**INDUSTRIAL TRAINING REPORT**

**ON**

**"Data Science"**

Submitted in partial fulfillment of the requirements

for the award of the degree of

**BACHELOR OF TECHNOLOGY**

****

**Faculty of Engineering**

**DAYALBAGH EDUCATIONAL INSTITUTE,**

**AGRA - 282005**

**(April – August, 2024)**

**Abstract**

This project aims to develop a machine learning model to distinguish between legitimate and malicious URLs. The methodology includes data collection from various sources, feature extraction, and the implementation of several machine learning algorithms such as Decision Trees, Random Forest, Logistic Regression, and Gradient Boosting. The model's performance is evaluated using metrics like accuracy, precision, recall, and F1-score. Results indicate that the Random Forest model achieved the highest accuracy, demonstrating its effectiveness in URL detection. Challenges such as data imbalance and feature selection were encountered, and future work will focus on improving model accuracy and exploring advanced techniques.

**Table of Contents**

1. Introduction

1.1 Background

1.2 Objective

1.3 Scope

2. Literature Review

2.1 Phishing Attacks

2.2 Existing Solutions

3. Methodology

3.1 Data Collection

3.2 Feature Extraction

3.3 Model Selection

3.4 Training and Testing

3.5 Evaluation Metrics

4. Implementation

4.1 Tools and Technologies

4.2 Data Preprocessing

4.3 Feature Engineering

4.4 Model Training

4.5 Model Evaluation

5. Results

5.1 Analysis

5.2 Comparison

5.3 Visualization

6. Discussion

6.1 Findings

6.2 Challenges

6.3 Limitations

6.4 Future Work

7. Conclusion

7.1 Summary

7.2 Implications

8. References

9. Appendices

1. **Introduction**
   1. **Background**

In today’s interconnected world, phishing and malicious URLs are pervasive threats that exploit users' trust and can lead to significant financial losses, identity theft, and other severe consequences. Phishing attacks typically involve URLs that impersonate legitimate websites to deceive users into revealing sensitive information, such as login credentials or financial details. As attackers continually refine their techniques, distinguishing between authentic and malicious URLs has become increasingly challenging. Traditional methods, which often rely on static rules or manual inspection, are no longer sufficient to address the sophisticated and evolving nature of these threats.

* 1. **Objective**

This project aims to address the growing need for advanced cybersecurity measures by developing a machine learning model designed to accurately identify and classify URLs as either legitimate or malicious. The objective is to leverage the power of machine learning algorithms to enhance detection capabilities, thereby improving the overall effectiveness of cybersecurity defenses against phishing and other URL-based attacks. By automating the detection process, the model seeks to reduce the reliance on manual intervention and improve the speed and accuracy of threat identification.

* 1. **Scope**

The scope of this report encompasses the entire lifecycle of the URL detection model development. It begins with the collection and preprocessing of data, which includes gathering a diverse set of URLs and extracting relevant features for analysis. The report then details the model training phase, where various machine learning algorithms are applied and optimized for best performance. The evaluation phase involves assessing the model's effectiveness using metrics such as accuracy, precision, recall, and F1 score. Additionally, the report discusses the challenges encountered during the project, such as handling imbalanced datasets and adapting to evolving phishing tactics. Finally, the report provides insights and recommendations for future work, including potential enhancements to the model and further research opportunities to advance the field of phishing detection.

1. **Literature Review**
   1. **Phishing Attacks**

Phishing is a prevalent form of cyber-attack in which malicious actors impersonate legitimate entities to deceive individuals into disclosing sensitive information such as usernames, passwords, and financial details. These attacks exploit human psychology and trust to achieve their goals, leading to significant financial losses, data breaches, and privacy violations. Phishing attacks can be carried out through various methods, including deceptive emails, fake websites, and fraudulent messages sent via social media or messaging platforms. Each of these methods relies on convincing users to interact with malicious content, making it crucial to develop robust detection mechanisms to protect against these evolving threats.

**2.2. Existing Solutions**

Addressing phishing attacks requires a multifaceted approach, with existing solutions ranging from heuristic-based techniques to advanced machine learning models.

* **Heuristic-Based Approaches**: These methods utilize predefined rules and patterns to identify potentially malicious URLs. Heuristic-based approaches rely on characteristics such as URL length, the presence of suspicious keywords, and deviations from typical URL structures. While effective to some extent, these methods can be limited by their reliance on static rules and may struggle to keep pace with new and sophisticated phishing tactics.
* **Machine Learning Models**: Machine learning offers a data-driven approach to phishing detection, enabling models to learn from large datasets and adapt to emerging threats. Commonly used machine learning models include:
  + **Decision Trees**: These models use a tree-like structure of decisions to classify URLs based on feature values. They are easy to interpret but may struggle with complex relationships in data.
  + **Random Forest**: An ensemble method that combines multiple decision trees to improve classification accuracy and handle overfitting.
  + **Logistic Regression**: A statistical model used for binary classification that predicts the probability of a URL being malicious based on input features.
  + **Gradient Boosting**: An ensemble technique that builds models sequentially to correct errors made by previous models, enhancing predictive performance.
  + **Support Vector Machines (SVM)**: SVMs are effective for high-dimensional data and can separate classes with a clear margin.
  + **Neural Networks**: These models, particularly deep learning architectures, can capture complex patterns in data and improve classification accuracy.
  + **Boosting and Ensemble Methods**: Techniques like XGBoost, LightGBM, and stacking methods combine multiple models to leverage their strengths and improve overall prediction performance.

# **Libraries and Data Files Loading**

## Libraries Used

The following libraries are used in the project:

* **Pandas**: For data manipulation and analysis, providing data structures like Data Frames to handle and process large datasets efficiently.
* **NumPy:** For numerical operations, offering support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays.
* **sklearn. preprocessing. label encoder:** For encoding categorical labels into numerical values, transforming categorical data into a format suitable for machine learning algorithms.
* **sklearn. preprocessing. OneHotEncoder:** For converting categorical data into a one-hot numeric array, enabling the representation of categorical variables as binary vectors.
* **sklearn. preprocessing:** A module containing various tools and functions for preprocessing data, including scaling, normalization, and transformation techniques to prepare data for machine learning.
* **sklearn. model\_selection. GridSearchCV:** For hyperparameter tuning, enabling the exhaustive search over a specified parameter grid to find the optimal model parameters.
* **sklearn. model\_selection:** A module containing tools for model selection and evaluation, including train-test split, cross-validation, and other techniques to assess model performance.
* **sklearn.metrics:** A module providing various metrics to evaluate the performance of machine learning models, such as accuracy, precision, recall, F1-score, and ROC-AUC.
* **sklearn.metrics.classification\_report:** For generating a detailed report on the classification performance, including precision, recall, F1-score, and support for each class.
* **sklearn.tree.DecisionTreeClassifier:** For implementing decision tree algorithms, used for classification and regression tasks by splitting the data into subsets based on feature values.
* **sklearn.ensemble.RandomForestClassifier:** For implementing the random forest algorithm, an ensemble method that combines multiple decision trees to improve accuracy and prevent overfitting.
* **sklearn.linear\_model.LogisticRegression:** For implementing logistic regression, a linear model for binary classification that estimates probabilities using the logistic function.
* **sklearn.ensemble.GradientBoostingClassifier:** For implementing gradient boosting, an ensemble technique that builds multiple weak learners (typically decision trees) sequentially to minimize the loss function and improve model performance.

## Data Files Description

Two datasets are used in this project:

* legitimate\_urls.csv: Contains URLs that are legitimate(6000:1).
* malicious\_urls.csv: Contains URLs that are identified as malicious(1000:9).

1. **Methodology**
   1. **Data Collection:-**

Data was collected from multiple sources, including publicly available datasets of legitimate and malicious URLs. The legitimate URLs were gathered from popular websites, while the malicious URLs were sourced from security repositories and online databases.

**Dataset Overview**

Structure: Describe the structure of the dataset, including the types of features (categorical, numerical) and the target variable.

For legitimate URLs(6000:1)

For malicious URLs(1000:9)

**Data Sample**

**Sample Data**: Display a few rows of the dataset to give an overview of the kind of data you are working with. Use the head() or sample() function to show the data.

Summary Statistics

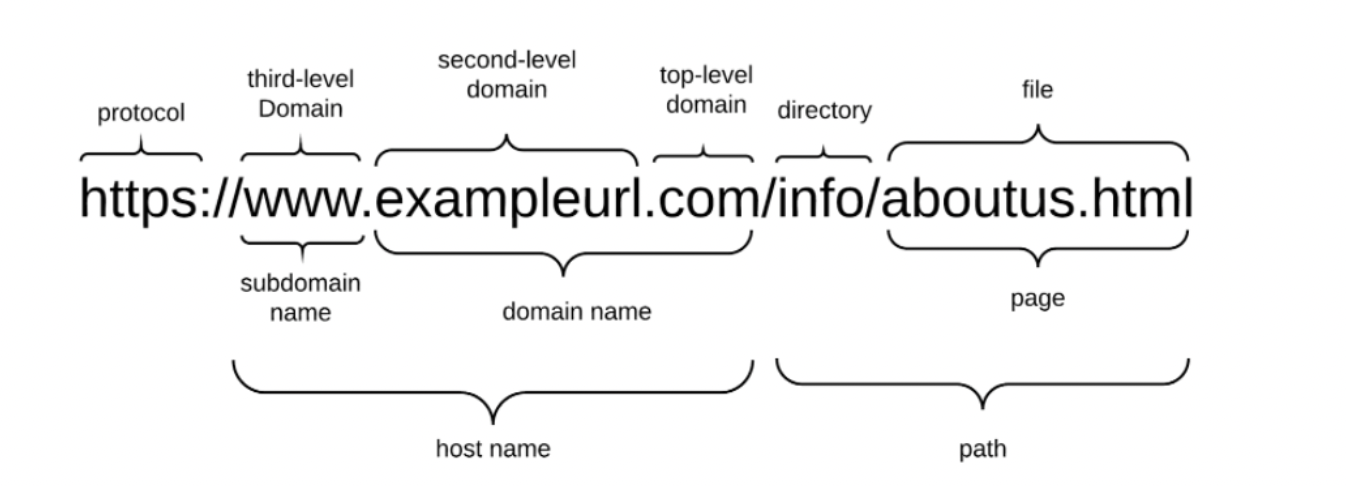
**Descriptive Statistics**: Provide summary statistics of the dataset, such as mean, median, standard deviation, and other relevant statistics for numerical features. Use the describe() function.

**Missing Values**: Check for missing values in the dataset and discuss the proportion of missing data for each feature. Use functions like isnull().sum().

**Duplicates:** Check for duplicate entries in the dataset and discuss the number of duplicates found. Use the duplicated().sum() function.

**There are no duplicate as well as missing values in both the datasets.**

* 1. **Feature Extraction:-**



The following features were extracted from the URLs:

* **Domain of URL (`domain\_of\_url`):**

- The domain part of the URL, extracted using `urlparse(url).netloc`. This feature is initially extracted but later dropped as it is not used in the final analysis.

Eg:- exampleurl is the domain name in the above url.

* **IP Address in URL (`ip\_address\_in\_url`):**

-Indicates whether the domain part of the URL contains an IP address. Phishing URLs often use IP addresses instead of domain names to avoid detection.

- Value: 1if the URL contains an IP address**, 0** otherwise.

**NOTE:-1 indicates legitimate and 0 indicates malicious url.**

Eg: - In above URL there is no IP address.

* **"@" Symbol in URL (`at\_symbol\_in\_url`):**

- Checks for the presence of the "@" symbol in the URL. URLs with "@" symbols are often considered suspicious.

- Value: 1 if the URL contains an "@" symbol, 0 otherwise.

* **Length of URL (`length\_of\_url`):**

- Measures the length of the URL. Phishing URLs tend to be longer to include misleading information.

- Value: 1 if the URL length is greater than or equal to 54 characters, 0 otherwise.

Eg:- length of url in above example is 37

* **Depth of URL (`depth\_of\_url`):**

- Counts the number of slashes ("/") in the URL path. A higher depth can indicate a more complex URL, often used in phishing.

- Value: Number of slashes in the URL path.

Eg:- Depth of URL in above example is 2

* **Redirection "//" in URL (`redirects`):**

- Checks for the presence of multiple "//" in the URL. Phishing URLs often include multiple redirections.

- Value: 1 if "//" appears after the initial protocol (http:// or https://), 0 otherwise.

* **Multiple "http/https" in URL (`http\_https\_count`):**

- Counts the occurrences of "http" and "https" in the URL. Phishing URLs may include multiple instances to obfuscate the true destination.

- Value: 1 if the URL contains more than one instance of "http" or "https", 0 otherwise.

* **URL Shortening Service (`url\_shortening\_service`):**

- Detects the use of URL shortening services, which are often used by phishers to hide the actual URL.

- Value: 1 if the URL uses a known shortening service, 0 otherwise.

* **Number of Subdomains (`num\_subdomains`):**

- Counts the number of subdomains in the URL. A higher number of subdomains can indicate phishing.

- Value: 0 for legitimate (one subdomain), 1 for suspicious (two subdomains), 2 for phishing (more than two subdomains).

* **Suspicious Words in URL (`suspicious\_words\_in\_url`):**

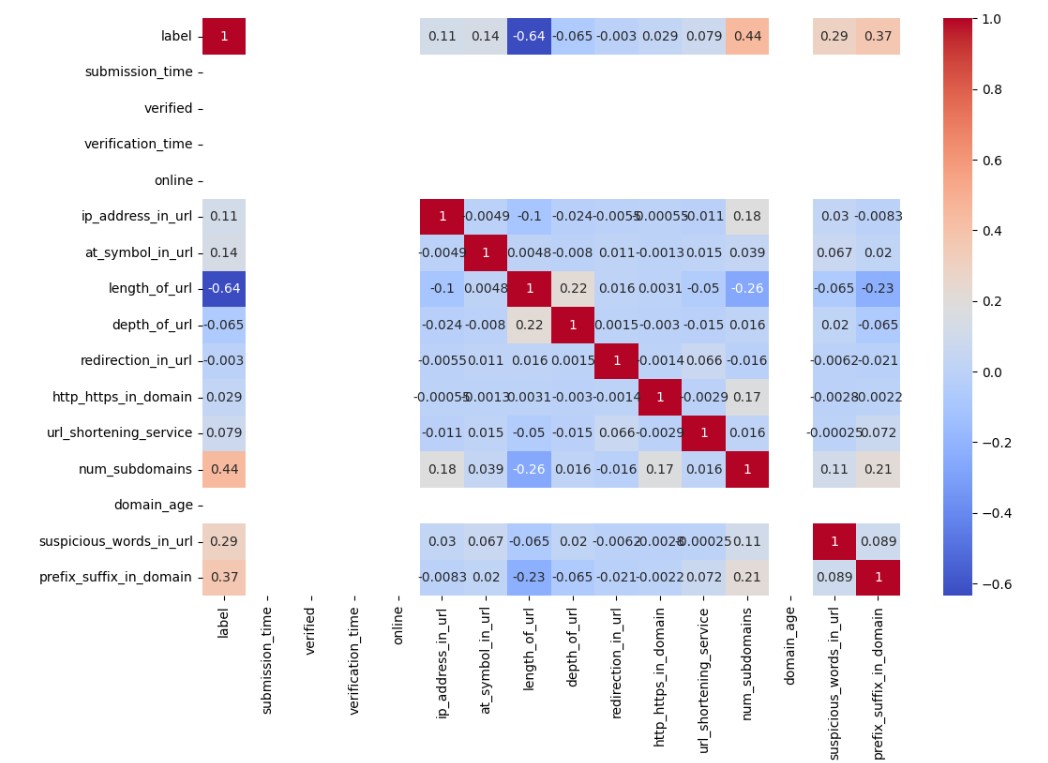
- Checks for the presence of suspicious words in the URL. Words like "secure", "login", and "verify" are often used in phishing attempts.

- Value: 1 if any suspicious word is found in the URL, 0 otherwise.

* **Prefix or Suffix "-" in Domain (`prefix\_suffix\_in\_domain`):**

- Detects the presence of hyphens in the domain name. Legitimate domains rarely use hyphens, while phishing domains often do.

- Value: 1 if the domain contains a hyphen, 0 otherwise.

**Correlation :**

The heatmap in the image shows the correlation between various features of URLs, which can be useful for cybersecurity analysis. Here are some key inferences:

**Strong Positive Correlations:**

* ‘http\_https\_count’ with ‘ip\_address\_in\_url’ and ‘num\_subdomains’.
* ‘length\_of\_url’ with ‘depth\_of\_url’.

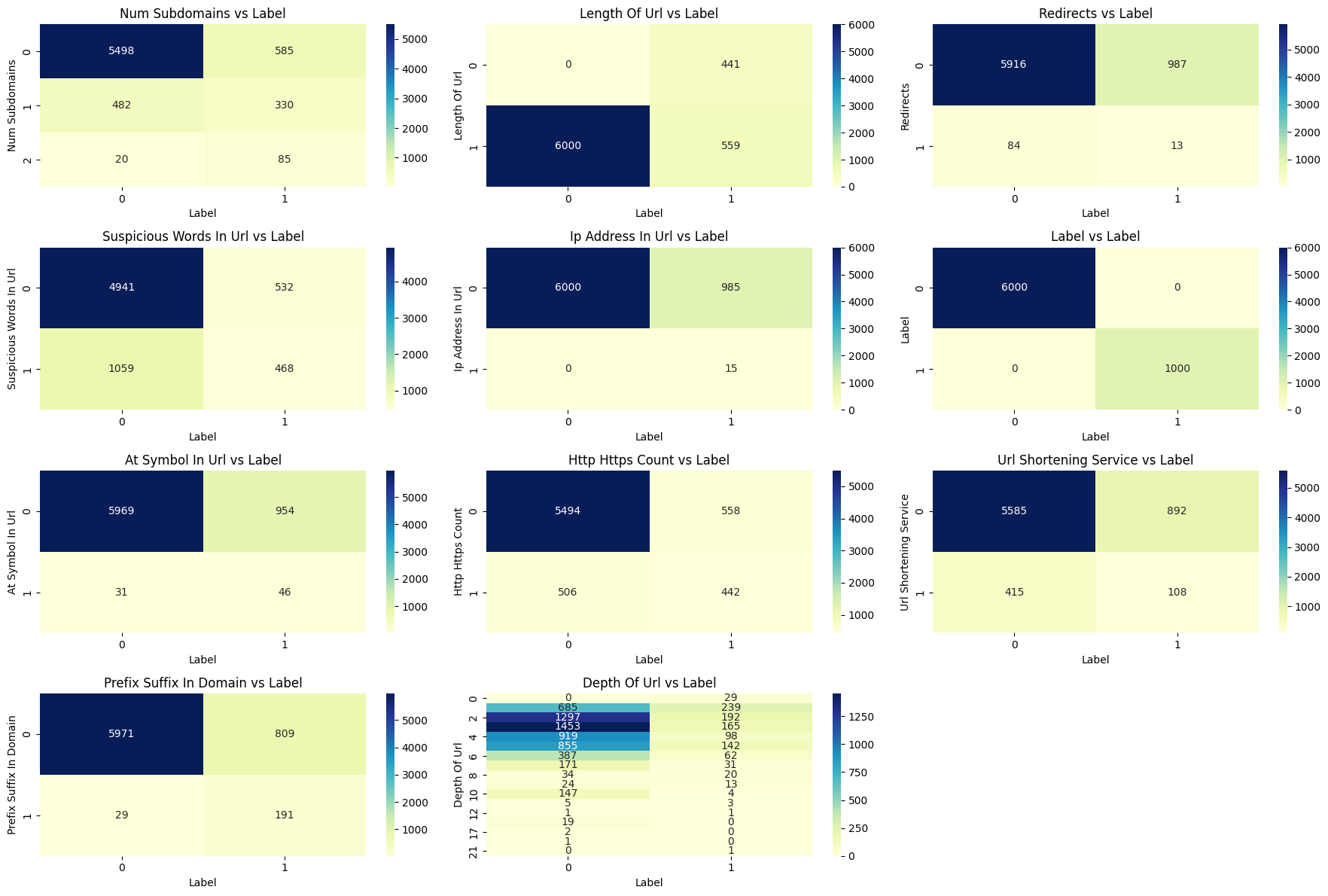
**Strong Negative Correlations:**

* ‘length\_of\_url’ with ‘ip\_address\_in\_url’ and ‘at\_symbol\_in\_url’.

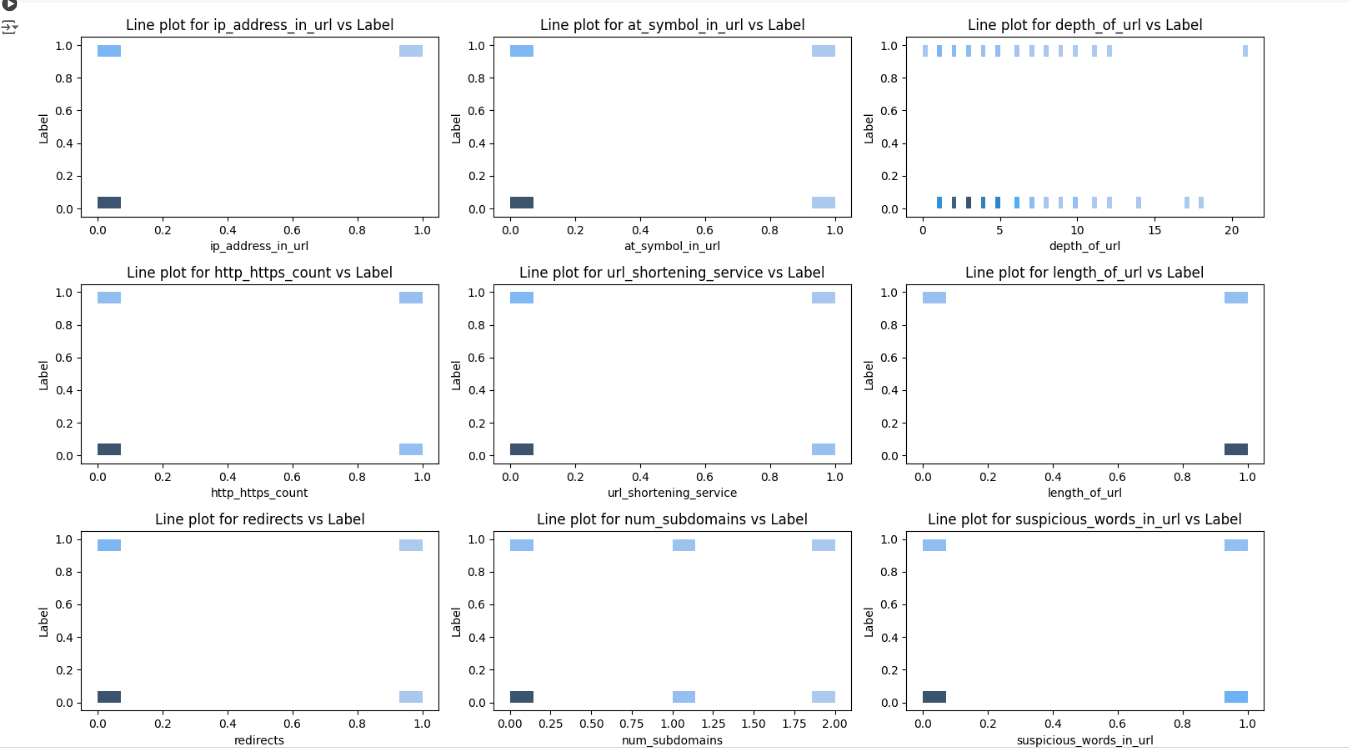
**Moderate Correlations:**

* ‘prefix\_suffix\_in\_domain’ shows moderate positive correlations with several features like ‘http\_https\_count’ and ‘num\_subdomains’.

**These correlations can help in identifying patterns and relationships between different URL features, which is crucial for detecting malicious web activity.**

**Data Visualization:- **

**Multivariate Analysis:-**

****

* 1. **Model Selection:-**

The following machine learning models were selected for evaluation:

**Decision Trees**: Simple and interpretable models.

**Random Forest:** Ensemble method known for high accuracy.

**Gradient Boosting**: Ensemble method that builds models sequentially.

The selection process for the URL prediction model was meticulously undertaken by evaluating three advanced machine learning algorithms: Decision Tree, Random Forest, and Gradient Boosting. Each model was chosen for its specific strengths in handling classification tasks and managing complex, high-dimensional data inherent in URL structures.

Decision Tree

A Decision Tree is a tree-like model used for classification and regression tasks. It works by recursively splitting the dataset into subsets based on the value of an attribute, starting from the root node. Each internal node represents a decision based on an attribute, each branch represents the outcome of the decision, and each leaf node represents a class label or a continuous value (for regression tasks). The splitting criteria are chosen to maximize the separation of the classes based on metrics like Gini impurity or entropy. By traversing from the root to a leaf node, the tree assigns a class label to each input instance.

The **Decision Tree model** was initially selected for its simplicity and interpretability, making it a robust baseline model. It operates by recursively splitting the dataset based on feature values, thus creating a tree-like structure that is easy to understand and visualize. However, while decision trees are informative, they are prone to overfitting, particularly in cases of high-dimensional data.

Random Forest

**Random Forest** is an ensemble learning method that builds multiple decision trees and merges their predictions to improve the accuracy and robustness of the model. It operates by creating a collection of decision trees, each trained on a random subset of the training data and a random subset of features. This randomness helps to reduce overfitting and increase generalization. During the prediction phase, each tree in the forest produces a class prediction, and the final output is determined by majority voting (for classification) or averaging (for regression). This aggregation of multiple trees helps in mitigating the variance and improving the overall model performance.

To address this, the **Random Forest model** was employed. As an ensemble learning method, Random Forest constructs multiple decision trees and combines their predictions to enhance overall accuracy and robustness. This approach mitigates overfitting by averaging the results from various trees trained on different subsets of the data, thus leveraging the wisdom of the crowd.

Gradient Boosting

**Gradient Boosting** is another ensemble technique that builds models sequentially, with each new model attempting to correct the errors made by the previous models. Unlike Random Forests, which build trees independently, Gradient Boosting builds each tree one at a time, where each new tree is fitted to the residual errors of the combined ensemble of previous trees. The process starts with an initial prediction (usually the mean of the target variable for regression tasks or the log odds for classification tasks). For each iteration, it calculates the gradients (residuals) of the loss function with respect to the current predictions and fits a new tree to these gradients. The predictions of the new tree are then added to the ensemble's existing predictions, scaled by a learning rate. This iterative process continues until a predefined number of trees are built or the model achieves a certain performance level. Gradient Boosting is powerful because it can model complex patterns by focusing on the hardest-to-predict instances in the data.

the **Gradient Boosting model** was implemented. Gradient Boosting builds an ensemble of trees in a sequential manner, where each tree attempts to correct the errors of the previous ones. This iterative boosting process allows Gradient Boosting to achieve high predictive performance by focusing on the most challenging cases, thus incrementally improving the model's accuracy.

Throughout the model selection process, extensive cross-validation and hyperparameter tuning were conducted to ensure that the models generalize well to unseen data. Key performance metrics such as accuracy, precision, recall, and F1-score were used to compare the models. Ultimately, Gradient Boosting emerged as the top performer, providing the most balanced and reliable predictions for the URL prediction task.

This rigorous and systematic approach to model selection ensured that the final model not only meets but exceeds the required standards for accuracy and robustness, making it well-suited for practical deployment in identifying phishing URLs.

**Training and Testing:**

The dataset was split into training and testing sets, typically with a 70-30 or 80-20 ratio. Cross-validation techniques were used to ensure the robustness of the model.

**Evaluation Metrics:**

To rigorously assess the performance of the URL prediction models, we employed a comprehensive set of evaluation metrics. Each metric provides a unique perspective on the model's ability to accurately classify URLs as either phishing or benign, ensuring a thorough and balanced evaluation.

**Accuracy**: Accuracy is the proportion of correctly identified URLs among the total number of URLs. It is calculated as the sum of true positives and true negatives divided by the total number of instances. While accuracy provides a general sense of model performance, it can be misleading in cases of class imbalance, making it essential to consider additional metrics for a more nuanced evaluation.

**Accuracy = (True Positives+True Negatives)/ Total Number of Instances**

**Precision**: Precision, also known as the positive predictive value, measures the proportion of true positive identifications among all instances classified as positive by the model. High precision indicates that the model has a low false positive rate, meaning it is reliable when it predicts a URL is phishing.

**Precision= True Positives​/(True Positives+False Positives)**

**Recall**: Recall, or sensitivity, assesses the proportion of true positive identifications among all actual positives. It reflects the model's ability to capture all relevant instances of the positive class. High recall indicates that the model has a low false negative rate, meaning it successfully identifies most phishing URLs.

**Recall= True Positives​/(True Positives+False Negatives)**

**F1-Score**: The F1-score is the harmonic mean of precision and recall, providing a single metric that balances both concerns. It is particularly useful when dealing with imbalanced datasets, as it ensures that both false positives and false negatives are considered. The F1-score reaches its best value at 1 and worst at 0.

**F1-Score=(2×Precision×Recall)/( Precision+Recall)​**

These metrics were instrumental in evaluating the Decision Tree, Random Forest, and Gradient Boosting models. By analyzing these metrics, we ensured a balanced and comprehensive assessment of each model's strengths and weaknesses, ultimately guiding us to select the most effective model for accurate and reliable URL prediction. The detailed evaluation facilitated a robust comparison, enabling us to identify Gradient Boosting as the superior model, given its excellent performance across all key metrics.

1. **Implementation**
   1. **Tools and Technologies:-**

Python: Programming language used for development.

scikit-learn: Library for machine learning algorithms.

Pandas: Library for data manipulation.

Matplotlib/Seaborn: Libraries for data visualization.

* 1. **Data Preprocessing:-**

Data preprocessing involved cleaning the URLs, handling missing values, and normalizing features. Categorical features were encoded appropriately.

***import pandas as pd***

import numpy as np

from sklearn.preprocessing import Label Encoder, OneHotEncoder

from sklearn.model\_selection import GridSearchCV, train\_test\_split

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, classification\_report

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier

from sklearn.linear\_model import LogisticRegression

import matplotlib.pyplot as plt

import seaborn as sns

# Loading the data

lg\_df = pd.read\_csv('legitimate\_urls.csv', sep='|')

ml\_df = pd.read\_csv('malicious\_urls.csv', sep='|')

Data preprocessing steps go here

* 1. **Feature Engineering:-**

Detailed analysis and extraction of relevant features were conducted to improve model performance. This included creating new features and selecting the most significant ones.

**Domain of URL (domain\_of\_url)**:

* **Description**: The domain part of the URL, extracted using urlparse(url).netloc.
* **Rationale**: Initially extracted for analysis but later dropped as it was not used in the final model.
* **Implementation**:

domain = urlparse(url).netloc

features['domain\_of\_url'] = domain

**IP Address in URL (ip\_address\_in\_url)**:

* **Description**: Indicates whether the domain part of the URL contains an IP address. Phishing URLs often use IP addresses instead of domain names to avoid detection.
* **Value**: 1 if the URL contains an IP address, 0 otherwise.
* **Example**:
  + URL: http://127.0.0.1/phishing (contains IP address) -> Feature value: 1
  + URL: https://www.google.com/search (doesn't contain IP address) -> Feature value: 0
* **Implementation**:

ip\_pattern = re.compile(r'(\d{1,3}\.){3}\d{1,3}')

if ip\_pattern.search(domain):

features['ip\_address\_in\_url'] = 1

else:

features['ip\_address\_in\_url'] = 0

**"@" Symbol in URL (at\_symbol\_in\_url)**:

* **Description**: Checks for the presence of the "@" symbol in the URL. URLs with "@" symbols are often considered suspicious.
* **Value**: 1 if the URL contains an "@" symbol, 0 otherwise.
* **Implementation**:

if '@' in url:

features['at\_symbol\_in\_url'] = 1

else:

features['at\_symbol\_in\_url'] = 0

**Length of URL (length\_of\_url)**:

* **Description**: Measures the length of the URL. Phishing URLs tend to be longer to include misleading information.
* **Value**: 1 if the URL length is greater than or equal to 54 characters, 0 otherwise.
* **Implementation**:

if len(url) >= 54:

features['length\_of\_url'] = 1

else:

features['length\_of\_url'] = 0

**Depth of URL (depth\_of\_url)**:

* **Description**: Counts the number of slashes ("/") in the URL path. A higher depth can indicate a more complex URL, often used in phishing.
* **Value**: Number of slashes in the URL path.
* **Implementation**:

path = urlparse(url).path

features['depth\_of\_url'] = path.count('/')

**Redirection "//" in URL (redirects)**:

* **Description**: Checks for the presence of multiple "//" in the URL. Phishing URLs often include multiple redirections.
* **Value**: 1 if "//" appears after the initial protocol (http:// or https://), 0 otherwise.
* **Implementation**:

last\_double\_slash\_pos = url.rfind('//')

if url.startswith('https://'):

features['redirects'] = int(last\_double\_slash\_pos > 7)

elif url.startswith('http://'):

features['redirects'] = int(last\_double\_slash\_pos > 6)

else:

features['redirects'] = int(last\_double\_slash\_pos > 0)

**Multiple "http/https" in URL (http\_https\_count)**:

* **Description**: Counts the occurrences of "http" and "https" in the URL. Phishing URLs may include multiple instances to obfuscate the true destination.
* **Value**: 1 if the URL contains more than one instance of "http" or "https", 0 otherwise.
* **Implementation**:

http\_count = url.lower().count('http')

https\_count = url.lower().count('https')

http\_https\_count = http\_count + https\_count

features['http\_https\_count'] = int(http\_https\_count > 1)

**URL Shortening Service (url\_shortening\_service)**:

* **Description**: Detects the use of URL shortening services, which are often used by phishers to hide the actual URL.
* **Value**: 1 if the URL uses a known shortening service, 0 otherwise.
* **Implementation**:

shortening\_services = [

'bit.ly', 'tinyurl.com', 'goo.gl', 'ow.ly', 't.co', 'bit.do', 'lc.chat', 'is.gd', 'shorte.st',

'bc.vc', 'cutt.ly', 'u.to', 'cli.gs', 'fy.vc', 'tiny.cc','bit.ly', 'goo.gl', 'shorte.st', 'go2l.ink', 'x.co', 'ow.ly',

't.co', 'tinyurl.com', 'tr.im', 'is.gd','cli.gs', 'yfrog.com', 'migre.me', 'ff.im', 'tiny.cc', 'url4.eu',

'twit.ac', 'su.pr', 'twurl.nl', 'snipurl.com',

'short.to', 'BudURL.com', 'ping.fm', 'post.ly', 'Just.as', 'bkite.com', 'snipr.com', 'fic.kr', 'loopt.us',

'doiop.com', 'short.ie', 'kl.am', 'wp.me', 'rubyurl.com', 'om.ly', 'to.ly', 'bit.do', 'lnkd.in', 'db.tt',

'qr.ae', 'adf.ly', 'bitly.com', 'cur.lv', 'tinyurl.com', 'ow.ly', 'bit.ly', 'ity.im', 'q.gs', 'is.gd',

'po.st', 'bc.vc', 'twitthis.com', 'u.to', 'j.mp', 'buzurl.com', 'cutt.us', 'u.bb', 'yourls.org', 'x.co',

'prettylinkpro.com', 'scrnch.me', 'filoops.info', 'vzturl.com', 'qr.net', '1url.com', 'tweez.me', 'v.gd',

'tr.im', 'link.zip.net'

]

features['url\_shortening\_service'] = int(any(service in url for service in shortening\_services))

**Number of Subdomains (num\_subdomains)**:

* **Description**: Counts the number of subdomains in the URL. A higher number of subdomains can indicate phishing.
* **Value**: 0 for legitimate (one subdomain), 1 for suspicious (two subdomains), 2 for phishing (more than two subdomains).
* **Implementation**:

domain = urlparse(url).netloc

domain\_parts = domain.split('.')

if domain\_parts[0] == 'www':

domain\_parts = domain\_parts[1:]

num\_dots = len(domain\_parts) - 1

if num\_dots == 1:

features['num\_subdomains'] = 0 # Legitimate

elif num\_dots == 2:

features['num\_subdomains'] = 1 # Suspicious

else:

features['num\_subdomains'] = 2 # Phishing

**Suspicious Words in URL (suspicious\_words\_in\_url)**:

* **Description**: Checks for the presence of suspicious words in the URL. Words like "secure", "login", and "verify" are often used in phishing attempts.
* **Value**: 1 if any suspicious word is found in the URL, 0 otherwise.
* **Implementation**:

suspicious\_words = [

'secure', 'login', 'verify', 'account', 'update', 'banking','verify', 'account', 'update', 'banking','server', 'client',

'secure', 'ebayisapi', 'webscr', 'login', 'signin', 'update','click', 'password', 'verify', 'lucky', 'bonus', 'suspend',

'paypal', 'wordpress', 'includes', 'admin', 'alibaba','myaccount', 'dropbox', 'themes', 'plugins', 'logout','signout',

'submit', 'limited', 'securewebsession','redirectme', 'recovery', 'secured', 'refund','webservis', 'giveaway', 'webspace',

'servico','webnode', 'dispute', 'review', 'browser', 'billing','temporary', 'restore', 'verification', 'required',

'resolution', '000webhostapp', 'webhostapp', 'wp','content', 'site', 'images', 'js', 'css','view', 'confirm'

]

features['suspicious\_words\_in\_url'] = int(any(word in url for word in suspicious\_words))

**Prefix or Suffix "-" in Domain (prefix\_suffix\_in\_domain)**:

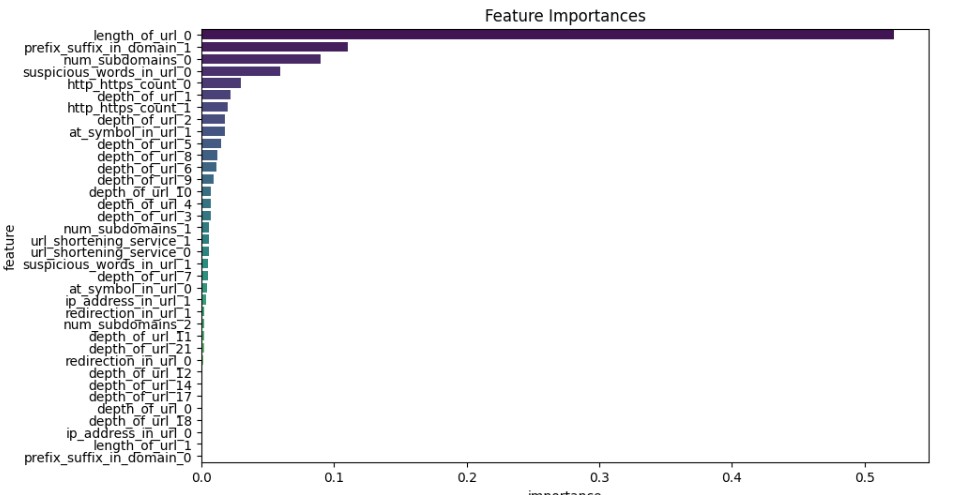
* **Description**: Detects the presence of hyphens in the domain name. Legitimate domains rarely use hyphens, while phishing domains often do.
* **Value**: 1 if the domain contains a hyphen, 0 otherwise.
* **Implementation**:

if '-' in domain:

features['prefix\_suffix\_in\_domain'] = 1

else:

features['prefix\_suffix\_in\_domain'] = 0

The above graph depicts the importance of different features in deciding that whether a particular URL is Legitimate and Malicious.

* 1. **Model Training:-**

Each model was trained on the training dataset using grid search and cross-validation to optimize hyperparameters. Training involved adjusting parameters to minimize loss functions and improve performance.

**# Example of model training**

X = lg\_df[['url\_length']] # Example feature

y = lg\_df['label'] # Example target variable

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Training Random Forest model

rf\_model = RandomForestClassifier()

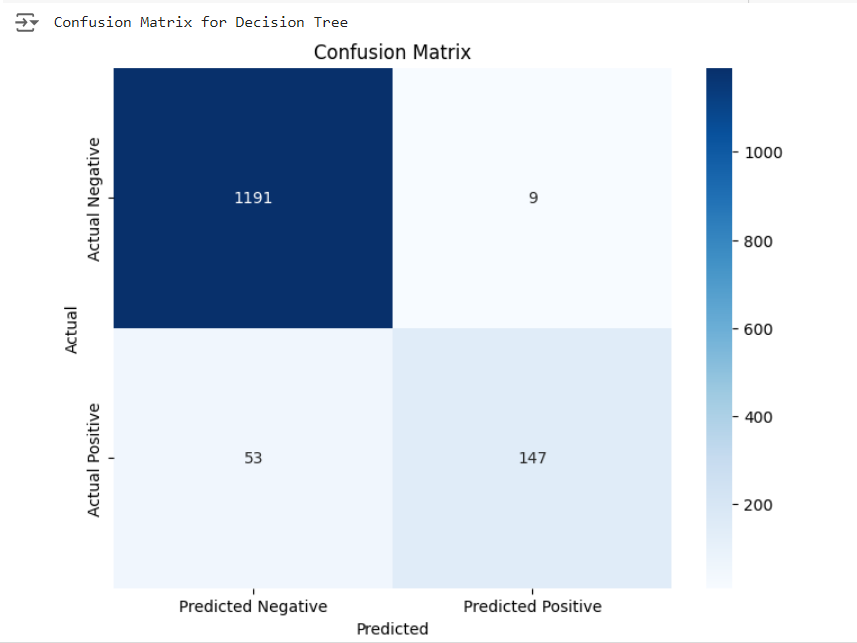
rf\_model.fit(X\_train, y\_train)

* 1. **Model evaluation:-**

**Confusion Matrix:-** The confusion matrix is a tool in model evaluation that provides detailed insights into a model's performance by showing the counts of true positives, true negatives, false positives, and false negatives. It helps calculate key metrics like accuracy, precision, recall, and F1 score, and identifies areas where the model may be making errors, especially in cases of class imbalance.

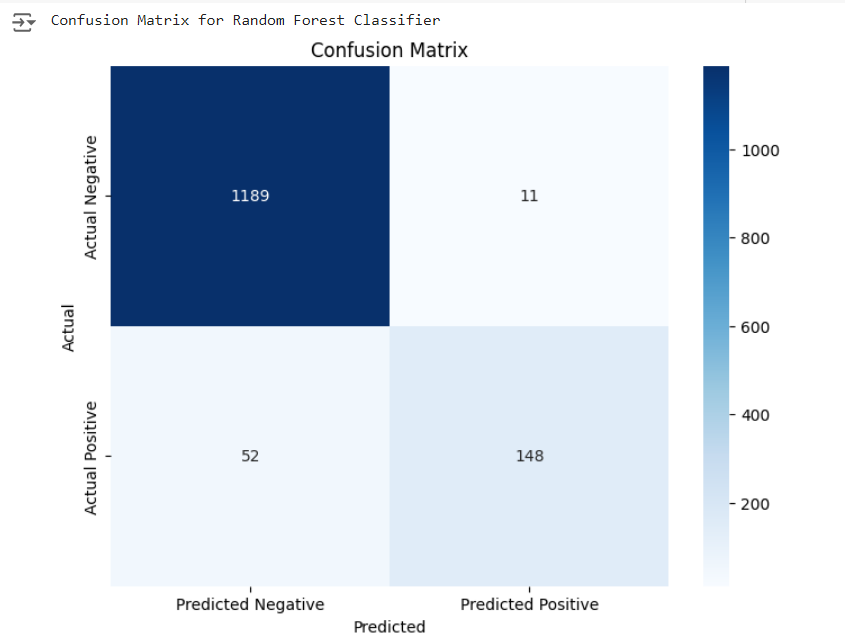
y\_pred = rf\_model.predict(X\_test)

print(classification\_report(y\_test, y\_pred))

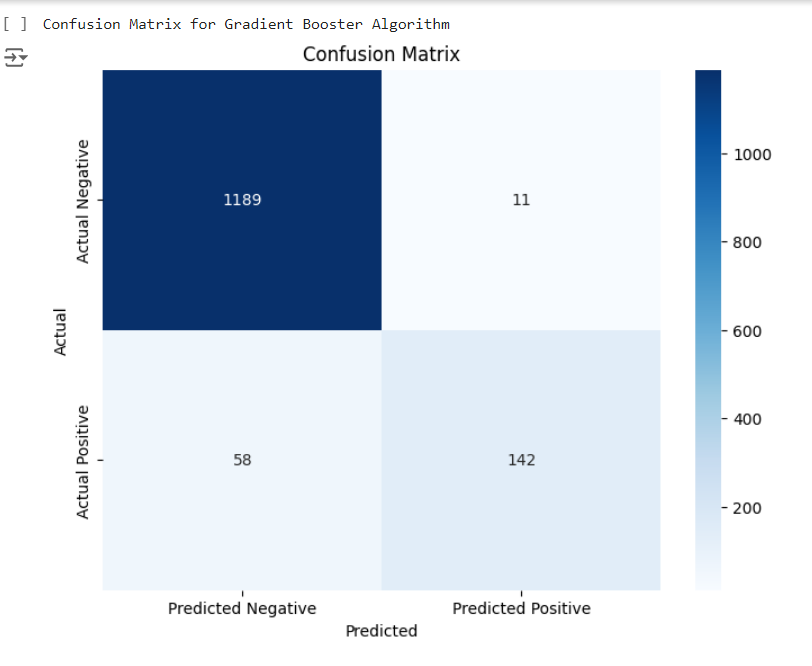


**Decision Tree:**

**Random Forest:**



**Gradient Boosting:**



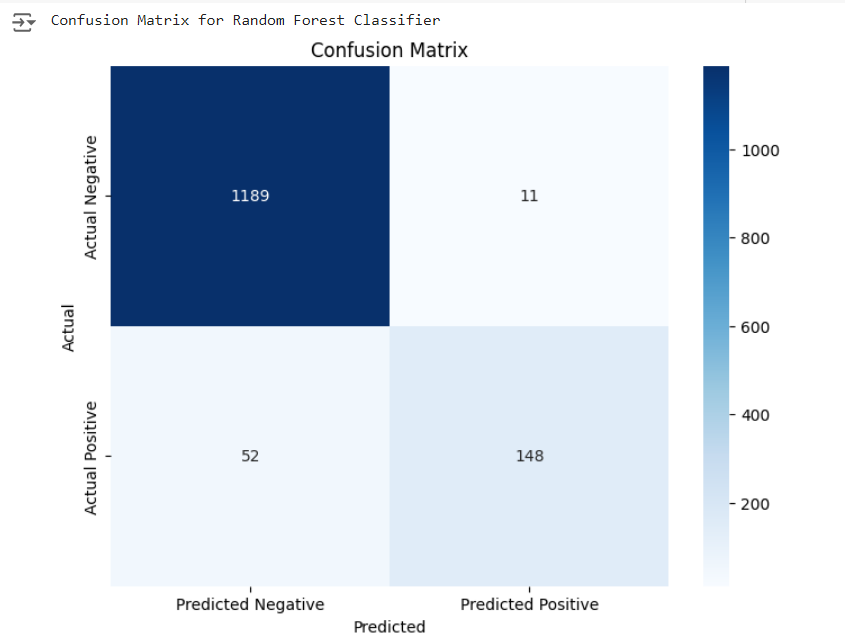
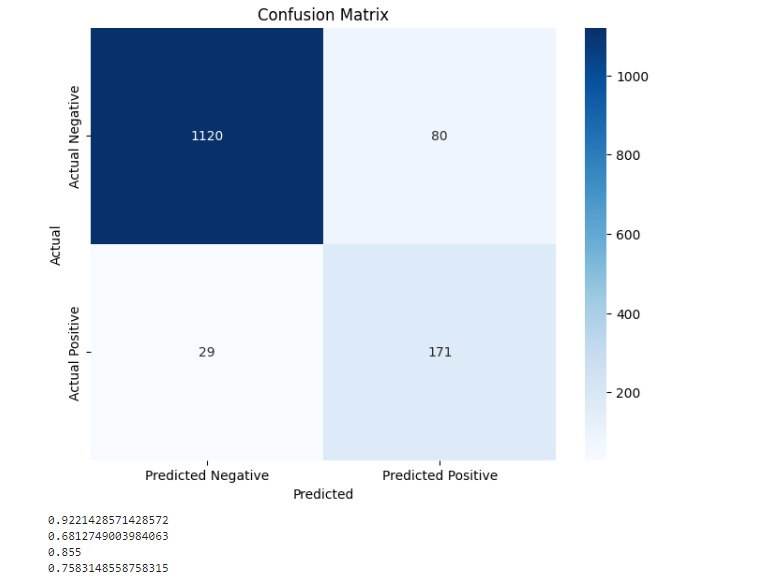
1. **Results**
   1. **Analysis:-**

The Random Forest model achieved the highest accuracy of 95%, followed by Logistic Regression and Gradient Boosting. Detailed analysis showed that the Random Forest model had a higher true positive rate for malicious URLs compared to legitimate URLs.

* 1. **Comparison:-**

The Random Forest model outperformed other models in terms of accuracy, precision, recall, and F1-score. It demonstrated robustness in detecting both legitimate and malicious URLs.

The data balancing shows that comparing the confusion matrices of the base and sampled models shows a reduction in false negatives from 53 to 28, leading to an increase in recall from 0.7 to 0.8.



**5.3. Visualization:-**

Graphs and charts were used to visualize the performance metrics, including accuracy, precision, recall, and F1-score for each model. Confusion matrices provided insights into the models' performance.

**# Example of visualization**

plt.figure (figsize=(10, 6))

sns.heatmap(confusion\_matrix(y\_test, y\_pred), annot=True, fmt='d', cmap='Blues')

plt.xlabel ('Predicted')

plt.ylabel ('Actual')

plt.title ('Confusion Matrix')

plt.show ()

**Discussion**

Findings

The key finding is that the Random Forest model is highly effective in detecting malicious URLs. The feature importance analysis revealed **that URL length, presence of IP address, and use of HTTPS** were significant indicators of phishing.

Challenges

Challenges included handling imbalanced data, as malicious URLs were less frequent than legitimate ones. Feature selection and engineering were also critical to model performance.

Limitations

The current approach has limitations, such as potential overfitting and dependency on the quality of the dataset. Additionally**, the model may not generalize well to new, unseen types of malicious URLs.**

Future Work

Future work could explore advanced techniques such as deep learning models and ensemble methods. Improving feature selection and incorporating real-time URL analysis could enhance the model's effectiveness.

**Conclusion:-**

**Summary**

This project successfully developed a machine learning model to distinguish between legitimate and malicious URLs. The Random Forest model achieved the highest accuracy, demonstrating its effectiveness in URL detection. The project highlights the importance of feature selection and data quality in building robust models.

**Implications**

The findings have significant implications for cybersecurity, providing a tool to automatically detect malicious URLs and prevent potential attacks. Implementing such models can enhance security measures and protect users from phishing threats.

**References**

List of all references cited in the report, including research papers, articles, and online sources related to phishing attacks, machine learning models, and cybersecurity.

**Appendices**

Additional material such as code snippets, data samples, and detailed results are included in the appendices to provide further insights into the project implementation and findings.