EVALUATION

```
# Calculate training accuracy for Wide & Deep model
wide_deep_train_accuracy = max(history_wide_deep.history['accuracy'])
print(f"Wide & Deep Training Accuracy: {wide_deep_train_accuracy * 100:.2f}%")

# Evaluate the Wide & Deep model on the test set
wide_deep_test_loss, wide_deep_test_accuracy = wide_deep_model.evaluate([X_test_scaled, X_test_scaled], y_test, verbose=0)
print(f"Wide & Deep Test Accuracy: {wide_deep_test_accuracy * 100:.2f}%")
```

```
Wide & Deep Training Accuracy: 83.27%
Wide & Deep Test Accuracy: 85.05%
```

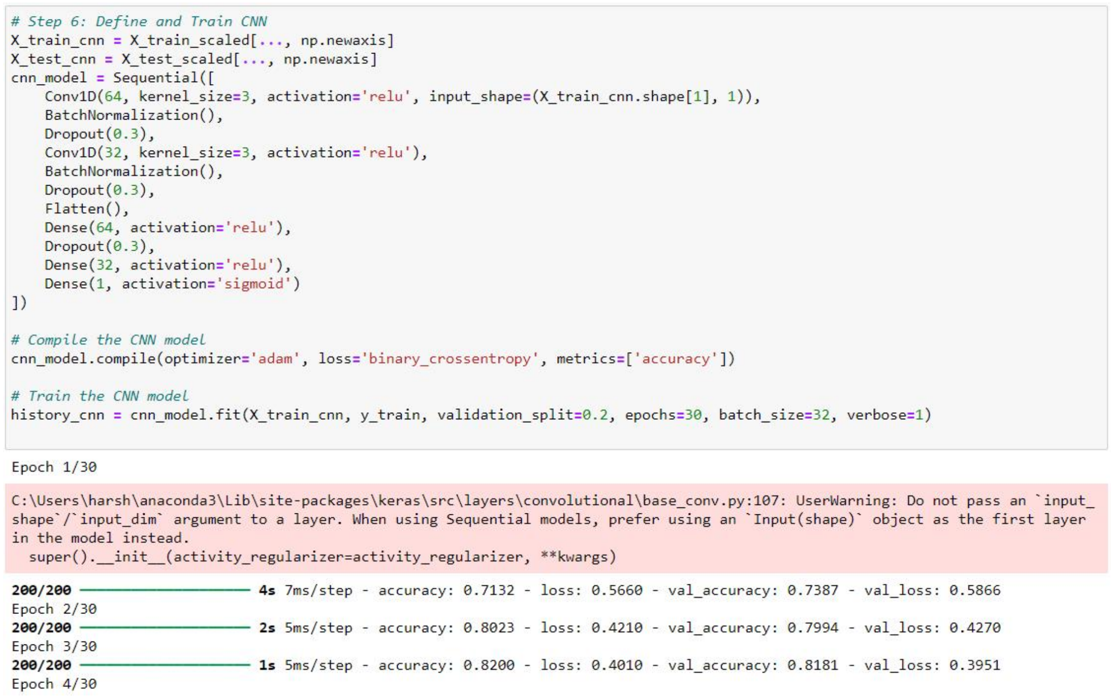
```
# Store Wide & Deep results
storeResults("Wide & Deep", wide_deep_train_accuracy, wide_deep_test_accuracy)
```

```
# Plot training history for Wide & Deep
plt.figure(figsize=(12, 6))
plt.plot(history_wide_deep.history['accuracy'], label='Train Accuracy')
plt.plot(history_wide_deep.history['val_accuracy'], label='Validation Accuracy')
plt.title('Wide & Deep Accuracy Over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



4.5 Convolutional Neural Network (CNN)

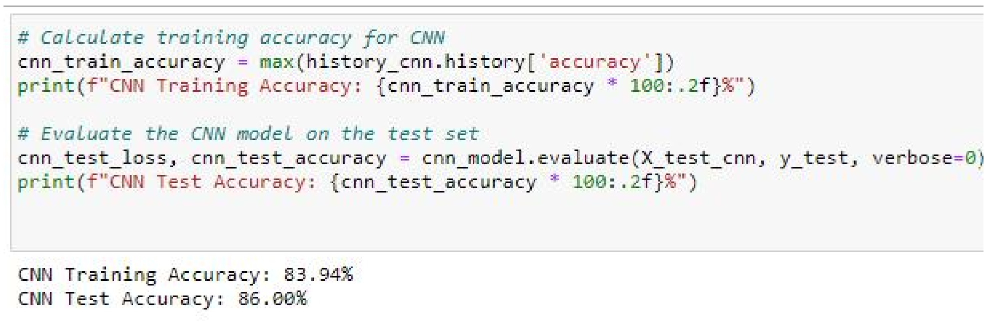
4.5.1 Defining the CNN Model



```
# Step 6: Define and Train CNN
X_train_cnn = X_train_scaled[..., np.newaxis]
X_test_cnn = X_test_scaled[..., np.newaxis]
cnn_model = Sequential([
    Conv1D(64, kernel_size=3, activation='relu', input_shape=(X_train_cnn.shape[1], 1)),
    BatchNormalization(),
    Dropout(0.3),
    Conv1D(32, kernel_size=3, activation='relu'),
    BatchNormalization(),
    Dropout(0.3),
    Flatten(),
    Dense(64, activation='relu'),
    Dropout(0.3),
    Dense(32, activation='relu'),
    Dense(1, activation='sigmoid')
])

# Compile the CNN model
cnn_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the CNN model
history_cnn = cnn_model.fit(X_train_cnn, y_train, validation_split=0.2, epochs=30, batch_size=32, verbose=1)
```

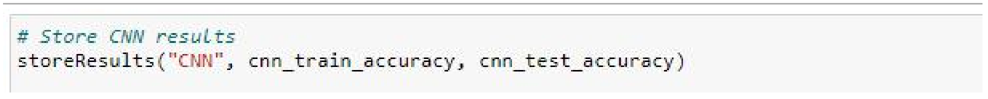
```
Epoch 1/30
C:\Users\harsh\anaconda3\Lib\site-packages\keras\src\layers\convolutional\base_conv.py:107: UserWarning: Do not pass an `input_
shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer
in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
200/200 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - accuracy: 0.7132 - loss: 0.5660 - val_accuracy: 0.7387 - val_loss: 0.5866
Epoch 2/30
200/200 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.8023 - loss: 0.4210 - val_accuracy: 0.7994 - val_loss: 0.4270
Epoch 3/30
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.8200 - loss: 0.4010 - val_accuracy: 0.8181 - val_loss: 0.3951
Epoch 4/30
```

4.5.2Training and Evaluation



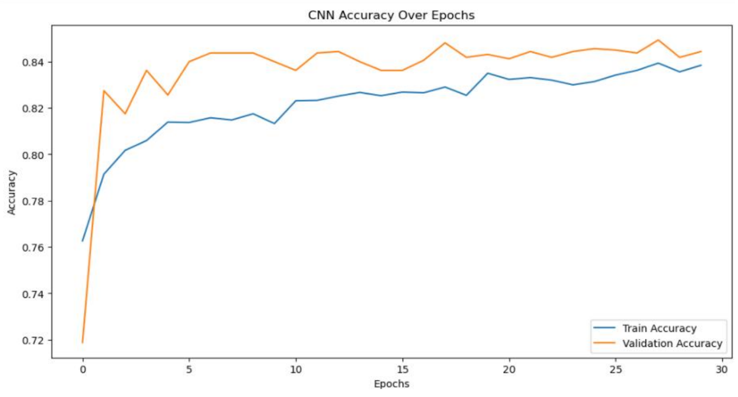
```
# Calculate training accuracy for CNN
cnn_train_accuracy = max(history_cnn.history['accuracy'])
print(f"CNN Training Accuracy: {cnn_train_accuracy * 100:.2f}%")

# Evaluate the CNN model on the test set
cnn_test_loss, cnn_test_accuracy = cnn_model.evaluate(X_test_cnn, y_test, verbose=0)
print(f"CNN Test Accuracy: {cnn_test_accuracy * 100:.2f}%")
```

```
CNN Training Accuracy: 83.94%
CNN Test Accuracy: 86.00%
```



```
# Store CNN results
storeResults("CNN", cnn_train_accuracy, cnn_test_accuracy)
```

```
# Plot training history for CNN
plt.figure(figsize=(12, 6))
plt.plot(history_cnn.history['accuracy'], label='Train Accuracy')
plt.plot(history_cnn.history['val_accuracy'], label='Validation Accuracy')
plt.title('CNN Accuracy Over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

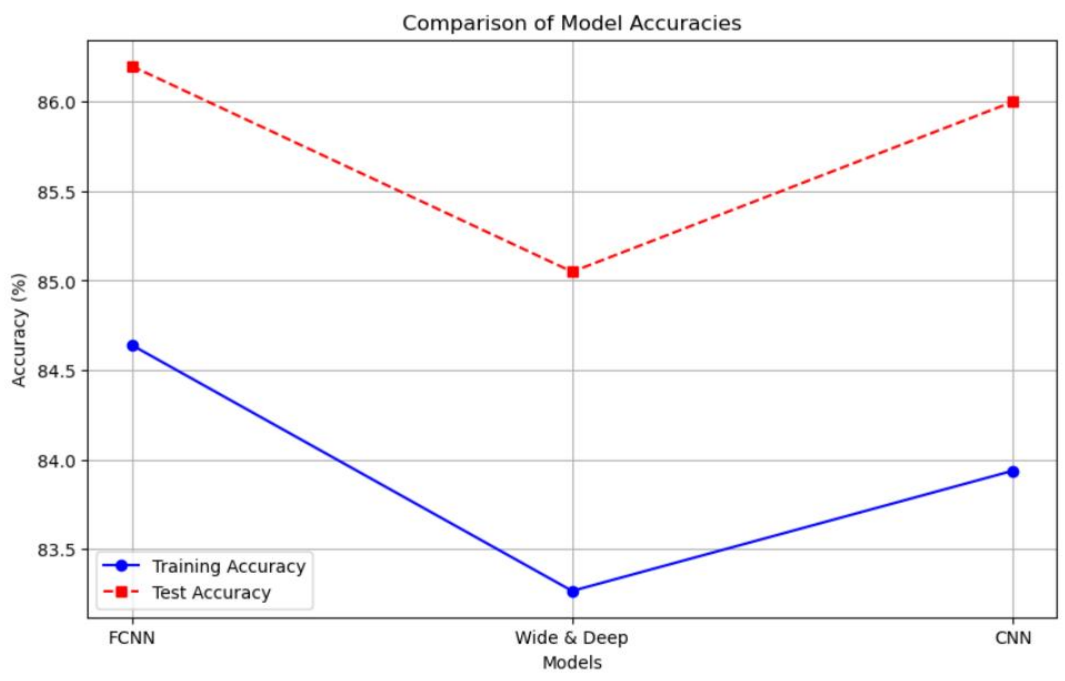


4.6 Comparing Model Performance

```
# Sort results by Test Accuracy
sorted_results = results.sort_values(by="Test Accuracy", ascending=False)
print("\nModel Performance Comparison (Sorted by Test Accuracy):")
print(sorted_results)
```

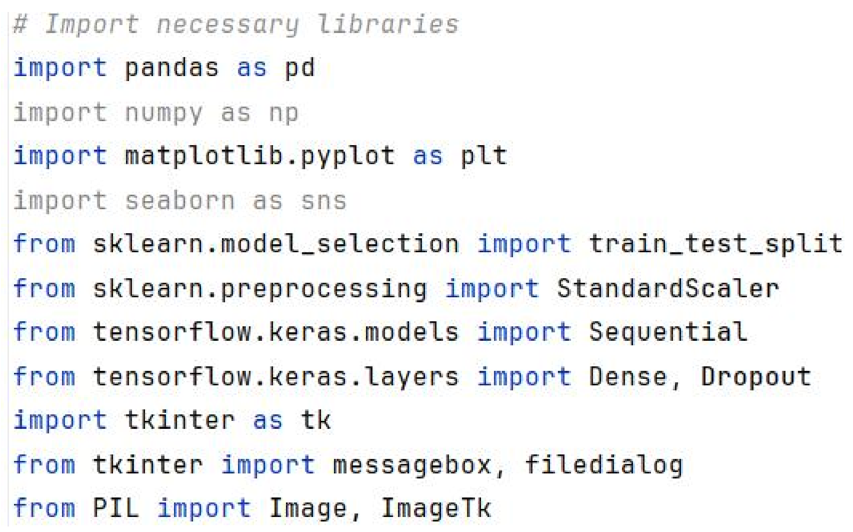
```
Model Performance Comparison (Sorted by Test Accuracy):
         Model  Train Accuracy  Test Accuracy
0         FCNN        0.846406         0.8620
2          CNN        0.839375         0.8600
1  Wide & Deep        0.832656         0.8505
```

```
# Model names
models = ["FCNN", "Wide & Deep", "CNN"]

# Training and Test Accuracies
train_accuracies = [fcnn_train_accuracy * 100, wide_deep_train_accuracy * 100, cnn_train_accuracy * 100]
test_accuracies = [fcnn_test_accuracy * 100, wide_deep_test_accuracy * 100, cnn_test_accuracy * 100]

# Create line plot
plt.figure(figsize=(10, 6))
plt.plot(models, train_accuracies, marker='o', linestyle='-', label='Training Accuracy', color='b')
plt.plot(models, test_accuracies, marker='s', linestyle='--', label='Test Accuracy', color='r')

# Add labels and title
plt.xlabel("Models")
plt.ylabel("Accuracy (%)")
plt.title("Comparison of Model Accuracies")
plt.legend()
plt.grid(True)

# Show plot
plt.show()
```


Comparison of Model Accuracies
Accuracy (%)
86.0
85.5
85.0
84.5
84.0
83.5
Training Accuracy
Test Accuracy
FCNN
Wide & Deep
CNN
Models

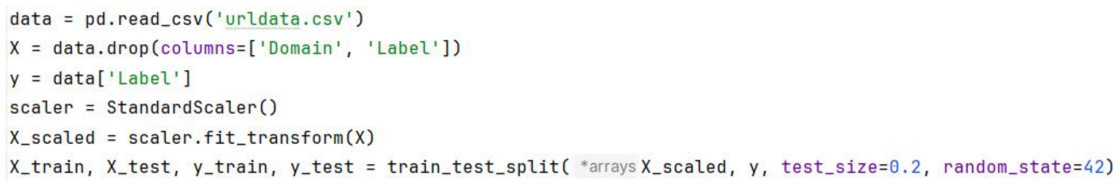
4.7INTERFACE DETAILS

4.7.1 Importing Required Libraries



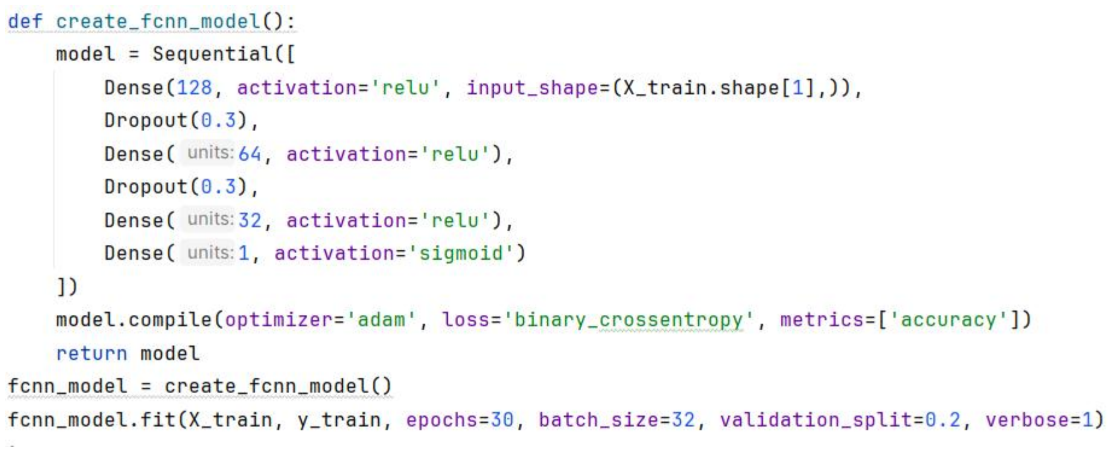
```
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
import tkinter as tk
from tkinter import messagebox, filedialog
from PIL import Image, ImageTk
```

4.7.2 Loading and Preprocessing the Dataset



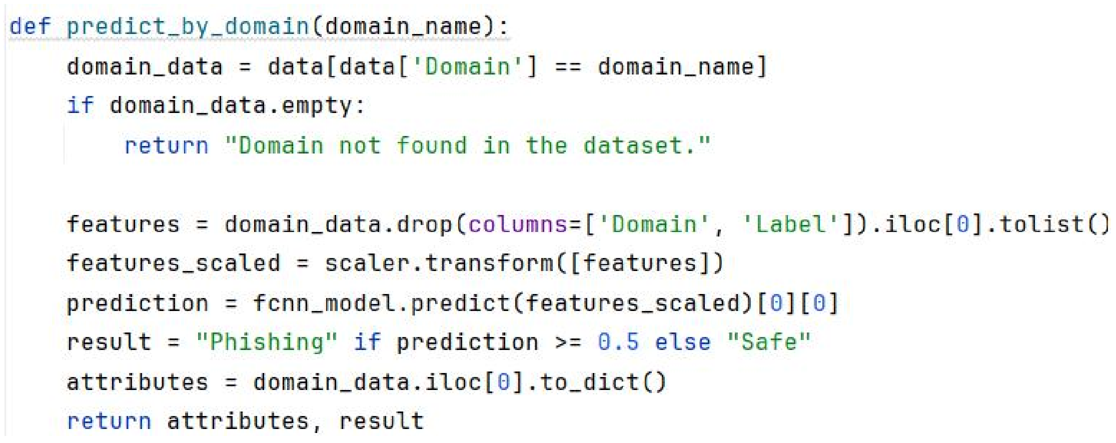
```
data = pd.read_csv('urldata.csv')
X = data.drop(columns=['Domain', 'Label'])
y = data['Label']
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split( *arrays X_scaled, y, test_size=0.2, random_state=42)
```

4.7.3 Defining and Training the FCNN Model



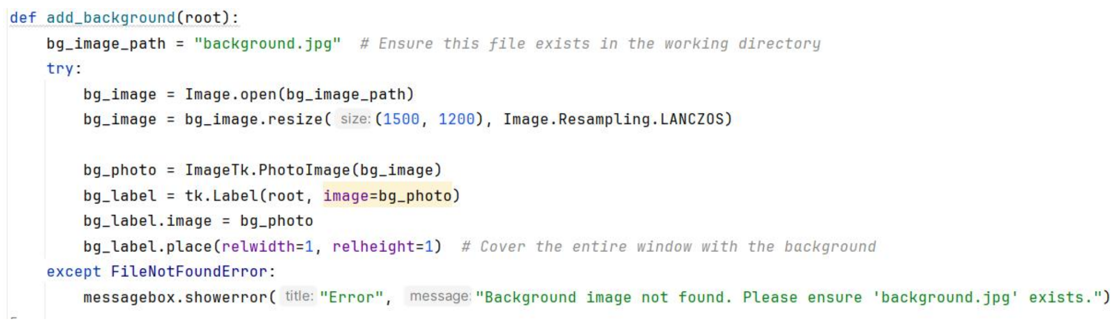
```
def create_fcnn_model():
    model = Sequential([
        Dense(128, activation='relu', input_shape=(X_train.shape[1],)),
        Dropout(0.3),
        Dense( units: 64, activation='relu'),
        Dropout(0.3),
        Dense( units: 32, activation='relu'),
        Dense( units: 1, activation='sigmoid')
    ])
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return model
fcnn_model = create_fcnn_model()
fcnn_model.fit(X_train, y_train, epochs=30, batch_size=32, validation_split=0.2, verbose=1)
```

4.7.4 Domain Prediction Function



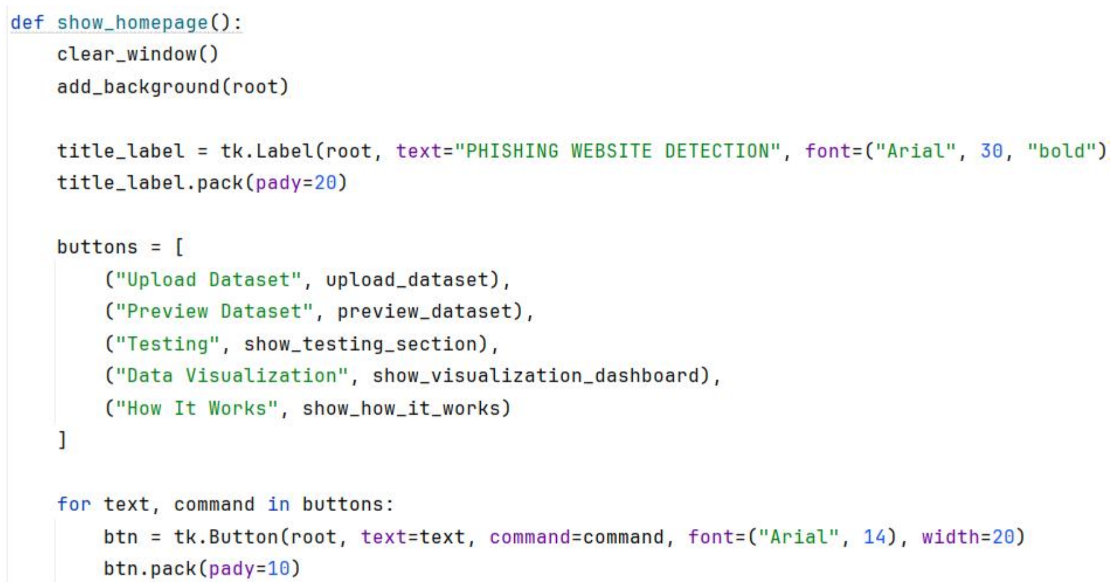
```python
def predict_by_domain(domain_name):
    domain_data = data[data['Domain'] == domain_name]
    if domain_data.empty:
        return "Domain not found in the dataset."

    features = domain_data.drop(columns=['Domain', 'Label']).iloc[0].tolist()
    features_scaled = scaler.transform([features])
    prediction = fcnn_model.predict(features_scaled)[0][0]
    result = "Phishing" if prediction >= 0.5 else "Safe"
    attributes = domain_data.iloc[0].to_dict()
    return attributes, result
```

4.7.5 Adding a Background Image



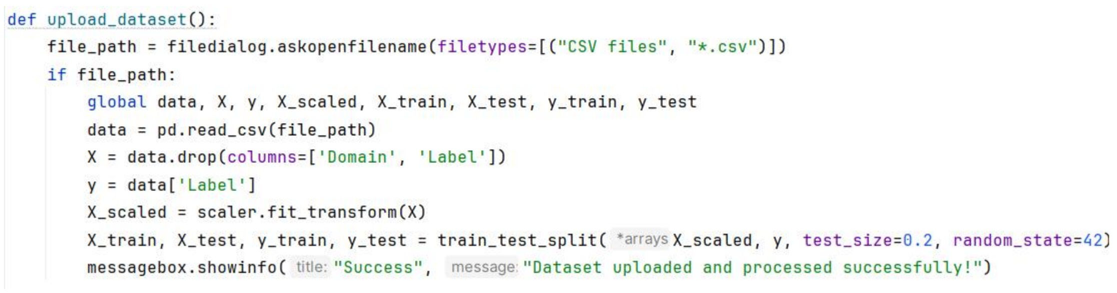
```python
def add_background(root):
    bg_image_path = "background.jpg"  # Ensure this file exists in the working directory
    try:
        bg_image = Image.open(bg_image_path)
        bg_image = bg_image.resize( size: (1500, 1200), Image.Resampling.LANCZOS)

        bg_photo = ImageTk.PhotoImage(bg_image)
        bg_label = tk.Label(root, image=bg_photo)
        bg_label.image = bg_photo
        bg_label.place(relwidth=1, relheight=1)  # Cover the entire window with the background
    except FileNotFoundError:
        messagebox.showerror( title: "Error",  message: "Background image not found. Please ensure 'background.jpg' exists.")
```

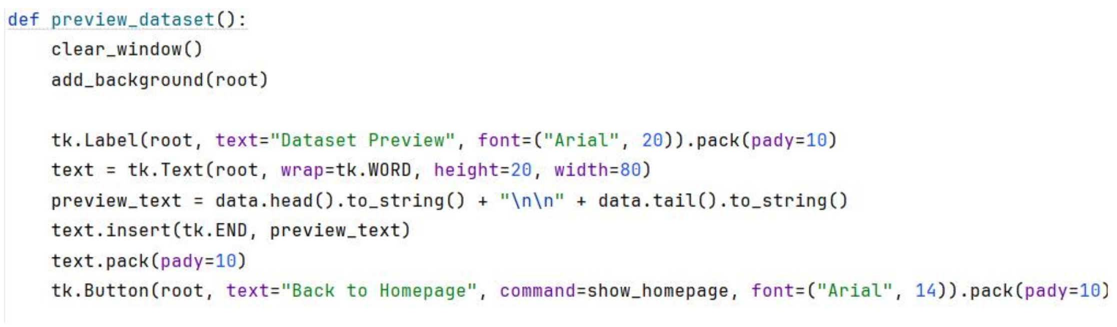
4.7.6 Creating the Main Application (GUI)



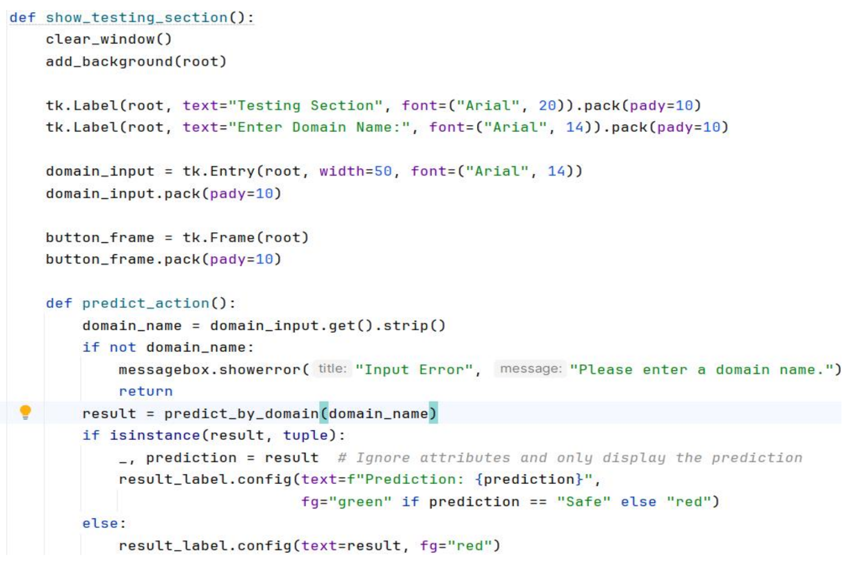
```python
def show_homepage():
    clear_window()
    add_background(root)

    title_label = tk.Label(root, text="PHISHING WEBSITE DETECTION", font=("Arial", 30, "bold"))
    title_label.pack(pady=20)

    buttons = [
        ("Upload Dataset", upload_dataset),
        ("Preview Dataset", preview_dataset),
        ("Testing", show_testing_section),
        ("Data Visualization", show_visualization_dashboard),
        ("How It Works", show_how_it_works)
    ]

    for text, command in buttons:
        btn = tk.Button(root, text=text, command=command, font=("Arial", 14), width=20)
        btn.pack(pady=10)
```

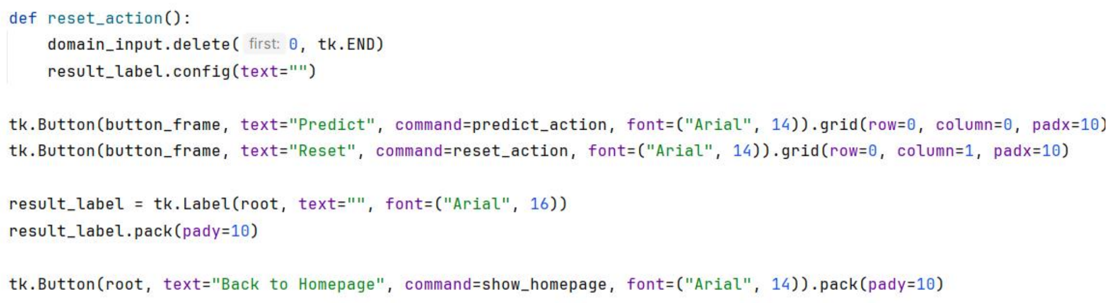
4.7.7 Uploading a New Dataset

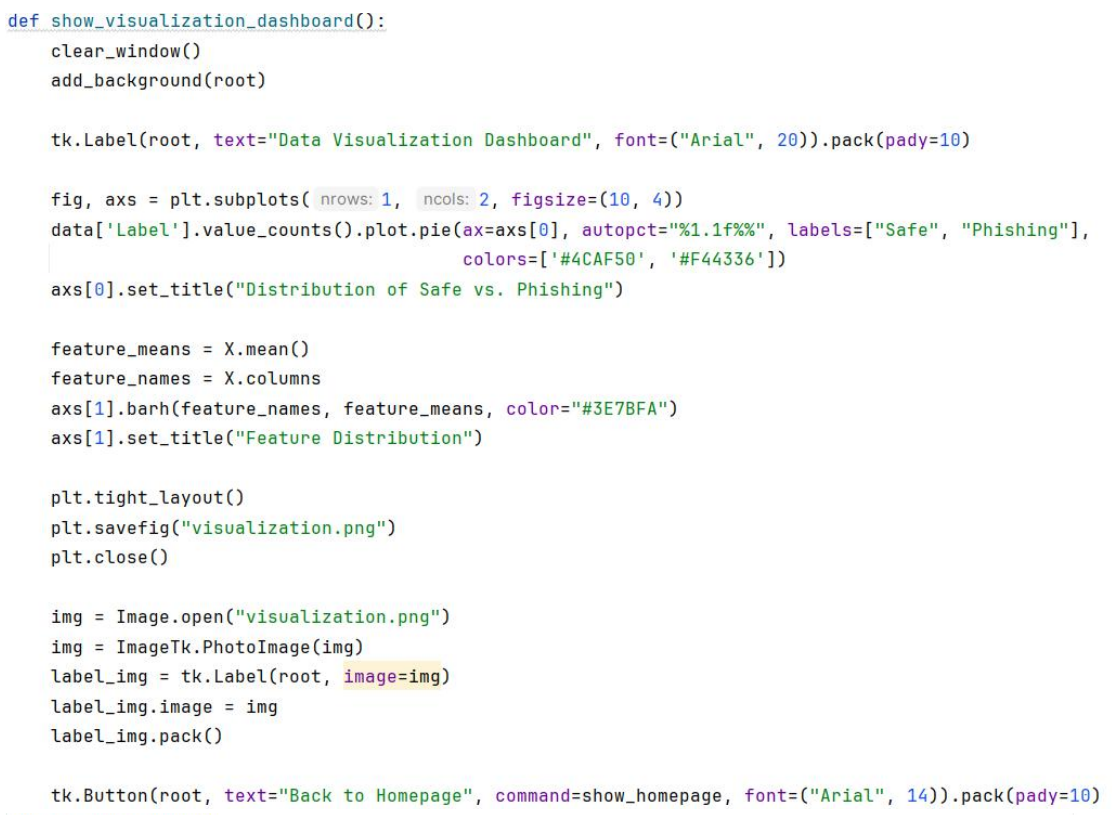

```
def upload_dataset():
    file_path = filedialog.askopenfilename(filetypes=[("CSV files", "*.csv")])
    if file_path:
        global data, X, y, X_scaled, X_train, X_test, y_train, y_test
        data = pd.read_csv(file_path)
        X = data.drop(columns=['Domain', 'Label'])
        y = data['Label']
        X_scaled = scaler.fit_transform(X)
        X_train, X_test, y_train, y_test = train_test_split( *arrays X_scaled, y, test_size=0.2, random_state=42)
        messagebox.showinfo( title: "Success",  message: "Dataset uploaded and processed successfully!")
```


```
def preview_dataset():
    clear_window()
    add_background(root)

    tk.Label(root, text="Dataset Preview", font=("Arial", 20)).pack(pady=10)
    text = tk.Text(root, wrap=tk.WORD, height=20, width=80)
    preview_text = data.head().to_string() + "\n\n" + data.tail().to_string()
    text.insert(tk.END, preview_text)
    text.pack(pady=10)
    tk.Button(root, text="Back to Homepage", command=show_homepage, font=("Arial", 14)).pack(pady=10)
```

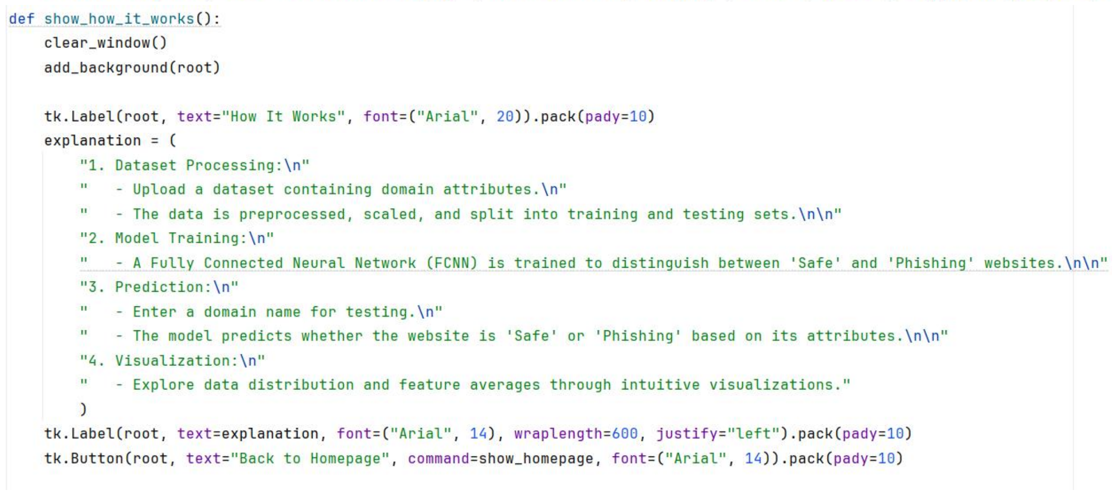
4.7.8 Testing a Domain


```
def show_testing_section():
    clear_window()
    add_background(root)

    tk.Label(root, text="Testing Section", font=("Arial", 20)).pack(pady=10)
    tk.Label(root, text="Enter Domain Name:", font=("Arial", 14)).pack(pady=10)

    domain_input = tk.Entry(root, width=50, font=("Arial", 14))
    domain_input.pack(pady=10)

    button_frame = tk.Frame(root)
    button_frame.pack(pady=10)

    def predict_action():
        domain_name = domain_input.get().strip()
        if not domain_name:
            messagebox.showerror( title: "Input Error",  message: "Please enter a domain name.")
            return
        result = predict_by_domain(domain_name)
        if isinstance(result, tuple):
            _, prediction = result  # Ignore attributes and only display the prediction
            result_label.config(text=f"Prediction: {prediction}",
                                fg="green" if prediction == "Safe" else "red")
        else:
            result_label.config(text=result, fg="red")
```


```
def reset_action():
    domain_input.delete( first: 0, tk.END)
    result_label.config(text="")

tk.Button(button_frame, text="Predict", command=predict_action, font=("Arial", 14)).grid(row=0, column=0, padx=10)
tk.Button(button_frame, text="Reset", command=reset_action, font=("Arial", 14)).grid(row=0, column=1, padx=10)

result_label = tk.Label(root, text="", font=("Arial", 16))
result_label.pack(pady=10)

tk.Button(root, text="Back to Homepage", command=show_homepage, font=("Arial", 14)).pack(pady=10)
```

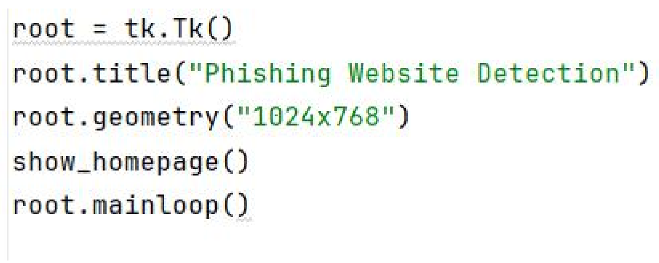


```python
def show_visualization_dashboard():
    clear_window()
    add_background(root)

    tk.Label(root, text="Data Visualization Dashboard", font=("Arial", 20)).pack(pady=10)

    fig, axs = plt.subplots( nrows: 1,  ncols: 2, figsize=(10, 4))
    data['Label'].value_counts().plot.pie(ax=axs[0], autopct="%1.1f%%", labels=["Safe", "Phishing"],
                                          colors=['#4CAF50', '#F44336'])
    axs[0].set_title("Distribution of Safe vs. Phishing")

    feature_means = X.mean()
    feature_names = X.columns
    axs[1].barh(feature_names, feature_means, color="#3E7BFA")
    axs[1].set_title("Feature Distribution")

    plt.tight_layout()
    plt.savefig("visualization.png")
    plt.close()

    img = Image.open("visualization.png")
    img = ImageTk.PhotoImage(img)
    label_img = tk.Label(root, image=img)
    label_img.image = img
    label_img.pack()

    tk.Button(root, text="Back to Homepage", command=show_homepage, font=("Arial", 14)).pack(pady=10)
```



```python
def show_how_it_works():
    clear_window()
    add_background(root)

    tk.Label(root, text="How It Works", font=("Arial", 20)).pack(pady=10)
    explanation = (
        "1. Dataset Processing:\n"
        "   - Upload a dataset containing domain attributes.\n"
        "   - The data is preprocessed, scaled, and split into training and testing sets.\n\n"
        "2. Model Training:\n"
        "   - A Fully Connected Neural Network (FCNN) is trained to distinguish between 'Safe' and 'Phishing' websites.\n\n"
        "3. Prediction:\n"
        "   - Enter a domain name for testing.\n"
        "   - The model predicts whether the website is 'Safe' or 'Phishing' based on its attributes.\n\n"
        "4. Visualization:\n"
        "   - Explore data distribution and feature averages through intuitive visualizations."
    )
    tk.Label(root, text=explanation, font=("Arial", 14), wraplength=600, justify="left").pack(pady=10)
    tk.Button(root, text="Back to Homepage", command=show_homepage, font=("Arial", 14)).pack(pady=10)
```

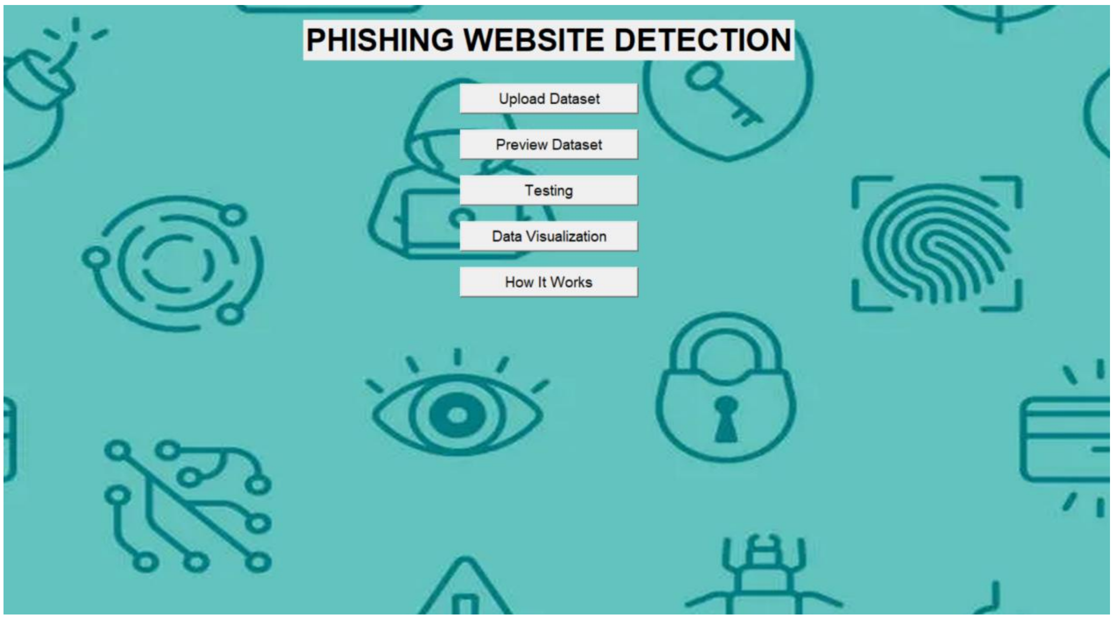
4.7.9 Running the Application



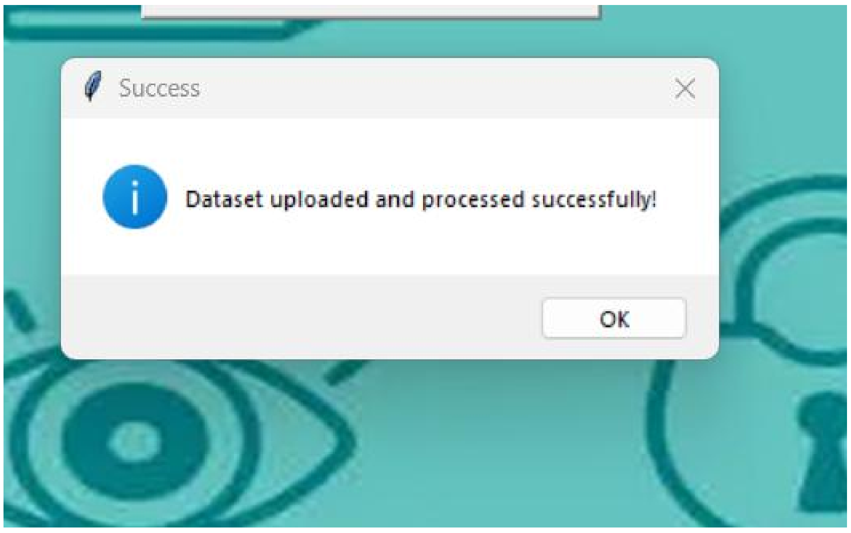
```python
root = tk.Tk()
root.title("Phishing Website Detection")
root.geometry("1024x768")
show_homepage()
root.mainloop()
```

4.8 INTERFACE SCREENSHOTS

4.8.1MAIN INTERFACE

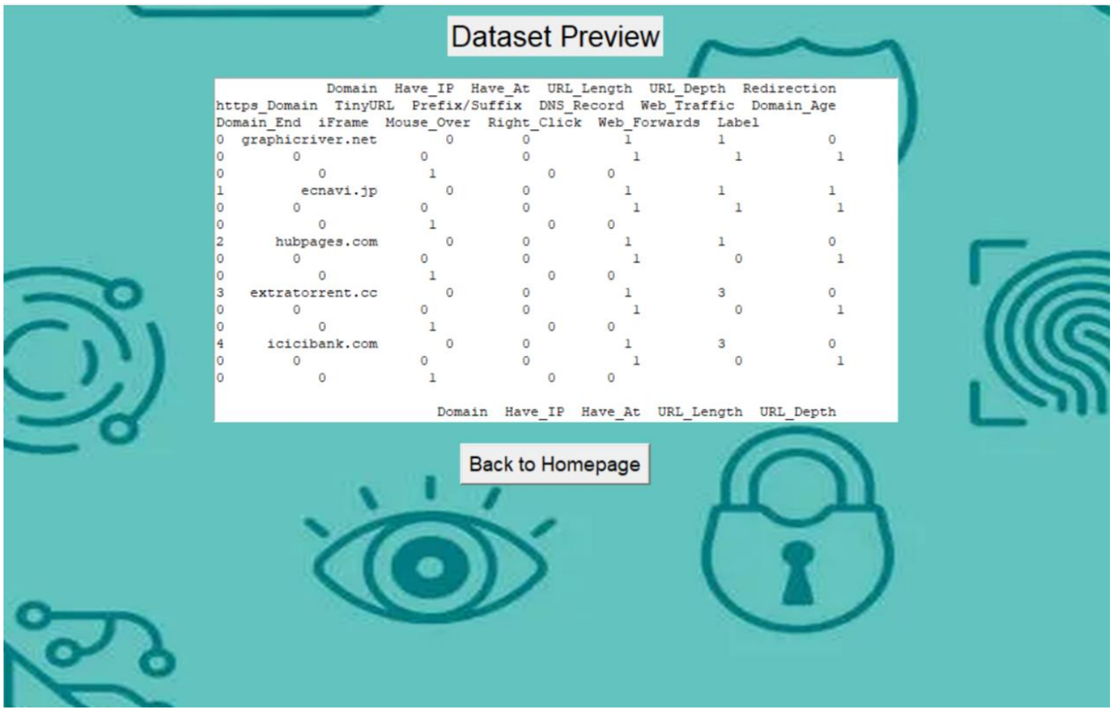


4.8.2 UPLOAD DATASET



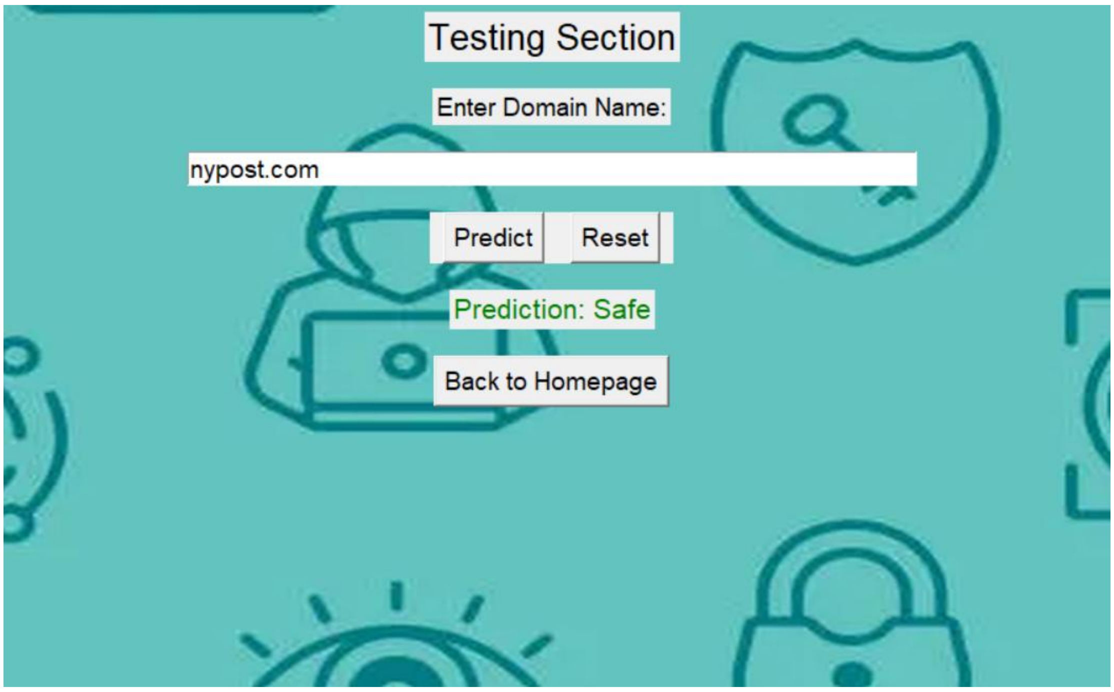
Dataset uploaded and processed successfully!

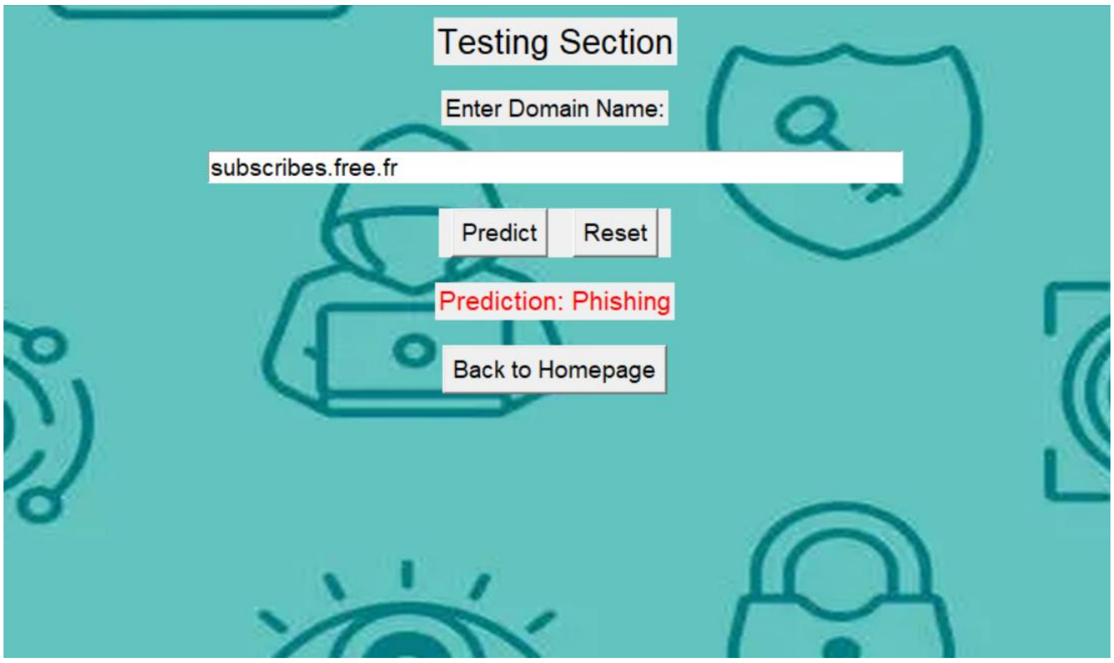
4.8.3 DATASET PREVIEW



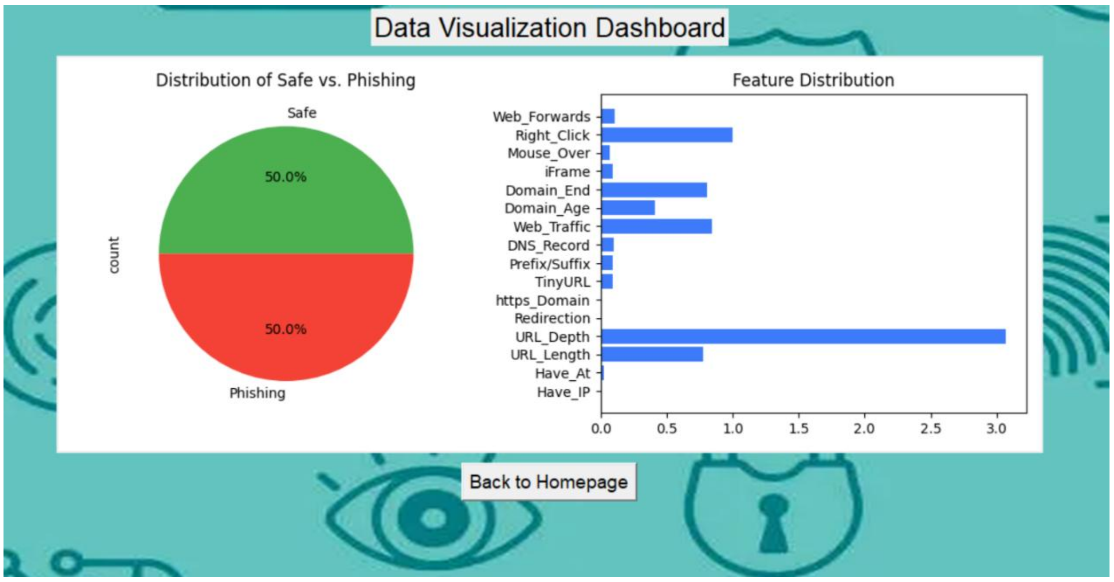
```
              Domain  Have_IP  Have_At  URL_Length  URL_Depth  Redirection
https_Domain  TinyURL  Prefix/Suffix  DNS_Record  Web_Traffic  Domain_Age
Domain_End  iFrame  Mouse_Over  Right_Click  Web_Forwards  Label
0  graphicriver.net        0        0           1          1            0
0       0              0        0           1          1            1
0          0           1             0      0
1        ecnavi.jp        0        0           1          1            1
0       0              0        0           1          1            1
0          0           1             0      0
2      hubpages.com        0        0           1          1            0
0       0              0        0           1          0            1
0          0           1             0      0
3   extratorrent.cc        0        0           1          3            0
0       0              0        0           1          0            1
0          0           1             0      0
4     icicibank.com        0        0           1          3            0
0       0              0        0           1          0            1
0          0           1             0      0

                             Domain  Have_IP  Have_At  URL_Length  URL_Depth
```

4.8.4 TESTING

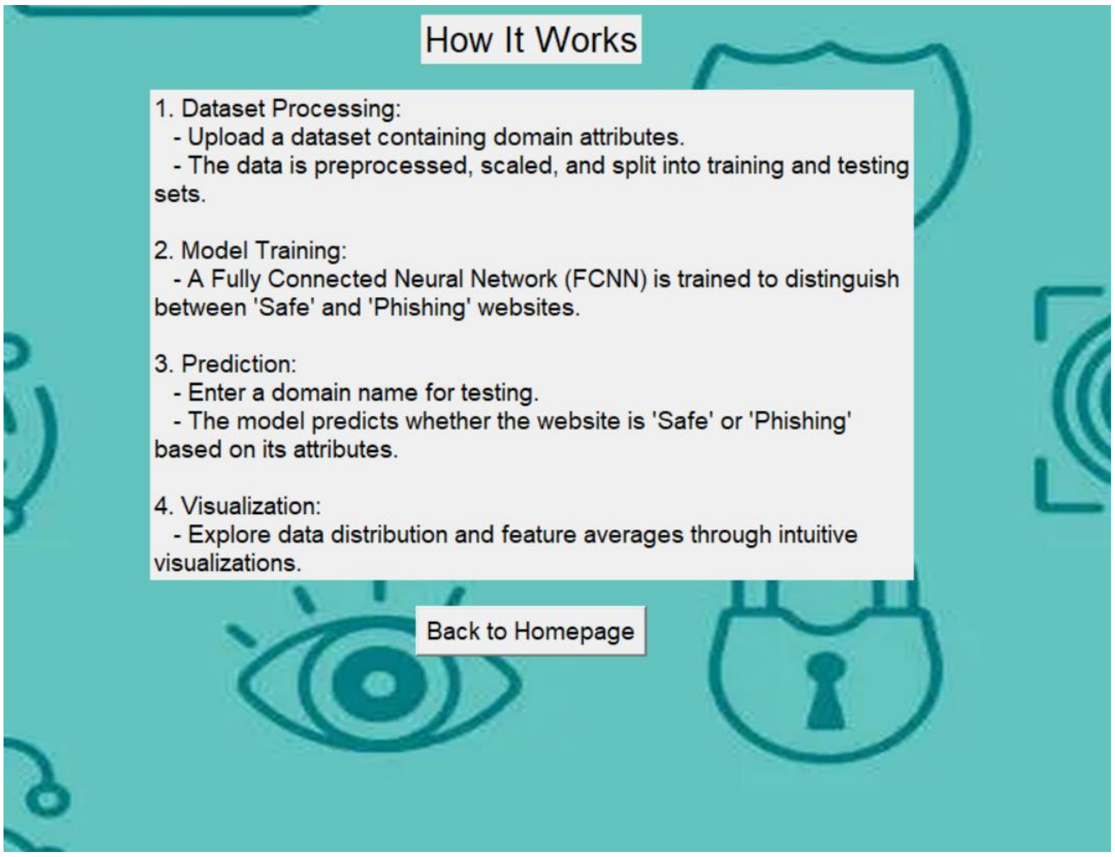




4.8.5 DATA VISUALIZATION



4.8.6 HOW IT WORKS



How It Works

1. Dataset Processing:
   - Upload a dataset containing domain attributes.
   - The data is preprocessed, scaled, and split into training and testing sets.

2. Model Training:
   - A Fully Connected Neural Network (FCNN) is trained to distinguish between 'Safe' and 'Phishing' websites.

3. Prediction:
   - Enter a domain name for testing.
   - The model predicts whether the website is 'Safe' or 'Phishing' based on its attributes.

4. Visualization:
   - Explore data distribution and feature averages through intuitive visualizations.

CHAPTER 5
CONCLUSION

CHAPTER 5
CONCLUSION

5.1 CONCLUSION

The phishing website detection system was successfully implemented using a deep learning approach, primarily leveraging a **Fully Connected Neural Network (FCNN)** due to its superior accuracy. The system is designed to classify websites as **phishing or legitimate** based on extracted URL-based features from a pre-existing dataset.

A user-friendly interface was developed to allow users to upload datasets, preview data, and test domain classifications. The FCNN model, trained on the dataset, demonstrated high accuracy, making it the preferred choice for phishing detection in this implementation.

Although the system does not support real-time detection, it serves as a reliable **dataset-driven classification tool** for analyzing phishing trends and training cybersecurity models. The project highlights the potential of deep learning in improving phishing detection accuracy while providing an intuitive platform for data analysis.

5.2 LIMITATIONS

1. Dataset Dependency:

- The system is entirely reliant on a pre-existing dataset and does not perform real-time phishing detection. Any new phishing techniques that are not included in the dataset cannot be identified by the model.

2. **No Live URL Analysis:**

- The model does not analyze live web traffic or dynamically fetch and evaluate URLs. This limits its applicability in real-world scenarios where phishing websites constantly evolve.

3. **Limited Feature Scope:**

- The detection is solely based on **URL-based attributes**, ignoring other important indicators like webpage content, metadata, or user behavior. This may lead to misclassification in cases where phishing attempts use deceptive webpage elements.

4. **False Positives and False Negatives:**

- Since the model is trained on a static dataset, it may incorrectly classify some legitimate websites as phishing (**false positives**) and vice versa (**false negatives**). Continuous dataset updates and fine-tuning are required to improve accuracy.

5. Lack of Adaptive Learning:

- The system does not have a self-updating mechanism to learn from new phishing trends. Without frequent retraining on updated datasets, its effectiveness may decrease over time.

6. Computational Requirements:

- Although FCNN was chosen for its accuracy, deep learning models require significant computational resources. Training the model on large datasets may demand powerful hardware, making real-time or large-scale implementation challenging.

5.3 FUTURE SCOPE

1. Defining scams during the time:

- Improvement of transmission may include direct url analysis by integrating web and URL scanning tools based on API. This will allow users to detect scam websites in real time instead of relying on static data sets.

2. The hybrid approach:

- Learn carefully about the traditional methods based on Heuristic and based on rules that can improve the accuracy of findings. For example, the integration of FCNN techniques into natural language treatment techniques (NLP) to analyze the web content can provide more strong scam detection.

3. Adaptive learning system:

- The imposition of automatic update models recover periodically using the newly detected scam URL can improve performance. A response can allow users to report new fraudulent efforts, enriching data sets.

4 Future versions can analyze additional indicators such as

- Details of the reputation of the school
- Details of SSL
- detected can be deployed as a service form Web or API, allowing businesses, companies to extend the network security and browser to integrate it into their safety frames to improve protection.

4. Integrated with the web browser and e-mail safety:

- The transmission deployment may include browser extensions or e-mail safety tools automatically report fraudulent links in Real time, providing user warning immediately.

5. Analysis and graphic report:

- An improved visual control panel can be given to follow fraudulent trends, display statistics to detect and provide network security information for organizations.

CHAPTER 6
REFERENCES

CHAPTER 6
REFERENCES

6.1 REFERENCES

1.Alkawaz, M. H., Steven, S. J., & Hajamydeen, A. I. (2020, February 28–29). Detecting phishing website using machine learning. 16th IEEE International Colloquium on Signal Processing & Its Applications (CSPA 2020).

2.Mishra, A., & Gupta, B. B. (2014). Hybrid solution to detect and filter zero-day phishing attacks. ERCICA.

3.Kiruthiga, R., & Akila, D. (2019). Phishing websites detection using machine learning. International Journal of Recent Technology and Engineering (IJRTE), 8(2S11).

4.Jain, A. K., & Gupta, B. B. (2018). A machine learning-based approach for phishing detection using hyperlinks information. Springer Nature.

5.Mahajan, R. (2018). Phishing website detection using machine learning algorithms.

6.Pujara, P., & Chaudhari, M. B. (2018). Phishing website detection using machine learning: A review.

7.Dobolyi, D. G., & Abbasi, A. (2016). PhishMonger: A free and open-source public archive of real-world phishing websites.

8.Satish, S., & Babu, S. K. (2013). Phishing websites detection based on web source code and URL in the webpage.

9.Yi, P. (2018). Web phishing detection using a deep learning framework.

10.Al Saedi, M., & Flayh, N. A. Phishing website detection using machine learning: A review.

11.Abuzuraiq, A., & Alkasassbeh, M. Review: Phishing detection approaches.

12.Sönmez, Y., Tuncer, T., Gökal, H., & Avci, E. (2018). Phishing websites features classification based on extreme learning machine. Proceedings of the 6th International Symposium on Digital Forensic & Security (ISDFS 2018).

13.El-Rashidy, M. A. (2021). A smart model for web phishing detection based on new proposed feature selection technique. Menoufia Journal of Electronic Engineering Research, 30(1), 97–104.

14.Lokesh, G. H., & BoreGowda, G. (2020). Phishing website detection based on an effective machine learning approach. Journal of Cyber Security Technology.

15.Zara, U., Ayyub, K., Khan, H. U., Daud, A., Alsahfi, T., & Ahmad, S. G. Phishing website detection using deep learning models.

16.Gopal, S. Phishing website detection by machine learning techniques – Dataset [CSV file]. GitHub.

17.Peng, T., Harris, I., & Sawa, Y. (2018). Detecting phishing attacks using natural language processing and machine learning. Proceedings of the IEEE 12th International Conference on Semantic Computing (ICSC), 300–301.

18.Sahingoz, O. K., Buber, E., & Kugu, E. (2024). DEPHIDES: Deep learning-based phishing detection system. IEEE Access, 12, 8052–8070.

19.Wenyin, L., Huang, G., Xiaoyue, L., Min, Z., & Deng, X. (2005, May). Detection of phishing webpages based on visual similarity. Proceedings of the 14th International Conference on World Wide Web.

20.Zhang, Y., Hong, I., & Cranor, F. (2007, May 8–12). CANTINA: A content-based approach to detecting phishing websites. Proceedings of the 16th International Conference on World Wide Web.

21.Mohammad, R. M., Thabtah, F., & McCluskey, L. (2014, August). Predicting phishing websites based on self-structuring neural networks. Neural Computing & Applications, 25(2), 443–458.

22.Buber, E., Diri, B., & Sahingoz, O. K. (2018). NLP-based phishing attack detection from URLs. In Intelligent Systems Design and Applications (Vol. 736, pp. 608–618). Springer.

23.Abu-Nimeh, S., Nappa, D., Wang, X., & Nair, S. (2007). A comparison of machine learning techniques for phishing detection. Proceedings of the Anti-Phishing Working Groups 2nd Annual eCrime Researchers Summit (eCrime 2007), 60–69.

24.Babagoli, L., Aghababa, M. P., & Solouk, V. (2019). Heuristic nonlinear regression strategy for detecting phishing websites. Soft Computing, 23(12), 4315–4327.

25.Butnaru, A., Mylonas, A., & Pitropakis, N. (2021, June). Towards lightweight URL-based phishing detection. Future Internet, 13(6), 154.

St. Francis College for Women

Begumpet, Hyderabad-500016

(Autonomous & Affiliated to Osmania University)

PLAGIARISM CERTIFICATE

1	Roll No.	121323030026
2	Name of the Student	AILNENI BANU HARSHINI
3	Title of the Dissertation	DETECTING PHISHING WEBSITES USING DEEP LEARNING TECHNOLOGIES
4	Name of the Supervisor	Dr. Sr. Sujatha Yeruva
5	Department	M.Sc. Data Science
6	Similar content (%) identified	8 %
7	Acceptable Maximum Limit	UG – 25% PG – 15%
8	Software Used	Turnitin - iThenticate 2.0
9	Date of Verification	27th February 2025

Verified by

[illegible]

Librarian

Librarian
St. Francis' College For Women
Begumpet, Hyderabad-16.

[illegible]

Research Supervisor

[illegible]

Head of the Department

Computer Science Dept
ST FRANCIS COLLEGE FOR WOMEN
Begumpet, Hyderabad - 500 016