1. **Phishing URL**

   The overall solution design and process architecture for our phishing website detection system are structured to ensure robustness and effectiveness in identifying malicious websites. We begin by collecting a dataset containing labeled URLs, classifying them as either legitimate or phishing sites. The data preprocessing phase involves extracting relevant features from the URLs and splitting the dataset into training and testing sets. Multiple machine learning models, including XGBoost, SVM, Random Forest, Decision Tree, and a Stacking Classifier, are trained on the training dataset. To assess model performance, we evaluate them using the testing dataset and measure key metrics such as accuracy, precision, recall, F1-score, and ROC AUC score. This rigorous evaluation process aids in selecting the most suitable model for deployment.

   Following model training and evaluation, we analyze feature importances to gain insights into the critical factors influencing the models' predictions. The chosen model is then saved for deployment, ensuring that it remains accessible and efficient in a production environment. We implement an API that exposes the model's prediction capabilities, enabling users to make HTTP POST requests with URLs for analysis. The API preprocesses the URL, employs the trained model for prediction, and returns the result

2. **XGBoost (Extreme Gradient Boosting):**
   Reason for Use: XGBoost is a powerful ensemble learning technique known for its high predictive accuracy and efficiency. We chose XGBoost as it can effectively capture complex relationships in the data and handle both numerical and categorical features. It is particularly suitable for classification tasks like phishing detection.

   Support Vector Machine (SVM):
   Reason for Use: SVM is a well-established model for binary classification tasks. It works by finding the optimal hyperplane that best separates the data points of different classes. SVMs are effective when dealing with high-dimensional data, which is common in URL-based phishing detection.

   Random Forest:
   Reason for Use: Random Forest is an ensemble learning method that combines multiple decision trees. It is known for its robustness and ability to handle noisy data. We chose Random Forest because it reduces overfitting and provides feature importance’s, which help us understand which URL features are most important for classification.

   Decision Tree:
   Reason for Use: Decision Trees are interpretable and can be useful for understanding the logic behind classification decisions. We included Decision Trees to provide a baseline model and to visualize decision boundaries for educational purposes.

   Stacking Classifier:
   Reason for Use: Stacking is an ensemble technique that combines multiple machine learning models, leveraging the strengths of each. We included a Stacking Classifier to potentially

improve predictive performance by allowing models to learn from each other's strengths and weaknesses.

3. **Final output results and analysis of results**

13 Phishing Website Checker

```
import requests

url = 'https://www.google.com'

# Send a GET request to fetch the web page content
response = requests.get(url)

# Check if the request was successful (status code 200)
if response.status_code == 200:
    # Get the content of the web page as text
    web_page_content = response.text

    # Perform your analysis or processing on the web_page_content
    # For example, you can check for specific keywords or patterns
    if 'phishing' in web_page_content.lower():
        print("Phishing detected")
    else:
        print("Legitimate")

else:
    print(f"Failed to fetch the web page. Status code: {response.status_code}")
```

```
Legitimate
```

Here I have provided URL: https://www.google.com and get output as Legitimate website

2nd dataset – spam emails

Problem
1. Data Collection: The process begins with the collection of text data, which can come from various sources such as emails, social media posts, or documents. This data is usually unstructured and may contain noise, requiring preprocessing.

Data Preprocessing: Before building a machine learning model, the text data undergoes preprocessing steps. These steps include removing special characters, handling accented characters, tokenization (breaking text into words or phrases), removing stopwords (common words like "the" and "and"), and potentially correcting spelling errors.

Feature Extraction: To convert text data into a format that machine learning models can understand, feature extraction techniques like TF-IDF (Term Frequency-Inverse Document Frequency) or word embeddings like Word2Vec or GloVe are applied. These methods represent words as numerical vectors.

Model Selection: A suitable machine learning or deep learning model is selected for text classification. Common choices include Support Vector Machines (SVMs), Naive Bayes

2. Logistic Regression:

Reasons to Use: Logistic Regression is a simple and interpretable model that works well for binary text classification tasks. It's a good choice when the dataset is small, and you want to understand the importance of each feature (word) in making predictions.

Naive Bayes:

Reasons to Use: Naive Bayes models, such as Multinomial Naive Bayes, are efficient and work well with text data, especially in cases where the dataset is not very large. They are known for their speed and can be used as a baseline model.

Support Vector Machines (SVM):

Reasons to Use: SVMs are versatile and perform well in text classification tasks, especially when there are complex decision boundaries. They are effective for both binary and multi-class classification problems.

Random Forest and Decision Trees:

Reasons to Use: Decision tree-based models like Random Forest can capture non-linear relationships in text data. They are robust and can handle noisy data.

Result: I am getting error so unable to get results in it