PHISHPOT- ML BASED ANTI PHISHING SYSTEM

Project Report

Abstract

The project presents the development of a phishing detection website using Flask and machine learning integration. The Flask application serves as the backbone, providing a user-friendly interface for users to submit URLs and receive real-time predictions. The integration of machine learning models, specifically the Random Forest classifier, enables the accurate detection of potential phishing websites. Through feature extraction, relevant patterns and characteristics are captured from the URLs, and a pre-trained Random Forest model is utilized for prediction. The seamless integration with Flask ensures real-time evaluation, enhancing user experience and website security. The project's success lies in the effective fusion of Flask, machine learning, and HTML templates, showcasing a robust and user-friendly phishing detection solution.

Chapter 1: Introduction - PHISHPOT

PHiSHPOT

CATCH THAT PHISH!!!

1.1 Background and Motivation

PhishPot is about limiting the threats of phishing attacks on frequent internet user. Phishing is a frequent cyberattack in which criminals pose as reliable sources in an effort to dupe victims into disclosing personal information. Machine learning is being investigated as a potential remedy for the shortcomings of conventional phishing detection approaches. A phishing detection website with machine learning can increase accuracy and efficiency by applying automated pattern recognition and data analysis, offering users real-time security.

1.2 Why did we choose ML for this project?

Machine learning offers a solution for traditional methods that fall short in detecting and preventing cyber-attacks such as phishing scams. By including machine-learning algorithms on a phishing detection website it becomes more effective in providing real-time protection for users against these attacks. It's capable of automating procedures, handling vast amounts of data intelligently while adjusting to dynamic fishing techniques without compromising accuracy hence reducing false negatives/positives instances significantly too. Finally, this feature-rich tool also enhances user education on the impact of cyber threats increasing their awareness about how to enhance their online safety.

1.3 Problem Statement

When it comes to digital security threats like phishing attacks few threats loom larger - posing serious risks both financially as well as for data security. Unfortunately however, traditional methods aimed at mitigating such dangers rarely live up to their promise- proving slow moving ineffective against emerging tactics developed by nefarious actors seeking personal gain. To address this problem we're proposing a machine learning based phishing detection website designed to provide an adaptive response that can accurately identify potential scams in real time. In doing so we hope to better protect users from further harm while also educating people about these risks so they can stay safe online moving forward. By leveraging cutting edge technology and real time analysis we believe our solution can offer significant benefits for individuals and organizations alike - providing the peace of mind that comes from knowing threats are being detected and mitigated quickly without any unnecessary false positives or negatives to worry about.

1.4 Phishing????

Phishing attacks are a common type of cyber-attack in which malevolent people pretend to be reliable organizations in order to trick others into disclosing sensitive information or performing acts that damage their security. These attacks take place via a variety of channels, including social media, email, instant messaging, and bogus websites. Phishing mails are deliberately made to look authentic and use psychological tricks to sway users into making snap decisions.

Common Phishing Techniques and Their Evolution:

Phishing techniques have evolved over time as attackers continuously adapt their strategies to bypass security measures and increase the success rate of their attacks.

Below lists some common phishing maneuver performed by malicious actors:

Phishing techniques have undergone changes and advancements over time as attackers continually adapt their strategies to overcome security measures and improve the success rate of their attacks. Here are some commonly used phishing techniques and their evolution:

1. Email Spoofing: Email spoofing involves falsifying the sender's email address to make it appear as if the message is from a legitimate source. Initially, attackers used basic methods like modifying the "From" field. However, as email security improved, attackers adopted more sophisticated techniques, such as mimicking legitimate domains and employing tactics like Display Name Spoofing to create authentic-looking emails.

2. Deceptive URLs: Phishers often employ deceptive URLs to trick users into visiting fraudulent websites that closely resemble genuine ones. In the past, attackers relied on simple tactics like using alternative domain names (e.g., "paypa1.com" instead of "paypal.com"). However, as user awareness increased, attackers evolved their strategies by utilizing more intricate approaches like homograph attacks, which involve substituting visually similar characters to deceive users (e.g., replacing "o" with "0" or "l" with "1").

3. Social Engineering: Social engineering is a tactic that manipulates human psychology to exploit vulnerabilities. Attackers craft messages that trigger urgency, curiosity, or fear to induce individuals to disclose sensitive information or carry out specific actions. Over time, social engineering techniques have become more sophisticated, with attackers leveraging personal information from public sources or data breaches to personalize their messages and enhance credibility.

4. Credential Harvesting: Phishers frequently target login credentials to gain unauthorized access to user accounts. Initially, attackers employed simple methods like embedding login forms within phishing emails or websites. However, as users became more cautious, attackers adapted by utilizing techniques like keylogging, where malware records keystrokes, or form hijacking, which involves injecting malicious code into legitimate web forms to capture user credentials.

5. Malware Distribution: Phishers increasingly utilize phishing emails or malicious attachments to distribute malware, such as keyloggers, ransomware, or spyware. These attacks have evolved to exploit software vulnerabilities or leverage social engineering techniques to persuade users to download and execute malicious files.

6. Voice Phishing (Vishing): Vishing attacks involve phone calls where attackers impersonate legitimate entities to deceive victims into revealing sensitive information. Vishing techniques have progressed to utilize Voice over Internet Protocol (VoIP) technology, making it easier for attackers to manipulate caller IDs and convincingly mimic trusted individuals or organizations.

7. Smishing: Smishing attacks exploit text messages to deceive users into taking actions or divulging sensitive information. Attackers have adapted by employing sophisticated methods, such as crafting messages that appear to originate from reputable sources like banks or popular services, often incorporating urgent requests or time-sensitive information.

8. Targeted Phishing (Spear Phishing): Targeted phishing attacks focus on specific individuals or organizations, tailoring messages to exploit their personal or professional relationships. Attackers gather extensive information from public sources, social media, or previous breaches to create highly personalized and persuasive messages, thereby increasing the likelihood of success.

9. Whaling: Whaling attacks specifically target prominent individuals like CEOs or senior executives. Attackers invest time in researching their targets and utilize social engineering techniques to construct tailored messages that appear genuine, with the objective of obtaining confidential or valuable information.

From the above mentioned techniques one of the most prevalent and deceptive one is manipulating the url and tricking the user to click that url and gain sensitive information.

Chapter 2: Cybersecurity Overview

Machine learning, a branch of artificial intelligence, has become a valuable asset in the field of cybersecurity. It offers the ability to analyze large amounts of data, identify patterns, and make informed decisions based on those patterns. By training machine learning algorithms, cybersecurity professionals can enhance their security measures and proactively defend against evolving cyber threats.

2.1 Role of machine learning in enhancing cybersecurity defenses

In order to strengthen cybersecurity defenses, machine learning is essential and transformational. Its extensive use of cutting-edge algorithms to analyze massive amounts of data and identify patterns has important implications for improving numerous facets of cybersecurity. The detection and classification of various cyber threats, including malware, phishing assaults, network intrusions, and unusual behaviors, are areas where machine learning algorithms thrive. These algorithms can recognise well-known attack signatures and even recognise developing or previously unknown attack strategies by learning from historical data and spotting patterns. In addition to threat detection, machine learning facilitates real-time monitoring and response, enabling the prompt identification and mitigation of potential security breaches. By continuously analyzing incoming data, machine learning algorithms can swiftly detect suspicious activities, unauthorized access attempts, or anomalous behaviors that may go unnoticed by traditional rule-based security systems. This proactive approach ensures that threats are promptly addressed, minimizing potential damage and reducing response time. Machine learning's strength lies in its ability to analyze user behavior patterns, enabling the identification of anomalies that may indicate compromised accounts or unauthorized access attempts. By establishing baseline behaviors for individual users or user groups, machine learning models can effectively detect deviations and raise alerts to prevent potential security breaches. This behavior-based approach adds an additional layer of protection to systems and enhances overall cybersecurity posture. Moreover, machine learning techniques can be leveraged to conduct vulnerability assessments, providing insights into software, system, or network weaknesses. By analyzing data related to past security incidents, known vulnerabilities, or system configurations, machine learning models can identify potential vulnerabilities and offer recommendations to fortify security measures. This proactive approach empowers organizations to address potential weaknesses and reduce the likelihood of successful attacks.

2.2 Challenges and Opportunities in applying ML in Cybersecurity

The use of machine learning in cybersecurity introduces a number of difficulties that may affect the dependability and efficiency of machine learning models in this area. These difficulties are connected to several steps in the integration process. Adversarial attacks are one of the major problems since they can seriously impair the accuracy of machine learning models because hackers can change input data to trick the algorithms. Moreover, due to privacy issues, restricted access to data on actual attacks, and the management of sensitive information, the quality and availability of data used for training can be difficult. Inaccurate predictions and poor performance might result from inadequate or biased training data. Machine learning models can operate as "black boxes," making it difficult to comprehend the decision-making process. This presents another issue in terms of the interpretability and explainability of the models. The inability to respond to threats efficiently might be hampered by this lack of openness, which can affect confidence. Scalability and performance are also important issues because machine learning models need to handle and analyze a lot of data in real-time in order to properly detect and respond to threats. Accuracy must be maintained while achieving great performance. Additionally, the constantly changing cybersecurity environment necessitates constant machine learning model adaptation to deal with new threats. Regulations and ethical issues, such as data protection laws, privacy laws, and the ethical use of personal data, must also be taken into account. It is essential to recognize that machine learning models should complement human expertise, not replace it. Collaboration between machine learning experts and cybersecurity professionals is crucial to interpret model outputs and provide domain knowledge for improved accuracy and effectiveness.

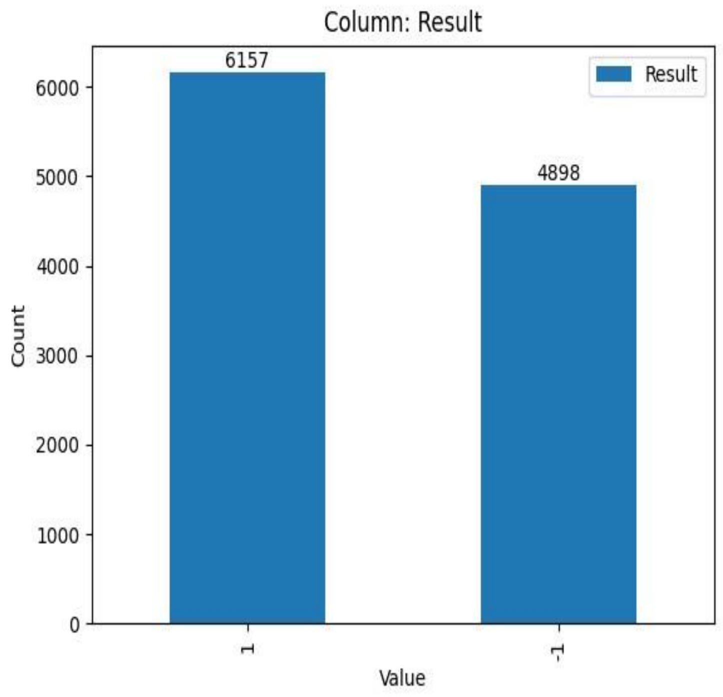
Addressing these challenges requires ongoing research and collaboration to develop robust and reliable machine learning models that can effectively enhance cybersecurity defenses.

2.3 Role of Awareness in combating cyber-attacks

Enhancing user awareness and providing education are essential in the fight against phishing attacks.Phishing attempts frequently involve social engineering approaches to persuade people to divulge private information or carry out hazardous actions. Organizations can enable consumers to spot and report fraudulent emails, messages, or websites by teaching them about the telltale indications and traits of phishing attempts. Training programmes can provide advice on safe email usage, secure web browsing, and the value of double-checking requests for personal information. A proactive defense is established by increased user awareness, which lowers the likelihood of falling victim to trickery. Users are kept up to date on changing attack strategies through ongoing education, allowing them to be aware and take the necessary precautions to protect their data as well as their own. Ultimately, user awareness and education act as a crucial defense line against phishing attacks, supplementing technical safeguards and fostering a security-conscious environment.

Chapter 3: Methodology

3.1 Dataset used:



Total Number of Phishing URLs : 4898
Total Number of Legitimate URLs : 6157

3.2 Overview of Features

We make a user-defined function named "prepare_url". This function will ensure that all URLs of the dataset are in a consistent format before we do feature extraction on that. A regular expression is used to determine if the URL begins with either "http://" or "https://". String concatenation is used to add "http://" to the beginning of the URL if it is not preceded by either of these protocols. To process the URL further, the URL must be accessible via the HTTP protocol. This function is a useful utility function that ensures all URLs are formatted in the same way and can be processed efficiently.

Our user define function named get_soup is created to scrape websites by retrieving their HTML content and parsing it into a BeautifulSoup object. The function makes an HTTP GET request to the supplied URL using the well-known requests library, and then parses the HTML data using the BeautifulSoup constructor. Any issues that may arise during the request, such as a network error or a timeout, are handled by the try and except blocks. The function returns value '-999' for soup in the event of an error, signifying that the request was unsuccessful. If not, the object soup, which contains the HTML content that was parsed, is returned.

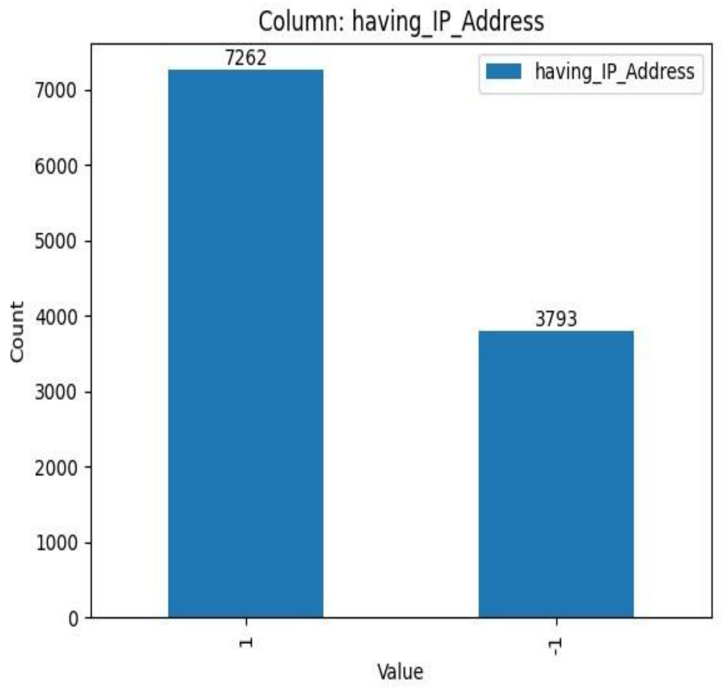
Next, we developed the "find_domain()" function, which pulls the domain name from a URL using a regular expression and the re module. The regular expression specifically looks for a substring that contains the characters "://" and "/", capturing the string in between. The domain name of the URL is represented by this text.

This function uses another regular expression to determine whether the domain name has a period (".") after it has been extracted. If so, the function cuts the period from the domain name's beginning.Function then produces an updated domain name, so we can use for further processing of features extracting.

The function named "whois_response()", which obtain the WHOIS data for a specified domain from the whois library. The response is returned as a dictionary object after the function performs a query to the proper WHOIS server for the domain. This dictionary includes a range of domain-related details, such as the registrar's information, the domain's creation date and date of expiration, and contact details of the domain.Depending on the our specific needs of the application, the returned dictionary object may be used for further feature extracting.

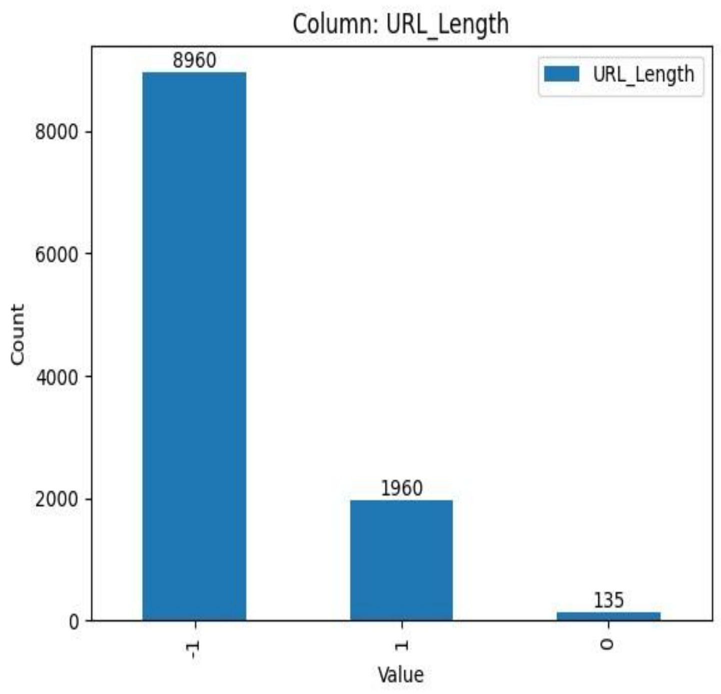
The function named "find_global_rank()" method sends a POST request with the provided domain as a parameter to the checkpagerank.net website using the requests library. This website offers a tool to look up a domain's global rank. The function then extracts the global rank from the HTML responses received from the checkpagerank.net website using a regular expression. It specifically looks for the pattern "Global Rank:" followed by one or more digits and it gets the digits as an integer number. The function returns a value of -1 in the case that the regular expression fails to identify a valid global rank. In the last function gives the global rank as an integer. This value can be used to get the relative popularity of the specified domain in comparison to other domains.

having_ip_address():

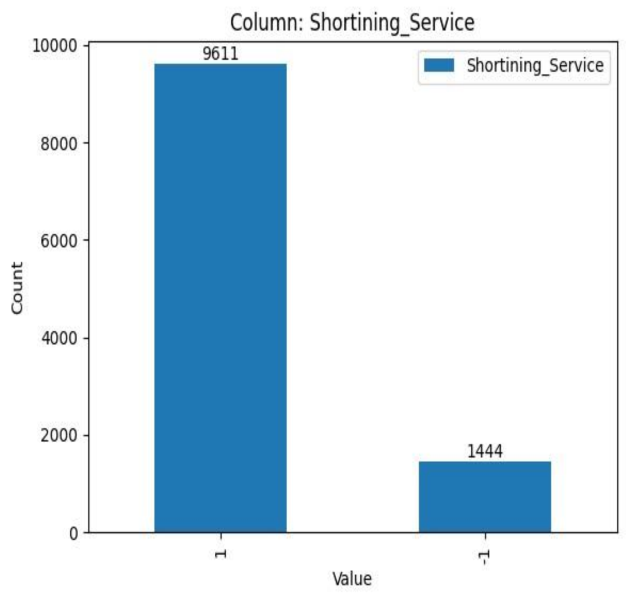


To extract our very first feature we define a function named "having_ip_address()" , which scans a URL as input to see if it has an IP address using the ipaddress library. The function returns -1 if an IP address has been found, suggesting that the URL is probably a phishing website. The function returns 1, suggesting that the URL is probably a legitimate website if no IP address has been found.
The feature was developed in response to the identifying that phishing websites frequently substitute IP addresses for domain names in their URLs in order to get over DNS-based security mechanisms. On the other hand, legitimate websites generally incorporate domain names inside their URLs. As a result, this functionality can be utilized to help differentiate between authentic and fraudulent websites.

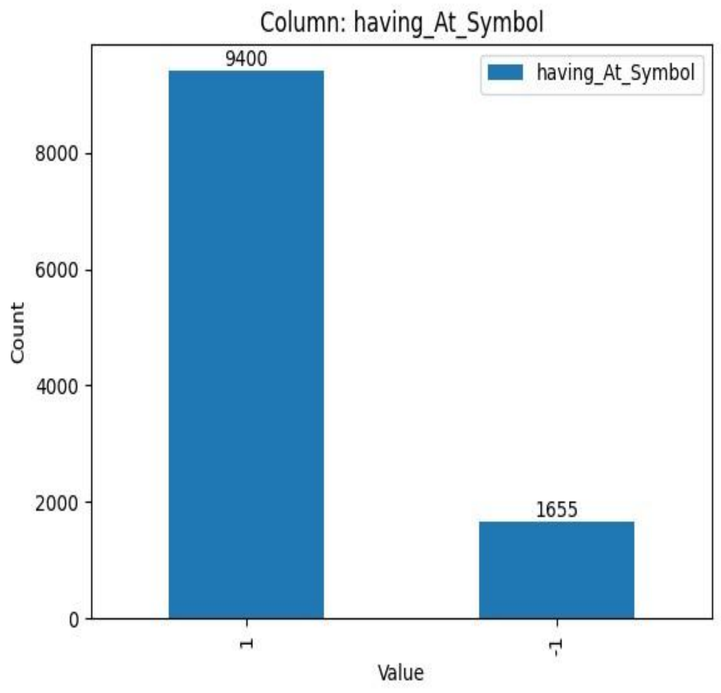
url_length():



To extract our second feature, URLs are sent to the function named "url_length()", which calculates its length. The function returns a value of 1, indicating that the URL is probably valid, if the length is lower than 54 characters. The function returns a value of 0 if the URL's length ranges from 54 to 75 characters, suggesting that the length alone cannot tell if the URL is authentic or not. The function returns a value of -1 if the URL length exceeds 75 characters, suggesting the likelihood that the URL is fraudulent or phishing-related. This is so that attackers can mask their true intents and fool people into clicking on them by frequently using lengthy and complicated URLs. **shortening_services():**



To extract our third feature, "shortening_services()" defined function uses a regular expression to match against popular URL shortening service domains to determine whether a URL was created by a URL shortening service or not. This function returns -1, which denotes a possibly suspicious URL, if there is a match. And this function returns 1, suggesting that there is probably no match and that the URL is not a shortened URL.
This function's goal is to identify and warn about potentially dangerous URLs that could be used for phishing or other malicious actions. It can be challenging for visitors to understand where they will be redirected because URL shortening services are frequently used to hide the true destination of a link. **having_at_symbol():**

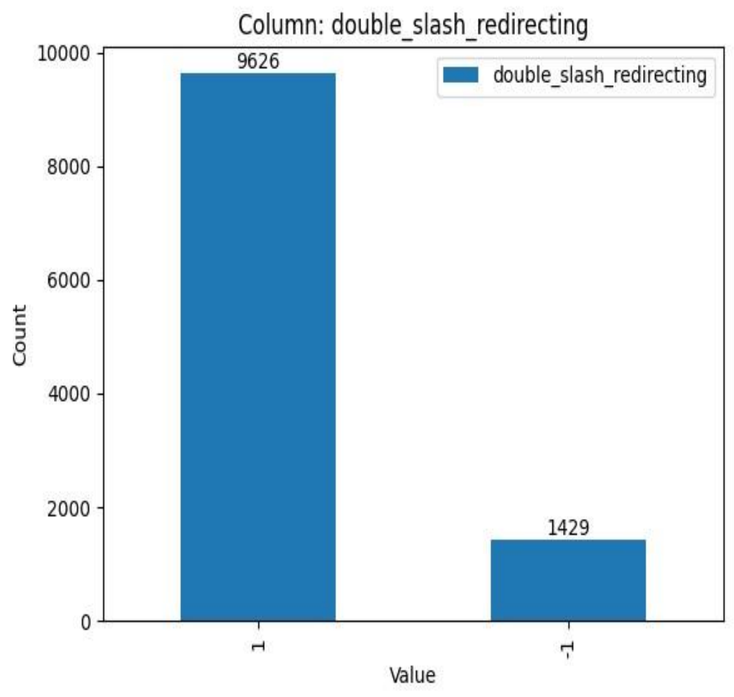


To extract our next feature we defined function named "having_at_symbol()" which detects whether the "@" symbol appears in the provided URL. The function returns a value of -1 if such a symbol is

present in the URL, showing that the URL may be suspect or potentially dangerous. The "@" symbol is frequently used in email addresses, and its use in a URL can mean that sensitive information, such as login details or private data, is being recorded there.
Additionally, the function returns a value of 1, indicating that the URL is less likely to be malicious or suspicious, if it does not contain the "@" character.
This function's goal is to find and indicate potentially dangerous URLs that could be exploited for

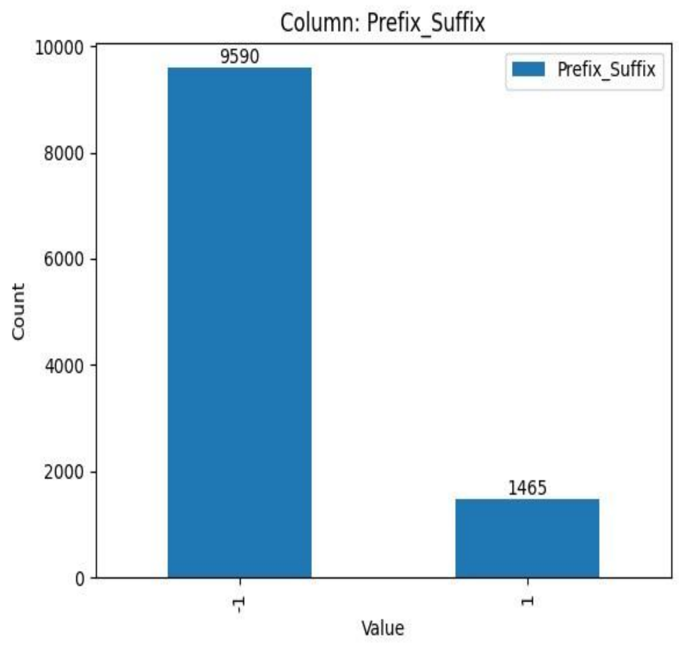
phishing or other malicious activities. by identifying and highlighting URLs with the "@" mark.

double_slash_redirecting():



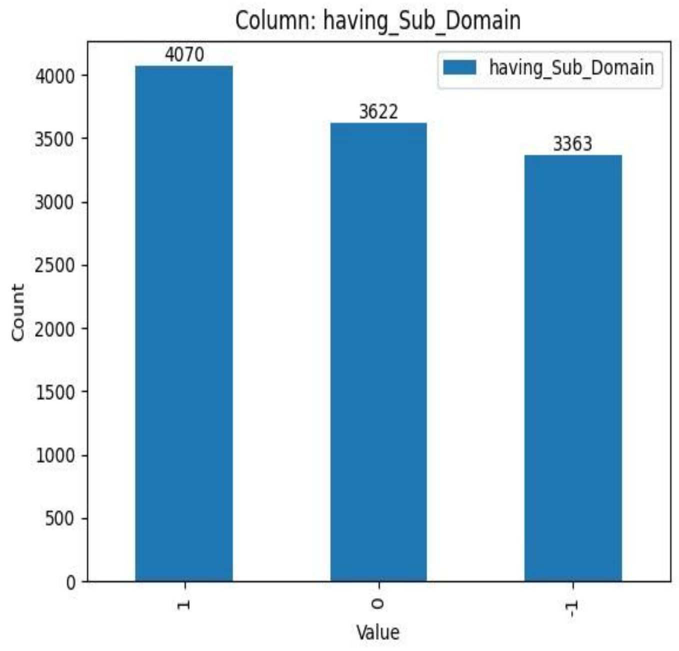
To extract our fifth feature we defined a function named "double_slash_redirecting()", which scans for double forward slashes "//" in the URL. The function locates the last instance of these double forward slashes in the URL if they are observed. The function gives a value of -1, indicating that the URL may be exploited for phishing or other malicious purposes, if the index of the last occurrence is more than 6, indicating that the double forward slashes occur after the domain name. It can be risky for attackers using double forward slashes to drive users towards a different website. The function gives a value of 1, signaling that the URL is probably secure, if the position of the final double forward slash is not greater than or equal to 6. This is explained by the fact that valid URLs often have double forward slashes placed prior to the domain name.

prefix_suffix():



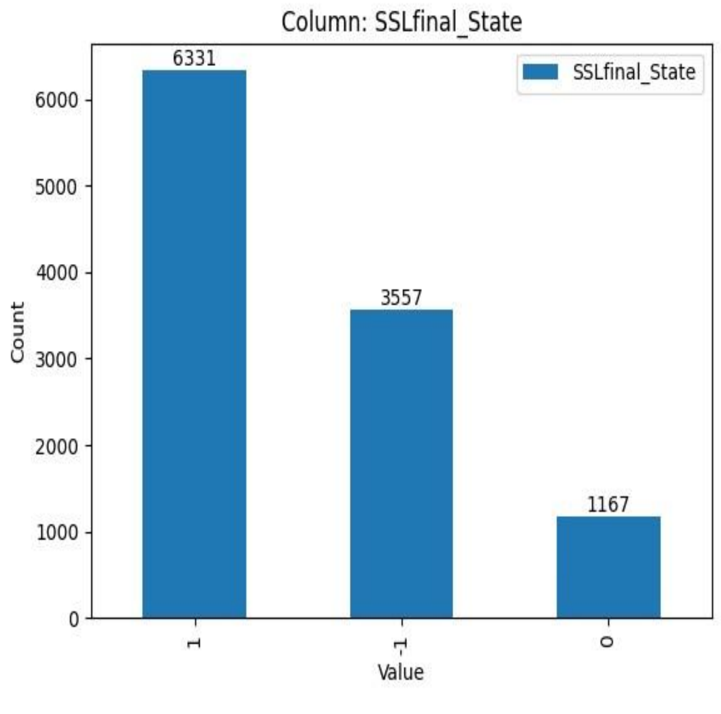
To find out whether a specific URL includes a prefix and suffix separated by a hyphen, we created a function named "prefix_suffix()". This is done by matching URLs that start with "http://" or "https://" and have a minimum one hyphen between two non-hyphen letters using a regular expression. The function gives a value of -1, showing that the URL can be potentially harmful if it meets this pattern. Attackers can create phony websites that seem like real ones by combining prefixes and suffixes in URLs, misleading people into providing sensitive information.
In contrast, the function returns a value of 1 in the case where the given URL does not have a prefix and suffix that are separated by a hyphen, suggesting that the URL is less probable to be suspicious.

having_sub_domain():



To extract our next feature we defined function named "having_sub_domain()" ,which determines whether or not a specified URL has a subdomain. Before running the check, the function takes out the IP address from the URL if it is present. This is so that the function's correctness can be impacted by the presence of an IP address, which can be used as a subdomain.
The function first recognizes all of the dots in the URL before running the check. After that, a threshold value is used to compare the number of dots. The function gives a value of 1, suggesting that the URL is more inclined to be malicious or suspicious if it only has three dots or fewer. This is due to the fact that URLs with fewer dots are frequently linked to trustworthy websites that avoid using subdomains.
The function returns a value of 0 if the URL contains four dots. This suggests that further investigation may be necessary as the URL might or might not be suspicious or malicious. The function gives a value of -1 if the URL has more than four dots, suggesting that the URL could be suspicious or potentially harmful. This is due to the greater probability of subdomains, which could be utilized for phishing or other harmful activities, in URLs with more dots.
The function searches the URL for IP addresses using regular expressions. Before completing the subdomain check, it checks against IPv4 as well as IPv6 addresses and deletes them from the original URL.

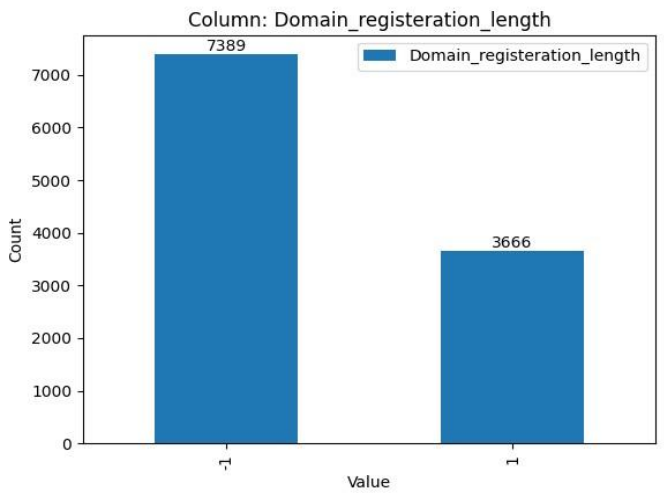
ssl_final_state():



To extract our 8th feature we defined a function named "ssl_final_state()", which verifies the SSL(Secure Sockets Layer) certification of a particular website response. The function receives the website's response and measures it for length. The function returns a value of 1, suggesting that the SSL certificate of the website is authentic and that it is less likely to be suspicious if the length of the answer is more than 50.

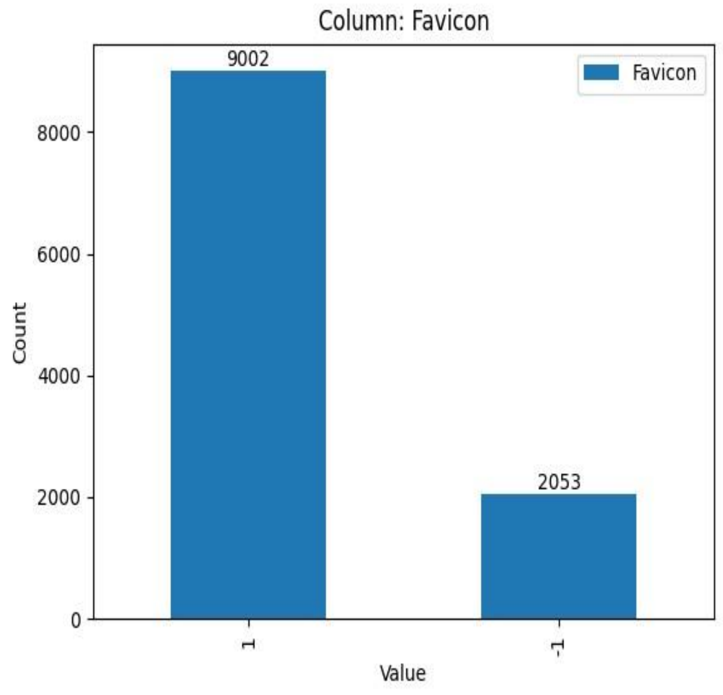
The exchange of information between a website and its users is protected by SSL certificates. They are used to create an encrypted link between the user's internet browser and the website's server, guaranteeing the privacy and security of any data sent between them. A crucial aspect of evaluating the legitimacy of a website is its use of a safe connection, which is indicated by a valid SSL certificate. This function gives a value of -1 if the length of the answer is 50 or less, suggesting that the SSL certificate of the website may be missing or improperly set. Because of its possible vulnerability to assaults that violate the security of user data, this could indicate that the website might be harmful.

domain_reg_length():



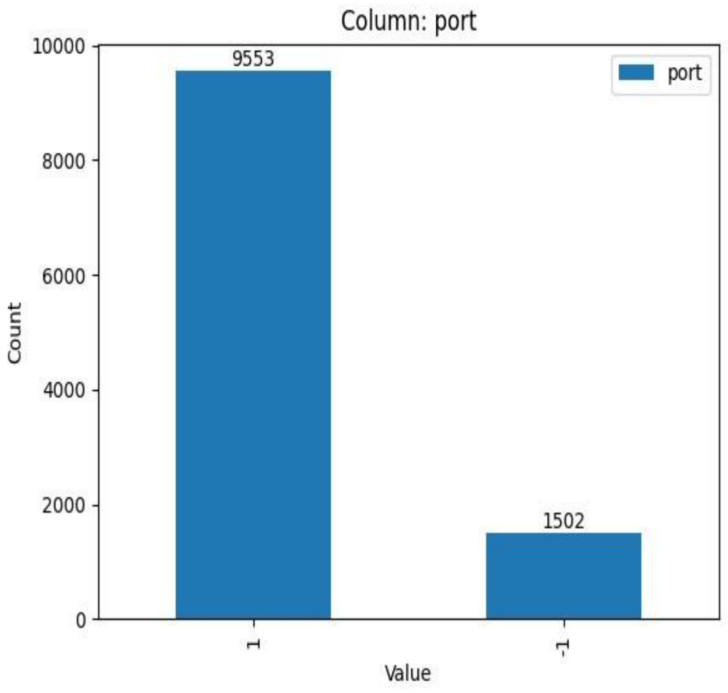
To extract our next feature we defined function named "domain_reg_length()", which examines how long the domain registration will be active after receiving a WHOIS response as input. The WHOIS protocol is used to access databases that contain data on registered domains.
The procedure begins by obtaining the domain's expiration date from the WHOIS answer. The variable "expiration_date" contains this date. The "str" function is then used to turn the expiration date into a string, and the "time.strftime" function is applied to get the current date. The date is formatted as a string in the form 'YYYY-MM-DD' using the "strftime" function.
The "split" function is used to divide the expiration date into a list of strings, from which the first element is then extracted using indexing. This element is saved in a variable named "exp_date" and contains dates in the format "YYYY-MM-DD".By dividing the result by 365, the variation in the year of the expiration date and the present year is determined. This provides a rough estimate of the remaining days before the domain registration terminates.
This function returns a value of -1 if this value is lower than 365, signifying that the domain registration is going to expire or has already done so. Because of the fact that trustworthy websites typically renew their domain registration before it expires, this may be a sign that the website is potentially harmful or suspicious.
The function gives a value of 1, signifying that the domain registration is still active for at least one more year, if the value is higher than or equal to 365. Given that trustworthy websites often maintain their domain registration for a long time, this suggests that the website is more unlikely to be suspicious or harmful. The function gives a value of -1 as a default value if it is unable to obtain the date of expiration from the WHOIS response.

favicon():

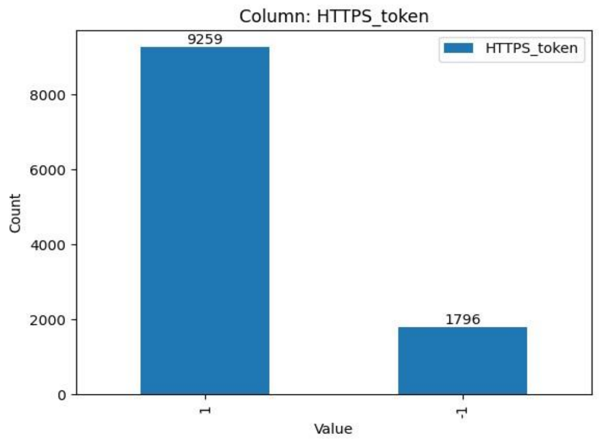


To extract our tenth feature we passed three parameters to the function named "favicon()" : the "soup" object, a BeautifulSoup object which represents the HTML content of a website; the "url" parameter, a string that represents the URL of the website being examined; and the "domain" parameter, a string that indicates the domain of the website under examination. The purpose of the function is to determine if the website's favicon is legitimate.
A favicon is a little icon or symbol that can be found in the address bar or tab of an internet browser. It serves in recognising a website and is often seen in the browser alongside to the website's name. Without a favicon, a website could come across as informal or unreliable, which might have an impact on the website's credibility.
This "soup" object is first compared to the value '-999' by our function. The function gives a value of -1 if the website fails to have a valid favicon, suggesting that it is true. When an error or an incorrect URL prevents the "soup" object from being produced, this value is normally returned. Then this function uses the "find_all" method of the BeautifulSoup object to look for a "link" element in the "head" part of the HTML code if the "soup" object is not equal to -999. Dots, which are a sign of a file extension, are looked for in the "href" property of the "link" element. The file is taken to be a favicon if there is only one dot in the "href" property.
Then this function then determines whether or not the "href" property contains the "url" argument or the "domain" parameter. after that our function gives a value of 1, suggesting that the website has a legitimate favicon when any of these values is present. The function gives a value of -1, signalling that the website lacks a legitimate favicon, when neither value is provided.

port():

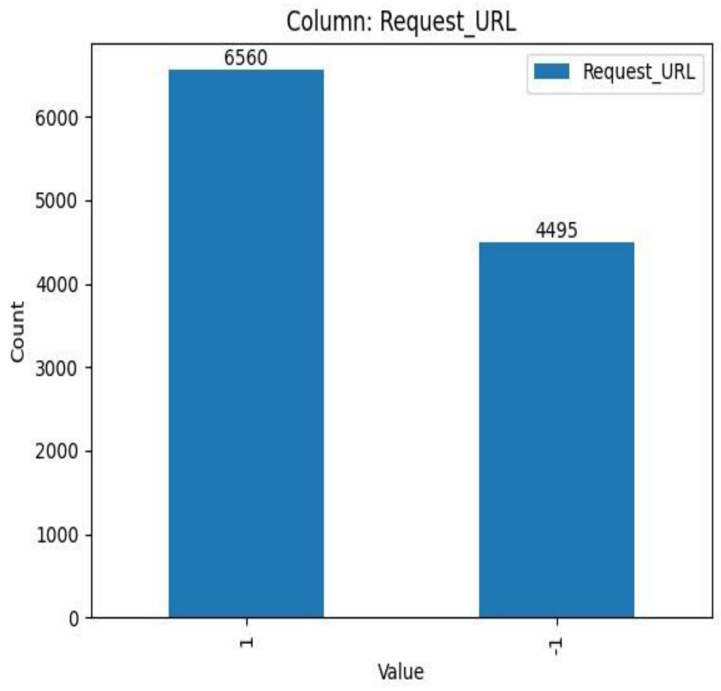


To extract our next feature we defined a function named "port()" , which accepts the domain name as input and determines whether the domain string contains a port number. The colon (":") delimiter, which separate the domain name and port number in an URLs, is used in the function's initial attempt to split the domain string. If the port number is contained in the domain string, it should be the second element of the resultant list. This value is stored in a variable called "port".
If the domain string contains a port number, our function gives a value of -1, suggesting that the domain is probably a subdomain or may be a potentially dangerous. As ports are often used to describe network services on a server rather than to identify a website, respectable websites are typically less likely to have a port number in their domain names.
The function gives a value of 1, suggesting that the domain is probably a genuine website with a standard URL syntax, if there is no port number included in the domain string.
This Function gives a default value of 1 if it detects an issue or fails to separate the domain string applying a colon delimiter. **https_token():**



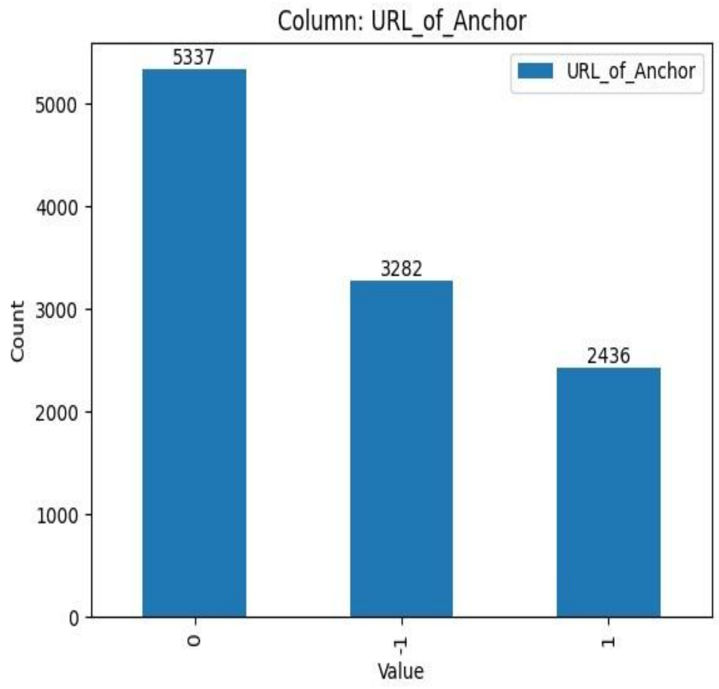
For our next feature extraction When a URL is given, the "https_token()" function determines whether the URL begins with "https://" or not. The function looks for a pattern that matching the start of the given string using a regular expression. The method gives a value of 1 if the pattern matches "https://", suggesting that the URL utilizes the HTTPS protocol, a secure protocol for sending data over the internet.
Any sensitive information communicated is kept private and secure because of the HTTPS protocol, which encrypts data sent within a user's web browser and the website's server. The HTTPS protocol transfers data encrypted, making it more reliable and secure than the HTTP protocol and which sends data in plaintext and is open to assaults that jeopardize the security of user information. This function gives a value of -1, suggesting that the website might be utilizing an insecure protocol, if the input URL does not begin with "https://".

request_url():



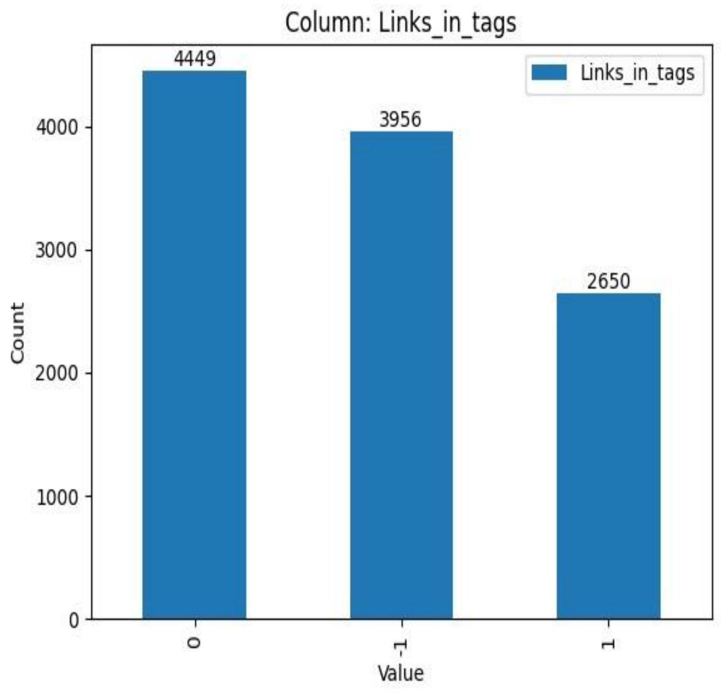
To extract the next feature we gave three parameters as inputs 'url','soup', and 'domain' to our defined function "request_url()". This function checks the quantity of requests to external resources, such as photos, audio files, iframes, and embedded objects, and extracts data from the HTML content of the provided web page using BeautifulSoup.
The function returns '-1' if the soup parameter is set to '-999'. If the specified web page's HTML content is present without any errors this function moves on to use BeautifulSoup to check for requests for external resources.
This function examines each resource request in the HTML code to see if the URL or domain name of the requested web page is included in the external resource's source code or if the external resource only has one domain level (that is, one dot in the URL). The function increases the 'success' counter if the URL, domain name, or external resource has a single domain level. Every time an external resource is found in the HTML text, the function also increases a variable named "i." The function determines the proportion of successful requests—that is, requests for which the URL or domain name of the specified web page is present or the external resource has a single domain level—after all

external resources have been evaluated. The number of successful requests is divided by the total number of requests, and the resulting number is multiplied by 100 to calculate the percentage.
If less than 22% of queries are successful, the function gives 1, suggesting that the provided website is probably secure. The function gives 0 if the range of successful requests is between 22% and 61%, suggesting that the safety of the web page is questionable. Finally, the function gives -1, indicating that the provided web page may not be secure, if the percentage of successful requests is above or equal to 61%. **url_of_anchor():**



Three parameters—the "url," "soup," and "domain"—are provided to our defined function "url_of_anchor" . By examining the proportion of safe hyperlinks, this function can determine whether or not a URL contains a risky hyperlink. Based on the percentage of safe links, the function gives one of three values: 1, 0, or -1.
The function returns -1 if the value of "soup" is -999, which denotes that the webpage is not accessible. Otherwise, the function uses the find_all() method to analyze the "a" tags found in the
HTML content. Then function examines each "a" element to see if the hyperlink has a "#" or "javascript" or "mailto" attribute in its href attribute. Additionally, it verifies that the hyperlink doesn't include the primary domain or URL. Any of these circumstances makes the hyperlink unsafe, and the "unsafe" counter is raised as a result.
In the end,This function then multiplies the result by 100 and divides the "unsafe" counter by the total number of "a" tags to determine the proportion of unsafe hyperlinks. Then the URL is evaluated to see if it includes a malicious hyperlink based on this percentage number. The function gives 1, denoting that the URL is secure if the percentage is below 31 The function gives 0 if the calculated percentage falls between 31 and 67, suggesting that the URL may not be secure. The function gives -1, signaling that the URL is unsafe, if the percentage is more than or equal to 67.

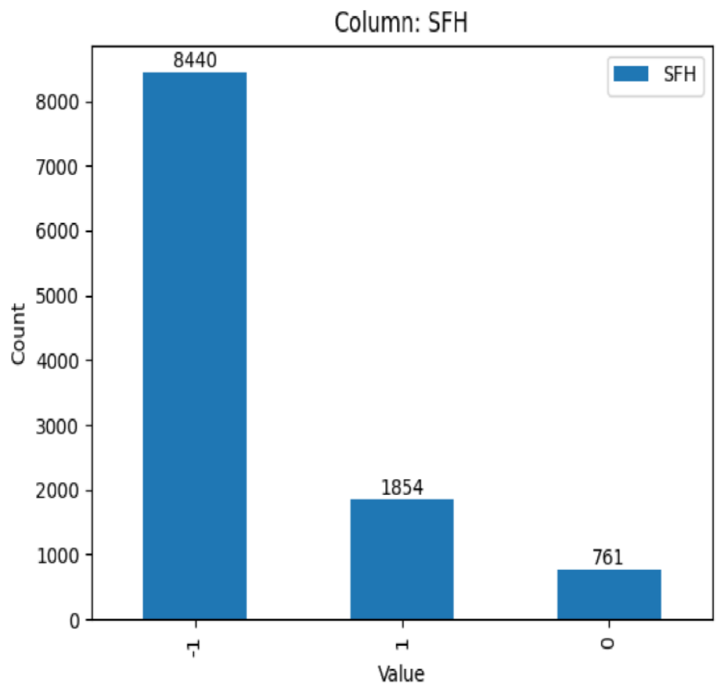
links_in_tags():



To assess whether the quantity of external links present in HTML tags like link and script is normal, abnormal, or severely abnormal, we defined a function named "links_in_tags()". The URL of the webpage to be examined, the BeautifulSoup object produced from the webpage, and the domain name of the webpage are the three input parameters required.

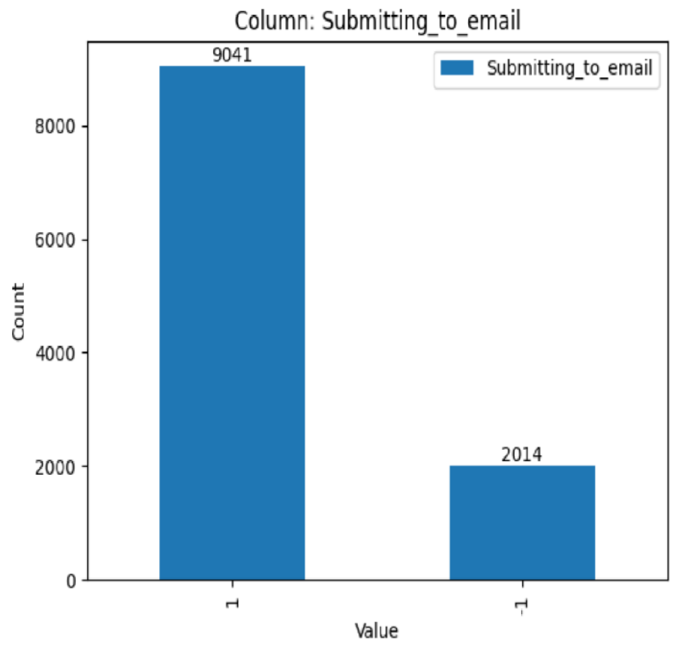
'i' and 'success' are two counts that the function first sets to zero. Then it determines whether the soup parameter is equal to -999, which would mean BeautifulSoup was unable to interpret the website. The function gives -1 in this situation, signifying an error.
If the webpage was successfully parsed, the function uses the find_all() method of the BeautifulSoup object to look for all instances of the 'link' and'script' tags in the HTML. It determines if the URL or domain name of the webpage is contained in the 'href' or 'src' property of each of these tags. Additionally, it looks to see if the 'href' or 'src' property only contains one dot, which might signify a relative path. The function increases the 'success' counter if any of these criteria are met.
Following the completion of all 'link' and 'script' tag processing, the function determines the proportion of successful matches by dividing the 'success' counter by the 'i' counter and multiplying the result by 100. The function gives 1, signifying a secure webpage, if the 'i' counter has a value of zero meaning no 'link' or 'script' tags were discovered.
A value of 1, 0, or -1 is given back by the function depending on the determined percentage. The function gives 1, suggesting that the webpage is secure, if the percentage is lower than 17%. The function returns 0 if the percentage falls between 17% and 81%, suggesting a dubious website. A phishing webpage is indicated by the function's returned value of -1 if the percentage is higher than 81%.

sfh():

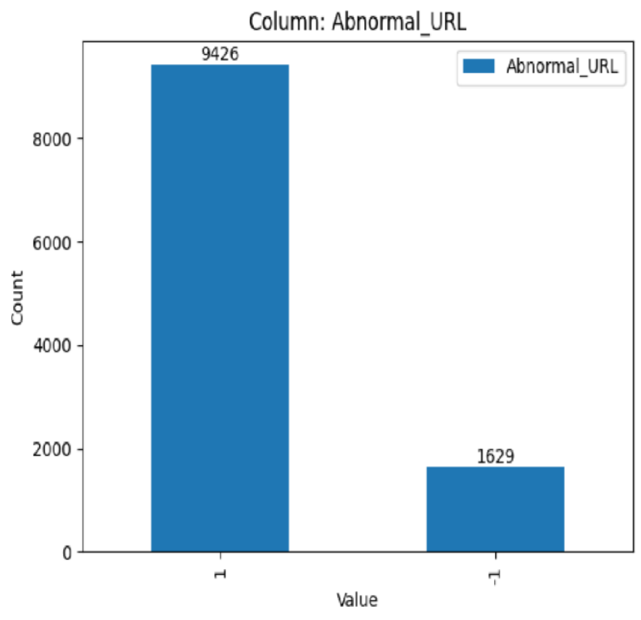


'url','soup', and 'domain' are the three parameters that the 'sfh' function takes as inputs. This function is used to examine a web page's "Submit Form to Host" (SFH) vulnerability.
The 'soup' parameter is initially checked to see if it equals -999 by this function. If it is, this function gives -1, suggesting that it is impossible to execute the SFH vulnerability check. If not, the function uses the 'find_all' method of the 'soup' object to look for every 'form' tag in the parsed HTML. The function determines whether the 'action' property for each 'form' element is "about:blank" or an empty string. If it is, the function gives a result of 1, meaning that the form submission has not been properly specified.
This function evaluates whether the 'url' or 'domain' parameters can be found in the 'action' attribute if it is not empty or "about:blank". The function gives 0, suggesting that the form submission may be susceptible to SFH, if either of them is absent.
The function gives 1, showing that the form submission is immune to the SFH vulnerability if the 'url' and 'domain' arguments are both present in the value of the 'action' attribute. Finally, this function gives 1 if none of the conditions are met.

submitting_to_email():

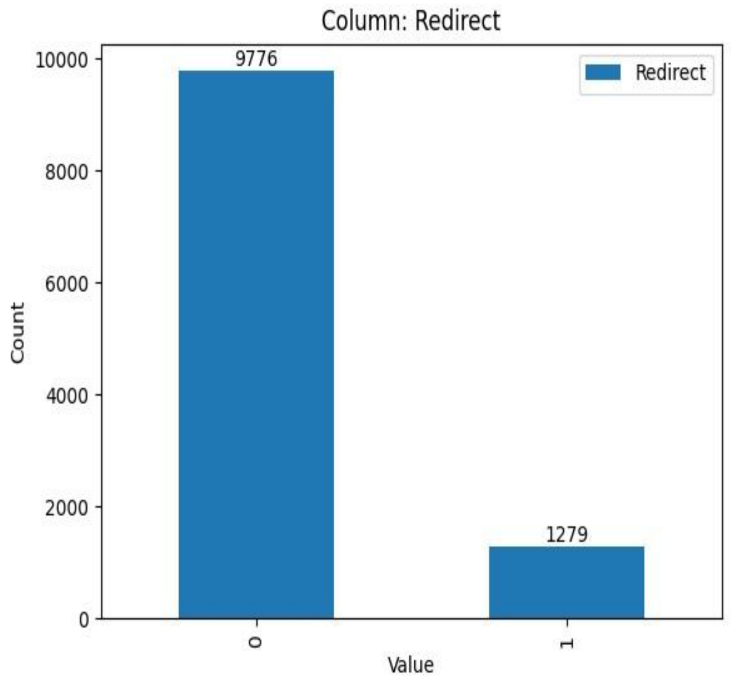


To extract our next feature we defined a function named 'submitting_to_email()' which only accepts a single argument, 'soup,' which is the BeautifulSoup object created by reading a webpage's HTML. This function checks whether an online form is sending information to an email address or not.
The 'soup' object is initially checked to see if it equals -999 by the function. If so, the function gives -1 to denote an error and shows that the HTML content could not be parsed. Otherwise, the function uses the 'find_all' method of the 'soup' object to loop over all 'form' tags in the HTML content. This function examines the value of the 'action' attribute for each 'form' tag. and then the function gives -1 to signal a possible phishing attempt if the result includes the string "mailto:," which indicates that the form is sending data to an email address. If none of the conditions are met, the method returns 1 to show that the form is not sending information to an email address.
then the function gives 1 to show that there is no risk of sending data to an email address if no 'form' tags are present in the HTML content. **abnormal_url():**



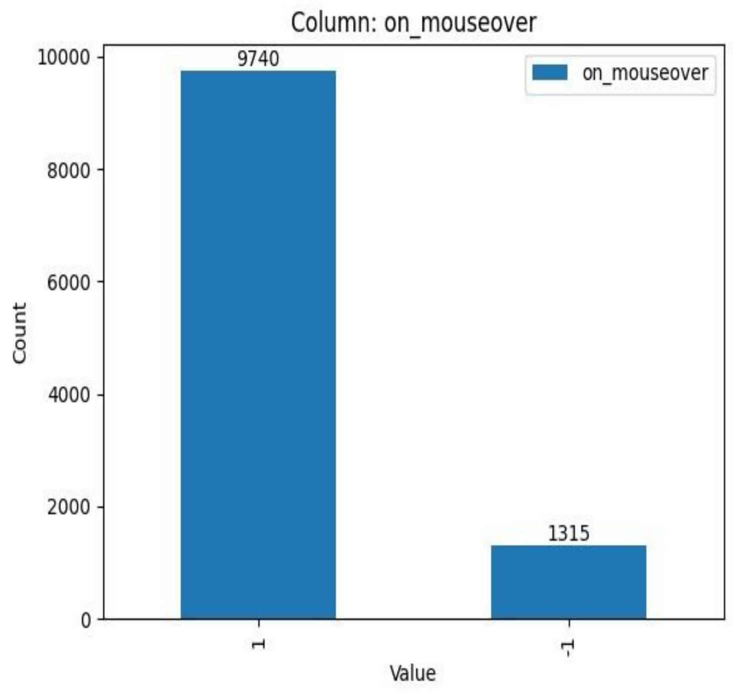
Domain and url are the two parameters that the "abnormal_url()" function accepts as inputs then the function looks for the domain string in the url string using the re module. If a match is discovered, it gives 1, meaning the url is valid. If not, it returns -1, meaning the URL is strange.
In simple terms, the function determines whether or not the URL is a part of the domain. If it does, the URL is accepted as normal; if it doesn't, it is seen as abnormal. The feature is helpful in identifying phishing attempts that use spoof URLs, or URLs that look to be genuine but are not. The function can determine whether a URL is real or fraudulent through comparing the domain in the URL to the real domain of the website.

redirect():

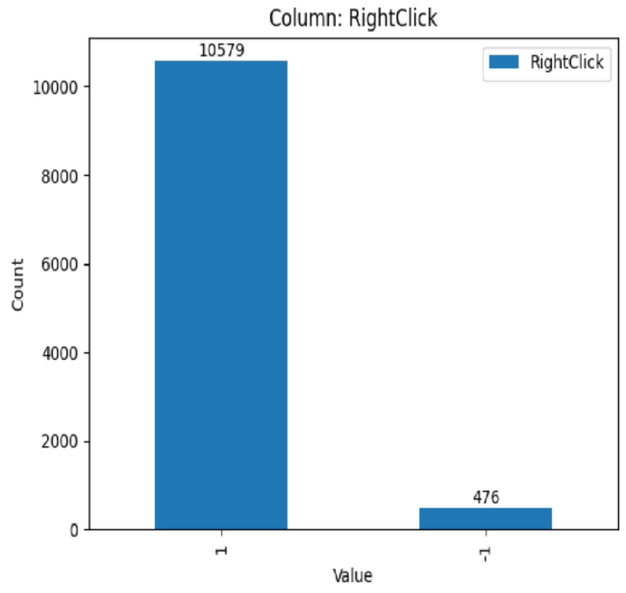


'redirect()' is a function that accepts a URL as an input and gives an integer value depending on whether the URL is a fake website or not. The function counts how many redirects there were between entering the URL and receiving the final result. The function produces an integer value that represents the safety of the URL depending on the number of redirections, with -1 denoting safe, 0 denoting potentially risky, and 1 denoting unsafe.
The 'requests' library is used by the function to first send a GET request to the given URL. The function gives -1, denoting an error, if the response is empty. If a response is received, the function determines how many redirections took place before receiving the whole response.
Then this function gives -1, showing that there has been no redirection and the URL is secure if the number of redirections is 0 or equal to 1. This function gives 0, which means the URL could represent a fake website if there are between 2 and 4 redirections. Lastly, the function gives 1 which suggests that the URL is a fake one, if the number of redirections is greater than 4.

on_mouse_over():



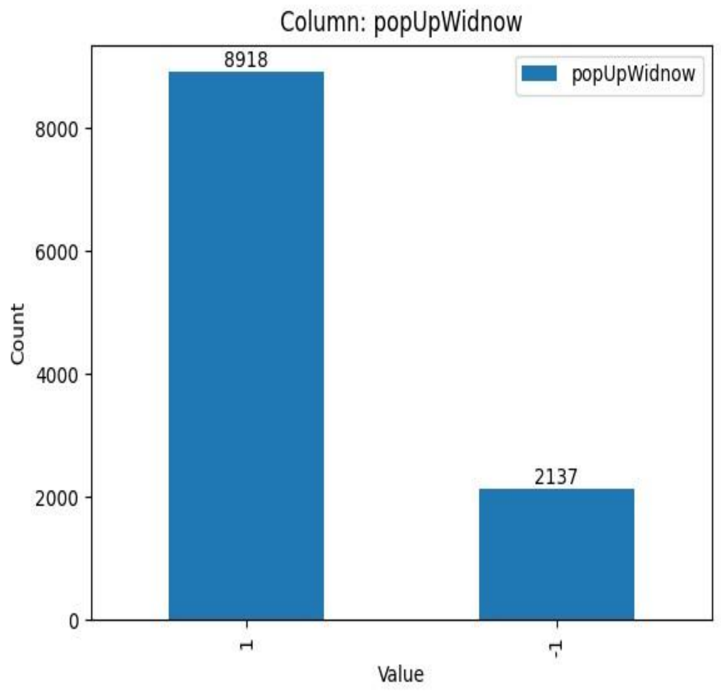
To extract this feature we created a function named "on_mouse_over()",which accepts a URL as an input argument and determines whether the webpage linked to the URL has any onmouseover event listeners. The function sends a GET request to the URL using the requests module, and parses the response's HTML using BeautifulSoup.
Then function gives -1 if the request meets an error or the answer is empty. Otherwise, it employs a regular expression pattern to look for any script element in the response content that has the onmouseover property. Then function gives -1 if such a tag is discovered, suggesting that the URL may not be secure. The function gives 1, suggesting that the URL is secure to access, if no onmouseover event is found. **right_click():**



The 'right_click()' function accepts a URL as an input parameter and, if right-clicking disabled is detected in the page source code, gives -1; otherwise it gives 1. It sends a GET request to the provided URL using the 'requests' library, and parses the HTML result using the 'BeautifulSoup' library.

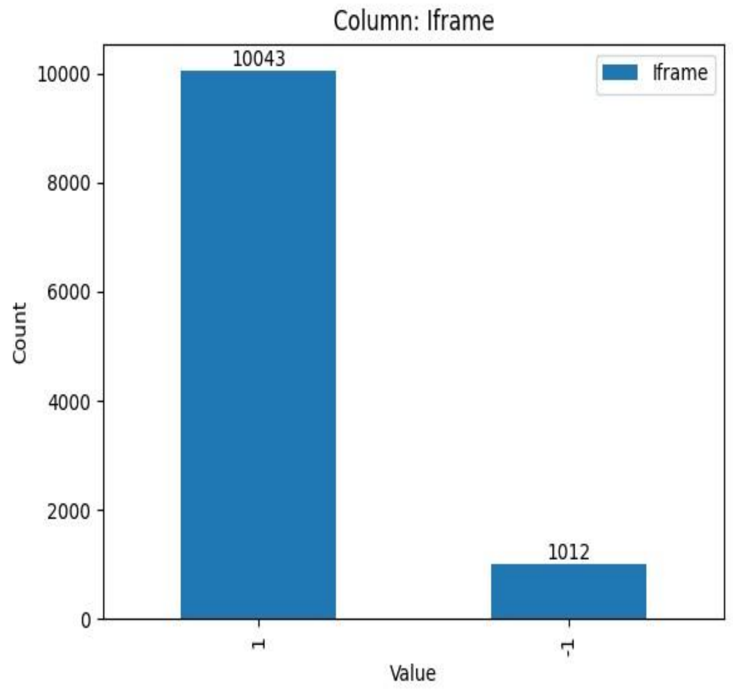
The function initially determines whether or not the response is empty. It returns -1 if the response is null, indicating that the URL is unsafe. The function checks to see if someone has right-clicked on the page if the response is not empty.
Using the 're.findall()' technique, it looks for instances of the regular expression 'event.button == 2' in the page's source code. The function returns -1 if this regular expression is discovered, indicating that right-clicking disable is possible on the page. If not, it gives 1, meaning that the URL is secure. As a safety measure, the function gives -1 if an exception arises while it is being executed.

popup_window():



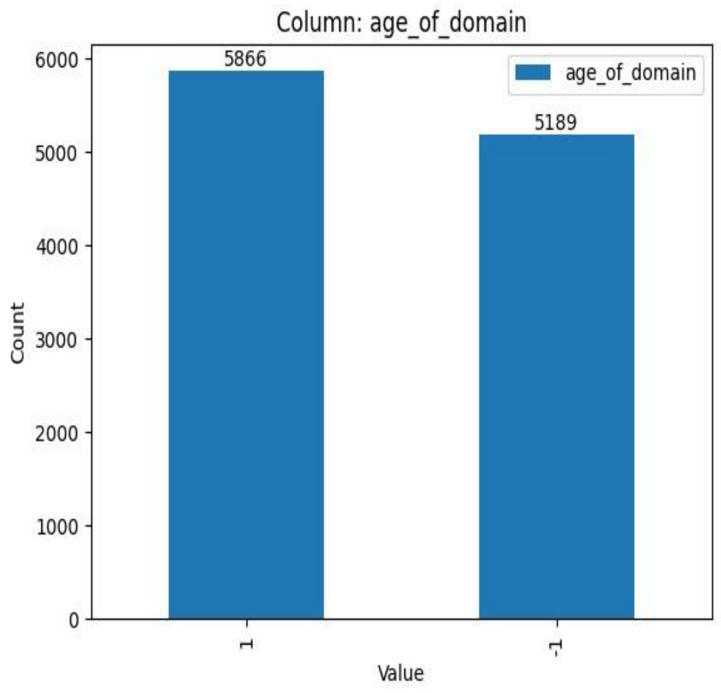
To extract our next feature we defined a function named 'popup_window()' , which accepts a 'url' as input and gives one of the integer values '-1', '0', or '1'.
This function first attempts to send a GET request to the specified 'url' using the 'requests' module. If the response is not null, it then uses the 'BeautifulSoup' module to parse the page's HTML code. The function then use a regular expression to determine whether the string "prompt(" is present in the page's HTML content. Then this function gives '-1' if the string is discovered, which suggests that the page trying to open a popup window for user input. Otherwise, it gives '1', indicating that no popup window is being attempted to be opened by the page. The function gives '-1' if any issues are encountered when processing the GET request or parsing the HTML content.

iframe():

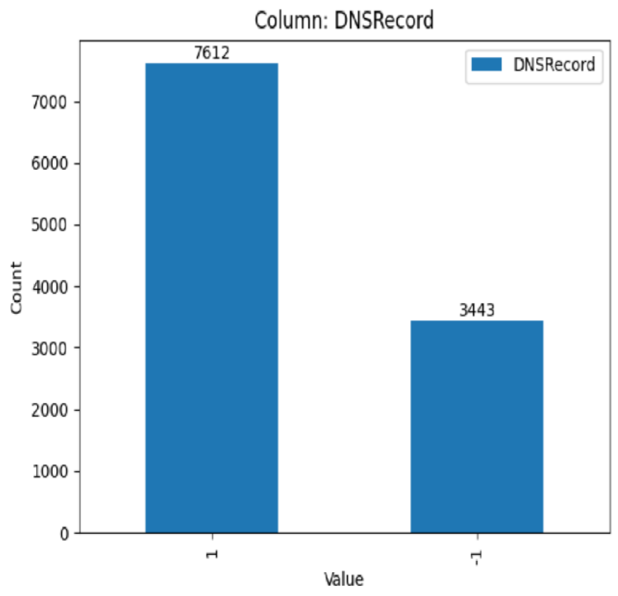


The 'iframe()' function examines a URL to see if it has any 'iframe>' or 'frameBorder>' tags in the HTML code. Then it gives 1, which denotes the existence of an iframe, if it does. If an iframe is not present on the webpage, then function gives -1, meaning it is not there. The function also returns -1 if the URL cannot be reached or if no HTML code for the webpage can be located.
The function sends a GET request to the provided URL and retrieves its HTML code using the 'requests' library. The HTML code is then parsed by the "BeautifulSoup" package, which employs a regular expression to look for any "iframe>" or "frameBorder>" tags.
The function gives 1 indicating the existence of an iframe if the regular expression matches any 'iframe>' or 'frameBorder>' tags in the HTML code. In the event that an iframe is not present, the function returns -1.
This 'iframe()' function can be useful for spotting potential security risks, such as the use of iframes to load malicious data from other sites or to insert destructive code into a website. It can also be used to verify adhering to security regulations that prohibit the usage of iframes on websites.

age_of_domain():



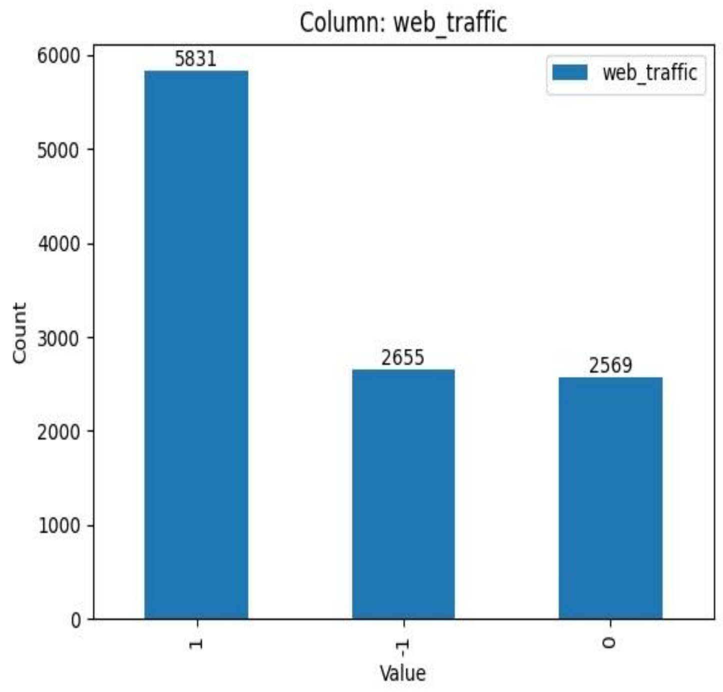
The whois response of a domain is the input for our defined function called "age_of_domain()", which, depending on the domain's age, gives an integer value of -1, 0, or 1.The whois response's domain registration date is first extracted by the programme. The difference between the current year and the year of registration is then used to compute how many years have gone by since the date of registration. The function gives -1 if the calculated value is below or equal to 1, which denotes that the domain is relatively fresh. Then function gives 1, showing that the domain is comparatively old, if the calculated value is more than 1. If a problem arises while the function is running, it returns -1, meaning that it was unable to determine the domain's age. **dns_record():**



'dns_record()' is a function that accepts a domain parameter as input and gives a value of 1 or -1.The DNS variable is first set to -1 by the function. Then it makes an effort to collect domain information using the whois module. The DNS variable is set to 1 if the whois call is successful. DNS stays at -1 if the whois call is unsuccessful.

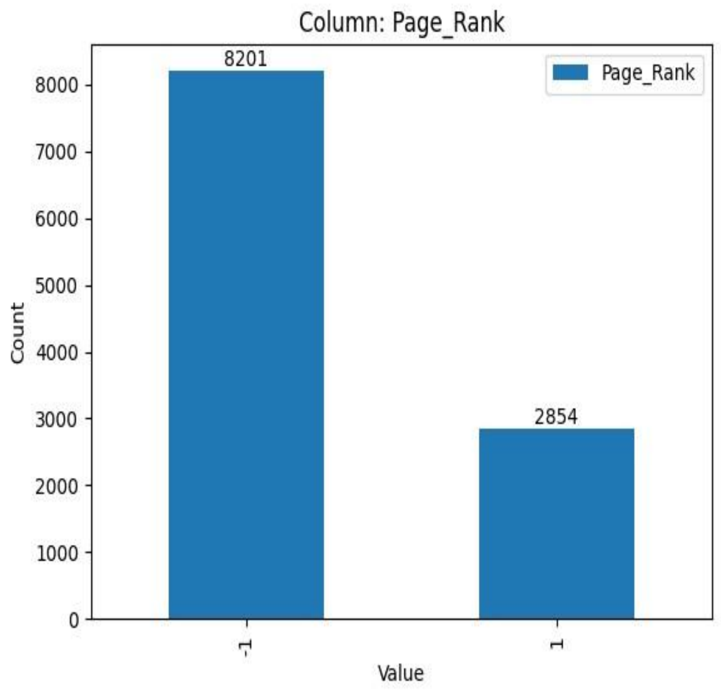
The function examines DNS to see whether it equals -1, and if it does, it returns -1, meaning the domain does not have a valid DNS record. Then function gives 1, showing that the domain has a valid DNS record, if DNS is not equal to -1.

web_traffic():



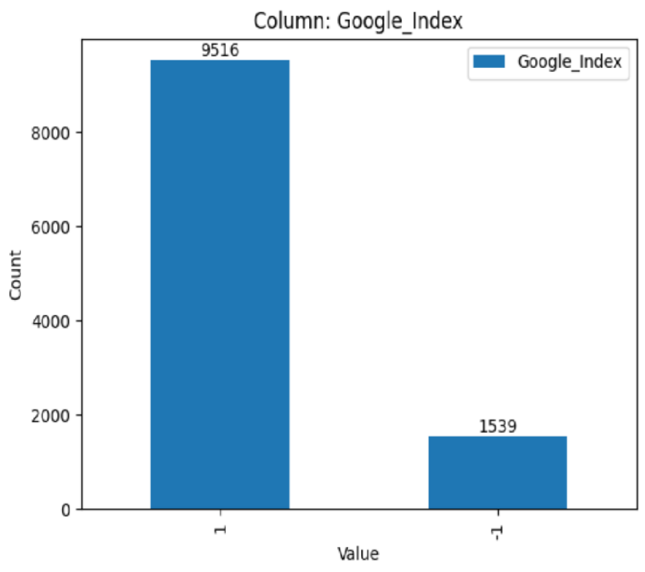
'web_traffic()' is a function that accepts a URL as input and gives a score showing how much web traffic the given website receives.
This function begins by retrieving the XML of the website's traffic statistics using the 'urllib' library. The 'REACH' element is then retrieved from the parsed XML data using the 'BeautifulSoup' package. Based on online traffic statistics, the website's global rank is displayed in the 'REACH' section.
If this function gives a score of 1, showing that the website has substantial traffic and global rank of the website is less than 100,000. If function gives a score of 0, which suggests that there is little traffic on the website, if the global rank of the website is more than or equal to 100,000.
The function gives a score of -1 in the case that there was a problem retrieving or parsing the XML data, suggesting that the web traffic level was not possible to determine.

page_rank():



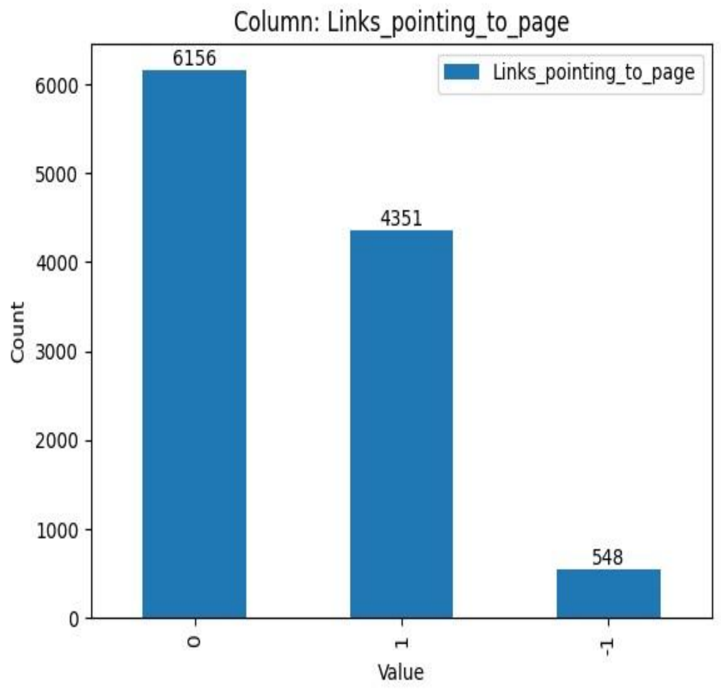
The page_rank() function accepts a website's overall rank as an input parameter and outputs a score that reflects the website's popularity or importance based on that rank.
This function starts by examining if the global rank is greater than 0 and less than 10,000. This range indicates websites that are more highly ranked and have a larger web presence and significance. If the function gives a score of 1, indicating that the website has a high page rank, if the global rank is within this range.
The function gives a score of -1, showing that the website has a low page rank or is not in the list of top-ranked websites, if the global rank is beyond the range of 0 to 10,000.
The function gives a score of -1 in the case where an error occurred while the code was being executed, such as an exception being raised, signifying that the page rank could not be established with accuracy.

google_index():



The 'google_index()' function determines whether the Google search engine has indexed a given URL.

The function accepts a URL as input and executes a Google search for the URL using the search() function. The top 5 results for the URL are returned by the search query.
The search results from the Google search are stored on the variable site. If the site variable has data, it indicates that Google has indexed the URL and included it in the search results. Then our function in this instance gives a value of 1, signifying that the URL has been indexed.
The URL is either not indexed by Google or fails to show up in the top 5 search results if the site variable is null. A result of -1, which denotes that the URL is not indexed, is returned by the function in this situation.
This function helps us to rapidly determine whether or not a particular URL is indexed by the search engine operated by Google. **links_pointing_to_page():**



The 'links_pointing_to_page()' function is designed to look at a URL and count the number of links pointing to that specific page.
When a URL is provided as an input argument, this method starts the process of submitting a request to retrieve the web page's content from the requests.get() method. The received respond is then processed using the BeautifulSoup library, producing an object called soup in the
BeautifulSoup collection.
The code looks for instances of the a href= tag in the response text using regular expressions. The re.findall() function is used to count the matches, and the result is kept in the number_of_links variable.
The code applies specific conditions based on the number of links to determine the level of suspicion. There are relatively few links, indicating a reduced degree of suspicion, if the total amount of number_of_links is equal to or less than 45. The function gives 1 in these circumstances.
It implies a moderate number of links for counts between 46 and 60 , indicating a need for caution. A return result of 0 from the function is used to indicate a significant level of suspicion.
If the number_of_links count is greater than 60, there are likely to be a lot of links, which could be a sign of spam or other nefarious activities. The function returns -1 to signal a high level of suspicion.

The function gives -1 as a way to indicate an error or an unknown status in cases where an exception happens during the analysis, such as running into problems downloading the webpage's content or dealing with network-related problems.

generate_dataset():

After extracting all 30 features we created one more function named 'generate_dataset()', which creates a dataset of features for a given URLs. The dataset is created by extracting various features related to the URL and its associated domain.
The collected features are initially stored in a list called "data_set."

Summary:

1. having_ip_address(): Verifies whether an IP address appears in the URL.
2. url_length(): Measures the URL's length.
3. shortening_services(): Indicates the usage of URL shortening services.
4. having_at_symbol(): verifies to see if the URL contains the "@" symbol.
5. double_slash_redirecting(): Verifies whether the URL contains a double slash redirection.
6. prefix_suffix(): Examines whether the URL contains a prefix or a suffix.
7. having_sub_domain(): Finds out whether the URL has a subdomain.
8. ssl_final_state(): Evaluates the SSL certificate validity based on the WHOIS answer.
9. domain_reg_length(): Determines the duration of the domain registration.
10. favicon(): Examines whether a favicon is present for the website.
11. port(): Identifies the port that the domain uses.
12. https_token(): Verifies whether the URL contains a "https" token.

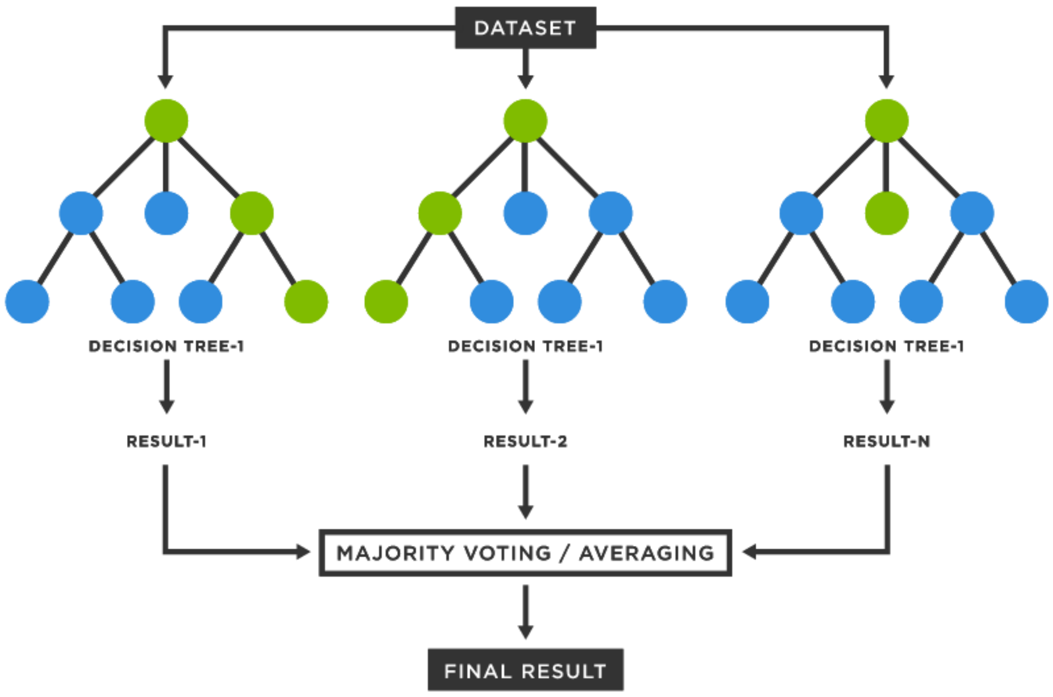
13 request_url(): examines the volume of requests made to the URL.

14. url_of_anchor(): This function establishes how many anchor tags are present in the HTML text.
15. links_in_tags(): function examines the number of links that are contained within HTML tags.
16. sfh(): Evaluates the use of the "Server Form Handler" on the webpage.
17. submitting_to_email(): Verifies whether email submission forms are present on the website.
18. "abnormal_url()": This function uses established patterns to identify abnormal URLs.
19. 'redirect()': function counts the number of redirection a user experiences when accessing a URL.
20. 'on_mouse_over()': checks to see if the webpage has JavaScript code for mouseover events.
21. 'right_click()': This function finds JavaScript code related to the right-click event.
22. 'popup_window()' checks whether pop-up windows are used by JavaScript on the website.
23. 'iframe()': checks to see whether there are any iframes or frame borders on the webpage.
24. 'age_of_domain()': determines the domain's age using the WHOIS response as a basis.
25. The 'dns_record()': function determines whether the domain's DNS records are accessible.
26. 'web_traffic()': analyses web traffic based on the website's ranking.
27. 'page_rank()': evaluates the domain's page rank based on its global rank.
28. 'google_index()': Checks to see if Google has indexed the URL.
29. 'links_pointing_to_page()': Determines how many of links pointing to the website.

30. 'statistical_report()': Conducts a statistical analysis of the URL and domain to look for suspicious characteristics.

3.3 Machine Learning Model:

3.3.1 Random Forest & Its Working:



A random forest model is a machine learning algorithm used for both classification and regression tasks. It is an ensemble learning method that combines multiple decision trees to make predictions. In a random forest, each tree is trained on a random subset of the training data and features. During prediction, the results of all trees are aggregated to produce the final output. This technique helps to reduce overfitting and improve the accuracy and robustness of the model.

- Our code reads a CSV file, 'PhisingWebsite_datset.csv', using the pandas library.
- Function named "prepare_data" is defined to prepare the data for training and testing.
- The "prepare_data" function separates the target variable ('Result') from the input features and creates training and testing sets by splitting the data.
- The code applies the "prepare_data" function to the loaded data and assigns the resulting values to corresponding variables.
- A random forest classifier model with 50 estimators is created using the RandomForestClassifier class.
- The model is trained on the provided training data using the 'fit' method. The trained model is used to predict the target variable for the testing data, and the predictions are stored in a variable called 'y_pred'.

- The code calculates the accuracy of the random forest model by comparing the predicted values with the actual values from the testing data using the 'accuracy_score' function.
- The resulting accuracy score is saved in the variable 'rf_acc'.
- The trained random forest model is serialized and saved as a pickle file named 'rfmodel.pkl' for future use.

Above was the summary of our code performing following tasks:

- Data Loading
- Data Preparation,
- Model Training,
- Accuracy Evaluation
- Saving a model in a pickle file

Chapter 4: Website Development & Integration

4.1 Overview of the website

PhishPot is a website developed with the primary objective of safeguarding users against phishing attacks. Phishing attacks are more common than ever in the digital world, posing serious hazards to people, companies, and organizations everywhere. PhishPot intends to address this rising issue by offering reliable detection tools and equipping users with the information necessary to recognise and avoid phishing attacks.

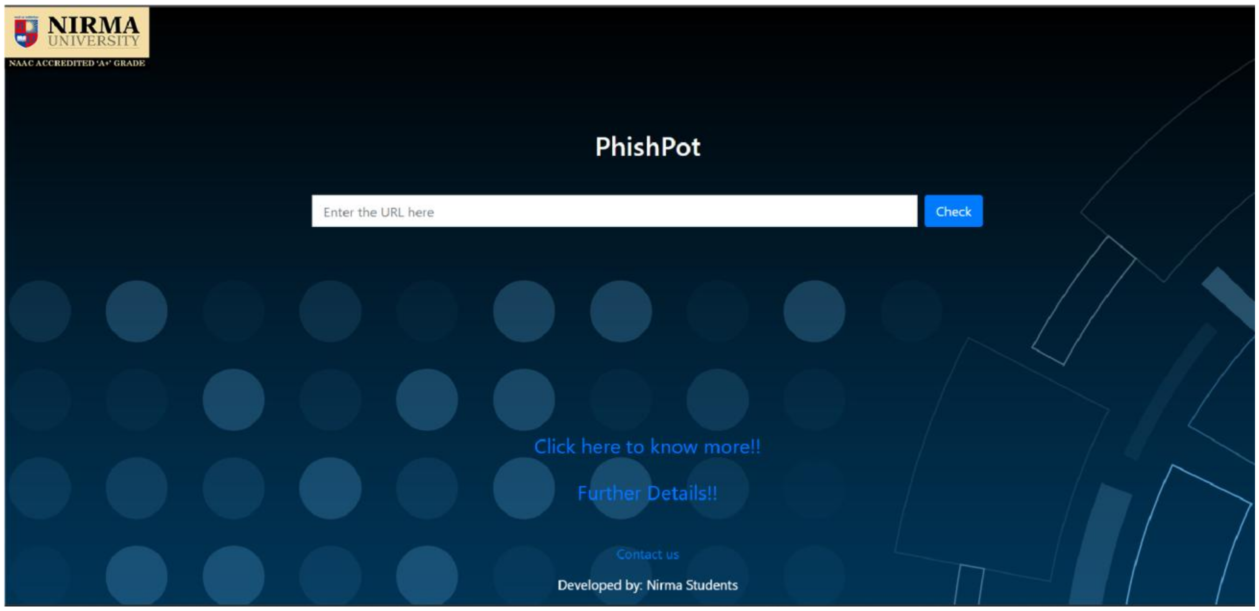
4.2 Website Features & Functionality

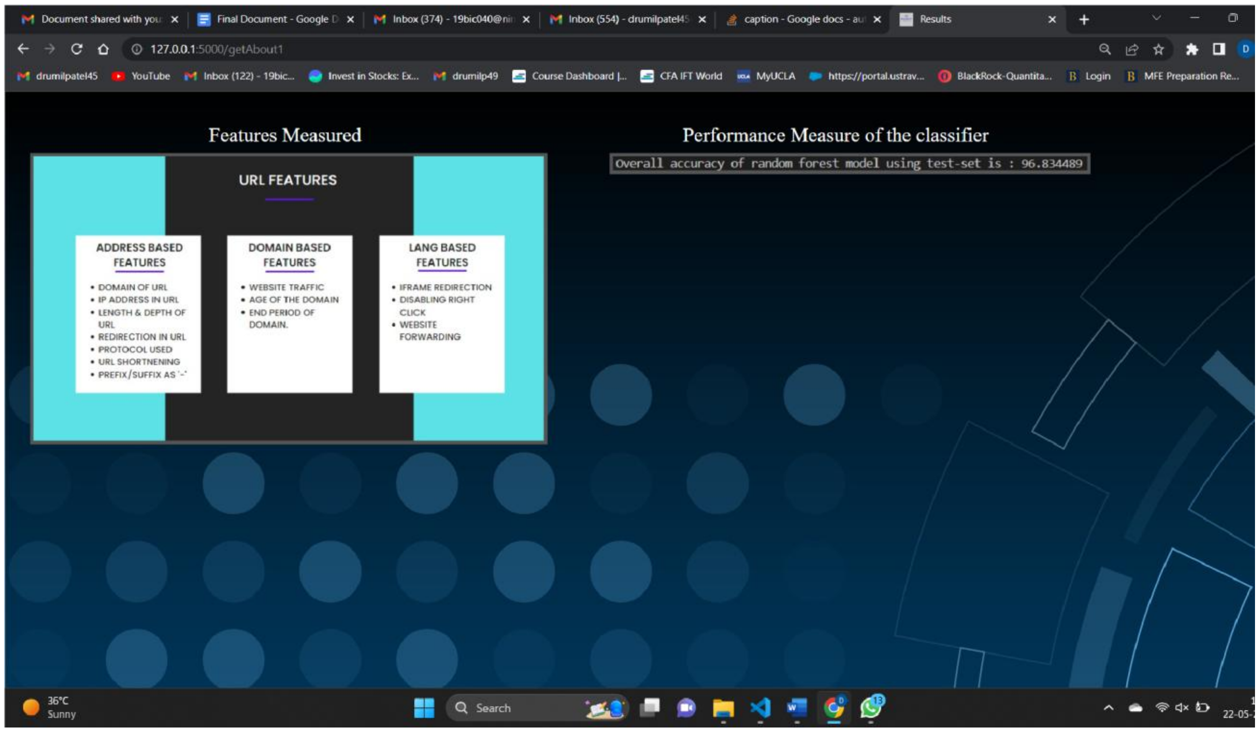
Advanced Phishing Detection: PhishPot uses ML algorithms to detect and identify phishing attempts accurately. By analyzing various URL parameters PhishPot can distinguish between legitimate and malicious communications, reducing the risk of falling victim to phishing attacks.

The website features an intuitive and user-friendly interface designed to make phishing detection accessible to users of all technical levels. With clear instructions and easily navigable sections, PhishPot offers a seamless experience for users seeking to safeguard themselves against phishing attacks

Multi-platform Support: PhishPot is designed to be compatible with multiple platforms, including desktops, laptops, tablets, and smartphones. Users can access the website from their preferred device, ensuring seamless protection against phishing attacks across various environments.

4.3 Frontend User Interface:


NIRMA UNIVERSITY
NAAC ACCREDITED 'A+' GRADE
PhishPot
Enter the URL here
Check
Click here to know more!!
Further Details!!
Contact us
Developed by: Nirma Students


Features Measured

URL FEATURES

ADDRESS BASED FEATURES
- DOMAIN OF URL
- IP ADDRESS IN URL
- LENGTH & DEPTH OF URL
- REDIRECTION IN URL
- PROTOCOL USED
- URL SHORTNENING
- PREFIX/SUFFIX AS '-'

DOMAIN BASED FEATURES
- WEBSITE TRAFFIC
- AGE OF THE DOMAIN
- END PERIOD OF DOMAIN.

LANG BASED FEATURES
- IFRAME REDIRECTION
- DISABLING RIGHT CLICK
- WEBSITE FORWARDING

Performance Measure of the classifier

```
Overall accuracy of random forest model using test-set is : 96.834489
```

4.4 Backend Architecture and technologies used:

Flask:-

Flask, a lightweight web framework written in Python, plays a pivotal role in the development of the phishing detection project described above. Flask offers a good environment for managing web requests and answers because of its simplicity and flexibility. The different URLs and their associated capabilities, such as the home page, URL submission, and other pages, are defined in this project using Flask's routes. These routes serve as the application's spine, executing particular code logic in response to user input.

oreover, Flask seamlessly integrates the machine learning model into the web application. Through Fla
he pre-trained Random Forest model (`rfmodel.pkl`) is loaded using the `pickle` library, enabling
model to be readily available for predicting whether a submitted URL is a phishing website or not. Fl
outes manage the process of feature extraction from the URL, data preparation, and passing ansformed
data to the model for evaluation. By harnessing Flask's capabilities, the integration of
machine learning model becomes efficient and streamlined.

Additionally, Flask's templating engine facilitates the rendering of dynamic web pages by incorporating HTML templates. In the phishing detection project, Flask templates are employed to display prediction results, error messages, and other relevant information to the users. The seamless integration of these templates with the code logic ensures a smooth and interactive user experience, enhancing the overall usability of the application. Overall, Flask's lightweight nature, Python compatibility, and versatile functionalities make it a valuable tool for developing web applications with integrated machine learning models. In the context of the phishing detection project, Flask empowers developers to handle web requests, seamlessly integrate the model, and render dynamic web pages, ultimately creating a user-friendly and efficient solution for detecting potential phishing attacks.

Flask Code Overview:-

The Flask code developed for the phishing detection website provides the necessary functionality to handle user requests, process URLs, extract features, and make predictions using the integrated machine learning model. The code starts by initializing a Flask application and defining routes for different pages. The `/` route renders the home page template, allowing users to submit a URL for phishing detection. The `/getURL` route is responsible for processing the user-submitted URL. It extracts relevant features from the URL using the `Features_Extraction` module and prepares the data for prediction. The pre-trained Random Forest model (`rfmodel.pkl`) is then loaded using the `pickle` library, and the extracted features are passed to the model for prediction. Depending on the prediction result, the appropriate response is generated and rendered on the website. If the prediction indicates a phishing website, the URL is added to the blacklist stored in the `blacklist.txt` file, helping to improve the system's effectiveness over time. Additionally, error handling is implemented to catch any exceptions and provide a consistent user experience. Overall, the Flask code forms the backbone of the phishing detection website, facilitating the seamless integration of the machine learning model and the user interface.

Machine Learning Integration:-

- The integration of the machine learning model is a crucial aspect of the phishing detection process in the Flask application.

- The code utilizes the Features_Extraction module to extract relevant features from the submitted URL, capturing important characteristics and patterns indicative of phishing attempts.

- The extracted features are transformed into a suitable format for prediction by the machine learning model.

- A pre-trained Random Forest model, trained on a carefully curated dataset, is loaded into memory using the 'pickle' library.

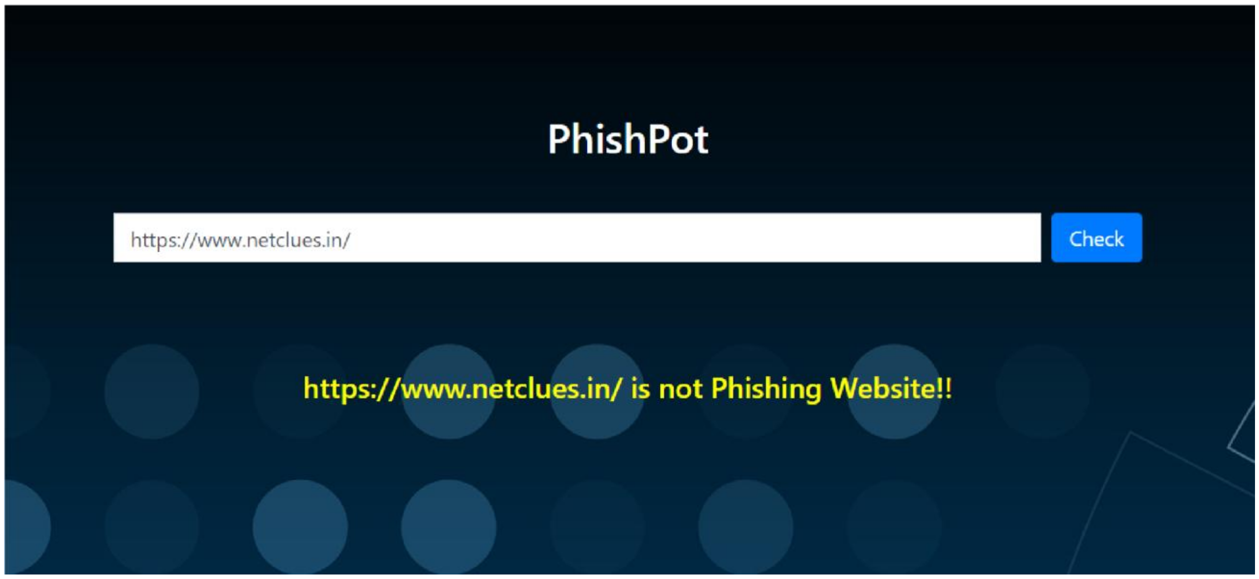
- The loaded model has learned to recognize patterns and make accurate predictions based on the extracted features.

- When a user submits a URL, the Flask route triggers the machine learning model and passes the transformed feature data to it.

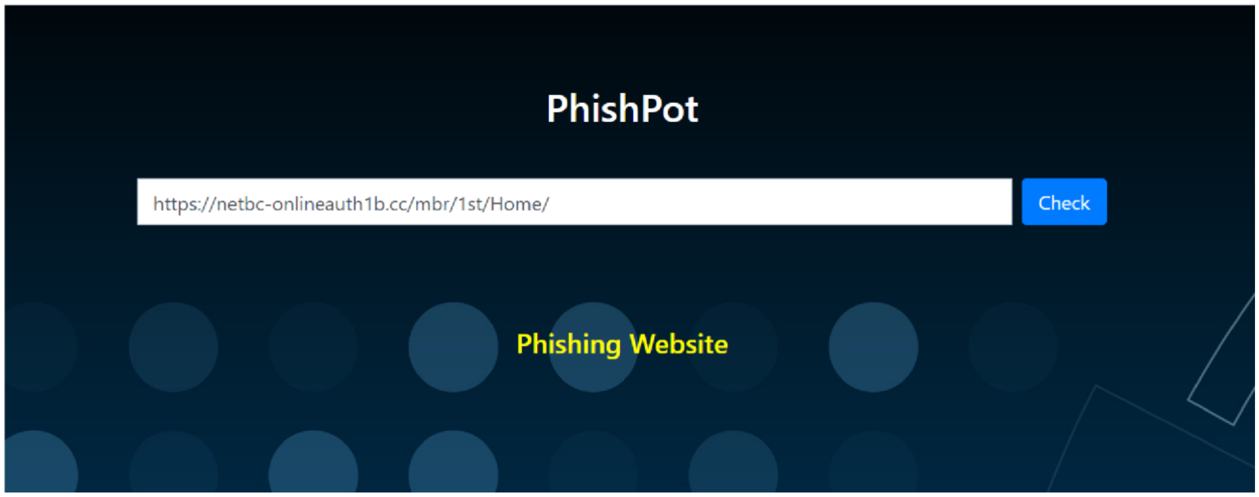
- The machine learning model predicts whether the URL corresponds to a phishing website or not based on the provided features.

- The prediction result obtained from the model is used to generate an appropriate response, indicating whether the URL is considered phishing or safe.

4.4 Results



https://www.netclues.in/ is not Phishing Website!!



Phishing Website

Conclusion:

In Conclusion, The project effectively combines Flask development and machine learning integration to create a robust phishing detection website. Leveraging the Flask framework, users can submit URLs for evaluation and receive real-time predictions. The integration of Random Forest classifier, allows for accurate detection of potential phishing websites. By extracting relevant features from the submitted URLs and utilizing a pre-trained Random Forest model, the system provides immediate feedback and protection against phishing threats. The integration ensures a user-friendly experience, while the visual consistency provided by HTML templates enhances the website's interface. Overall, this project exemplifies the successful fusion of Flask, Machine Learning, and Web Development, resulting in an efficient and reliable phishing detection solution named PhishPot.

References

1. Mohammed Hazim Alkawaz, Stephanie Joanne Steven and Asif Iqbal Hajamydeen, "Detecting Phishing Website Using Machine Learning", *16th IEEE International Colloquium on Signal Processing its Applications (CSPA 2020)*, 28-29 Feb. 2020.
2. S Nandhini and V. Vasanthi, *Extraction of Features and Classification on Phishing Websites using Web Mining Techniques*, vol. 5, no. 4, 2017, ISSN 2321-9939.
3. K. Thakur, J. Shan, A.-S.K. Pathan. Innovations of phishing defense: the mechanism, measurement and defense strategies.
4. J. Rashid, T. Mahmood, M.W. Nisar, T. Nazir. Phishing detection using machine learning techniques.
5. Gunter Ollmann, “The Phishing Guide Understanding & Preventing Phishing Attacks”, IBMInternetSecurity Systems, 2007.
6. https://resources.infosecinstitute.com/category/enterprise/phishing/the-phishing-landscape/phishing-data-attackstatistics.
7. Mahmoud Khonji, Youssef Iraqi, "Phishing Detection: A Literature Survey IEEE, and Andrew Jones, 2013
8. http://dataaspirant.com/2017/05/22/random-forest-algorithm-machine-learing/ 9. https://www.phishtank.com