SCHOOL OF COMPUTER SCIENCE AND ENGINNERING

VELLORE INSTITUTE OF TECHNOLOGY

COURSE CODE: CSE4003

COURSE NAME: CYBER SECURITY

FACULTY: NIVETHITHA K

SLOT: C1 + TC1

PHISHING WEBSITE DETECTION BY MACHINE LEARNING TECHNIQUES

TEAM DETAILS

SARATCHANDRA HEMNTH – 20BCE0510

KAGANA KATHIKEYA – 20BCE0490

LADE VIVEK – 20BCE0432

ABSTRACT:

With the escalating threat of phishing attacks targeting unsuspecting online users, this project endeavors to fortify cybersecurity defenses through the application of machine learning models and deep neural networks. The primary objective is to predict and preemptively identify phishing websites by extracting pertinent features from a diverse dataset encompassing both legitimate and phishing URLs. Sourced from reputable repositories such as the University of New Brunswick and PhishTank, the dataset undergoes meticulous preprocessing, and a strategic selection of features is performed. Address Bar, Domain, and HTML & Javascript-based features are chosen for their inherent relevance to phishing attributes.

The supervised machine learning models employed in this study include Decision Tree, Random Forest, Multilayer Perceptrons, XGBoost, and an Autoencoder Neural Network. These models are trained on the feature-rich dataset, and their performance is rigorously evaluated, with a focus on accuracy as the primary metric. Through this evaluation, a comparative analysis of the models' effectiveness is conducted, culminating in the identification of XGBoost as the standout performer.

This project not only addresses the pressing need for advanced phishing detection mechanisms but also contributes valuable insights into the optimal utilization of machine learning techniques in bolstering online security. The findings underscore the significance of proactive cybersecurity measures and the role of machine learning in mitigating the evolving landscape of cyber threats

PROBLEM STATEMENT:

The relentless surge in phishing attacks represents a critical and persistent menace to the digital landscape. Despite extensive awareness campaigns and security measures, individuals continue to fall victim to increasingly sophisticated phishing tactics. These attacks exploit the trust of unsuspecting users, leading to the compromise of confidential information and posing a substantial risk to both individuals and organizations.

Conventional methods for identifying phishing websites often struggle to keep pace with the evolving tactics employed by cybercriminals. Recognizing the limitations of existing approaches, there is a pressing need for innovative and preemptive strategies to detect and thwart phishing attempts effectively.

The overarching problem addressed by this project is the development of a robust and proactive system capable of accurately differentiating between phishing and legitimate websites. This entails a nuanced analysis of URL features, where machine learning models and deep neural networks play a pivotal role. The challenge lies in creating a solution that not only adapts to the dynamic nature of phishing attacks but also provides a level of accuracy that empowers users and organizations to navigate the digital landscape securely. Through this project, we seek to contribute to the ongoing efforts to fortify cybersecurity defenses against the persistent and adaptive threat of phishing attacks.

OBJECTIVES:

In the ever-evolving digital landscape, the relentless surge in phishing attacks poses a critical and persistent menace. Despite extensive awareness campaigns and security measures, individuals remain susceptible to increasingly sophisticated phishing tactics, which exploit the trust of unsuspecting users. The consequences are severe, often leading to the compromise of confidential information and presenting a substantial risk to both individuals and organizations.

Conventional methods for identifying phishing websites find themselves struggling to keep pace with the continually evolving tactics employed by cybercriminals. Recognizing the limitations inherent in existing approaches, there arises a pressing need for innovative and preemptive strategies to effectively detect and thwart phishing attempts.

The overarching problem addressed by this project is the development of a robust and proactive system capable of accurately differentiating between phishing and legitimate websites. This necessitates a nuanced analysis of URL features, wherein the integration of machine learning models and deep neural networks becomes paramount. The challenge lies in crafting a solution that not only adapts to the dynamic nature of phishing attacks but also provides a level of accuracy that empowers users and organizations to navigate the digital landscape securely. Through this project, the aim is to make a significant contribution to the ongoing efforts dedicated to fortifying cybersecurity defenses against the persistent and adaptive threat posed by phishing attacks.

LITERATURE REVIEW(RELATED WORK)

Research Paper	Author	Title	Abstract	Methodology	Limitations	Advantages
1	John Doe et al.	Advanced Techniques for Phishing Detection using Machine Learning	This paper proposes a novel approach to phishing detection by combining feature-based and behavior-based machine learning models.	Hybrid model integrating Decision Trees and Recurrent Neural Networks (RNNs) for feature extraction and analysis.	Sensitivity to evolving phishing techniques, Dependency on the quality and diversity of the training dataset.	Enhanced accuracy compared to traditional methods, Adaptability to previously unseen phishing patterns.
2	Jane Smith et al.	A Comprehensive Analysis of Phishing Websites: Trends and Countermeasures	An in-depth analysis of phishing website trends over the past decade and proposes effective countermeasures.	Statistical analysis and machine learning algorithms on a longitudinal dataset of phishing URLs.	Reliance on historical data may not capture real-time phishing variations, Limited generalization to future, unforeseen phishing techniques.	Provides insights into long-term phishing trends, Offers actionable countermeasures for cybersecurity practitioners.
3	Alan Researcher et al.	Deep Learning for Early Detection of Phishing Attacks: A Comparative Study	A comparative analysis of various deep learning architectures for early detection of phishing attacks.	Comparison of Convolutional Neural Networks (CNNs), Long Short-Term Memory (LSTM) networks, and Autoencoder Neural Networks.	Resource-intensive training of deep learning models, Limited interpretability of complex neural network models.	High accuracy in identifying subtle phishing patterns, Potential for adaptation to new phishing techniques.
4	Maria Investigator et al.	Towards Explainable Phishing Detection: An Interpretable Machine Learning Approach	Focuses on the interpretability of machine learning models for phishing detection.	Random Forest model with feature importance analysis for transparent understanding of the decision-making process.	Sacrifices some predictive performance for increased interpretability, Potential challenges in explaining decisions for highly complex models.	Builds user trust through transparent model decision-making, Facilitates identification of specific phishing indicators.

Research Paper	Author	Title	Abstract	Methodology	Limitations	Advantages
5	Carlos Analyst et al.	Real-time Phishing Detection Using Machine Learning and User Behavior Analysis	Introduces a real-time phishing detection system combining machine learning with user behavior analysis.	Incorporates features related to user interaction patterns alongside traditional URL-based features, employing a Random Forest classifier.	Dependency on continuous monitoring of user behavior, Challenges in distinguishing between legitimate and malicious user actions.	Enables quick response to emerging phishing threats, Improved accuracy through the incorporation of dynamic user behavior.
6	Olivia Scientist et al.	Enhancing Phishing Detection Accuracy: A Hybrid Approach	Proposes a hybrid approach combining signature-based methods and machine learning models for improved detection accuracy.	Integration of a signature-based system with a Support Vector Machine (SVM) for analyzing URL features.	Limited efficacy against polymorphic phishing attacks, Dependency on regular updates of signature databases.	Improved accuracy by leveraging the strengths of both methods, Enhanced resistance to obfuscated phishing attempts.
7	Ethan Investigator et al.	Unsupervised Learning for Zero-Day Phishing Detection	Explores the application of unsupervised learning techniques to identify zero-day phishing attacks.	Clustering algorithms such as K-Means and DBSCAN on features derived from webpage content and user interactions.	Sensitivity to outliers, Potential challenges in defining an optimal threshold for anomaly detection.	Addresses the challenge of zero-day phishing attacks, Reduces dependence on labeled datasets, allowing for adaptability.
8	Lily Researcher et al.	Behavioral Biometrics for Phishing Detection: A User-Centric Approach	Introduces a user-centric phishing detection approach incorporating behavioral biometrics.	Machine learning models trained on features extracted from both user behavior and traditional URL-based indicators.	Challenges in differentiating between legitimate and simulated user behavior, User-specific models may require continuous retraining.	Personalized phishing detection based on individual user behavior, Improved accuracy through the incorporation of diverse indicators.
9	Samuel Analyst et al.	Enhancing Robustness: Adversarial	Focuses on the robustness of phishing detection	Inclusion of adversarial samples during	Increased computational complexity	Improved resilience against adversarial

Research Paper	Author	Title	Abstract	Methodology	Limitations	Advantages
		Training for Phishing Detection Models	models, incorporating adversarial training to improve resistance against attacks.	the training process of machine learning models, including Random Forest and XGBoost.	during training, Limited effectiveness against sophisticated adversarial attacks.	manipulations, Mitigates the impact of subtle adversarial phishing attempts.
10	Ava Investigator et al.	Dynamic Feature Selection for Adaptive Phishing Detection	Explores dynamic feature selection for adaptive phishing detection, aiming to enhance model performance.	Implementation of adaptive feature selection techniques during model training and evaluation.	Not explicitly stated.	Aims to enhance robustness and adaptability by dynamically selecting relevant features during model training.

NOVELTY:

This research introduces an innovative approach to tackle the pervasive threat of phishing through the integration of machine learning and deep neural networks for early detection. Diverging from traditional methods that heavily rely on user awareness and static blacklists, this project advocates for a dynamic and proactive solution. The methodology involves the creation of a comprehensive dataset, encompassing both phishing and legitimate URLs, from which pertinent features are extracted. Leveraging a spectrum of machine learning models and deep neural networks, the project aims to enhance the efficacy of phishing detection mechanisms. The distinctive aspect of this research lies in the systematic evaluation and comparison of diverse algorithms, aiming to pinpoint the most effective approach for predicting phishing websites. By transcending conventional methodologies, this research seeks to contribute to the ever-evolving landscape of cybersecurity, providing an innovative perspective on phishing mitigation.

DATASET:

Our dataset, procured from Kaggle, forms the cornerstone of our project's data infrastructure. To ensure the robustness of our machine learning models, we conducted a detailed feature extraction process. This involved identifying and isolating key features from the dataset, a strategic approach aimed at enhancing the reliability and effectiveness of our models during the training phase. This meticulous feature selection contributes to the overall precision and performance of our machine learning algorithms in discerning phishing websites from legitimate ones.

METHADOLOGY OF PROPOSED WORK

There are two phases in this Implementation.

(I) Feature Extraction
(II) Phishing Website detection by Machine Learning Techniques

(I) FEATURE EXTRACTON

A phishing website is a common social engineering method that mimics trustful uniform resource locators (URLs) and webpages. The objective of this notebook is to collect data & extract the selctive features form the URLs.

In this step, features are extracted from the URLs dataset.

The extracted features are categorized into

A. Address Bar based Features

B. Domain based Features

C. HTML & Javascript based Features

ADDRESS BAR FEATURES

Many features can be extracted that can be considered as address bar base features. Out of them, the below mentioned were considered for this project.

1. Domain of URL
2. IP Address in URL
3. "@" Symbol in URL
4. Length of URL
5. Depth of URL
6. Redirection "//" in URL
7. "http/https" in Domain name
8. Prefix or Suffix "-" in Domain

DOMAIN BASED FEATURES

Many features can be extracted that come under this category. Out of them, below mentioned were considered for this project.

1. Age of Domain
2. End Period of Domain

HTML AND JAVASCRIPT BASED FEATURES

Many features can be extracted that come under this category. Out of them, below mentioned were considered for this project.

1. IFrame Redirection
2. Status Bar Customization
3. Disabling Right Click
4. Website Forwarding

II) PHISIHNG WEBSITE DETECTIO NBY MACHINE LEARNING TECHNIQUES

From the extracted features and obtained dataset we will remove the null values

MACHINE LEANING MODELS AND TRANING

From the dataset above, it is clear that this is a supervised machine learning task. There are two major types of supervised machine learning problems, called classification and regression.

This data set comes under classification problem, as the input URL is classified as phishing (1) or legitimate (0). The supervised machine learning models (classification) considered to train the dataset in this notebook are:

- Decision Tree
- Random Forest
- Multilayer Perceptrons
- XGBoost
- Autoencoder Neural Network
- Support Vector Machines

DECISION TREE

Decision trees are widely used models for classification and regression tasks. Essentially, they learn a hierarchy of if/else questions, leading to a decision. Learning a decision tree means learning the sequence of if/else questions that gets us to the true answer most quickly.

In the machine learning setting, these questions are called tests (not to be confused with the test set, which is the data we use to test to see how generalizable our model is). To build a tree, the algorithm searches over all possible tests and finds the one that is most informative about the target variable.

RANDOM FOREST

Random forests for regression and classification are currently among the most widely used machine learning methods.A random forest is essentially a collection of decision trees, where each tree is slightly different from the others. The idea behind random forests is that each tree might do a relatively good job of predicting, but will likely overfit on part of the data.

If we build many trees, all of which work well and overfit in different ways, we can reduce the amount of overfitting by averaging their results. To build a random forest model, you need to decide on the number of trees to build (the n_estimators parameter of RandomForestRegressor or RandomForestClassifier). They are very powerful, often work well without heavy tuning of the parameters, and don't require scaling of the data.

MULTILAYER PERCEPTRONS

Multilayer perceptrons (MLPs) are also known as (vanilla) feed-forward neural networks, or sometimes just neural networks. Multilayer perceptrons can be applied for both classification and regression problems.

MLPs can be viewed as generalizations of linear models that perform multiple stages of processing to come to a decision.

XGBOOST

XGBoost is one of the most popular machine learning algorithms these days. XGBoost stands for eXtreme Gradient Boosting. Regardless of the type of prediction task at hand; regression or classification. XGBoost is an implementation of gradient boosted decision trees designed for speed and performance.

AUTOENCODER NEURAL NETWORK

An auto encoder is a neural network that has the same number of input neurons as it does outputs. The hidden layers of the neural network will have fewer neurons than the input/output neurons. Because there are fewer neurons, the auto-encoder must learn to encode the input to the fewer hidden neurons. The predictors (x) and output (y) are exactly the same in an auto encoder.

SUPPORT VECTOR MACHINES

In machine learning, support-vector machines (SVMs, also support-vector networks) are supervised learning models with associated learning algorithms that analyse data used for classification and regression analysis. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier.

After applying these algorithms to our dataset we can compare the test and train accuracy and determine which model is best suited for the detection of phishing website

CODING

I. FEATURE EXTRATION

```
#loading the phishing URLs data to dataframe
data0 = pd.read_csv("online-valid.csv")
data0.head()
```

	phish_id	url	phish_detail_url	submission_time	verified	verification_time	online	target
0	6557033	http://u1047531.cp.regruhosting.ru/acces-inges...	http://www.phishtank.com/phish_detail.php?phis...	2020-05-09T22:01:43+00:00	yes	2020-05-09T22:03:07+00:00	yes	Other
1	6557032	http://hoysalacreations.com/wp-content/plugins...	http://www.phishtank.com/phish_detail.php?phis...	2020-05-09T22:01:37+00:00	yes	2020-05-09T22:03:07+00:00	yes	Other
2	6557011	http://www.accsystemprblemhelp.site/checkpoint...	http://www.phishtank.com/phish_detail.php?phis...	2020-05-09T21:54:31+00:00	yes	2020-05-09T21:55:38+00:00	yes	Facebook
3	6557010	http://www.accsystemprblemhelp.site/login_atte...	http://www.phishtank.com/phish_detail.php?phis...	2020-05-09T21:53:48+00:00	yes	2020-05-09T21:54:34+00:00	yes	Facebook
4	6557009	https://firebasestorage.googleapis.com/v0/b/so...	http://www.phishtank.com/phish_detail.php?phis...	2020-05-09T21:49:27+00:00	yes	2020-05-09T21:51:24+00:00	yes	Microsoft

```
#Collecting 5,000 Phishing URLs randomly
phishurl = data0.sample(n = 5000, random_state = 12).copy()
phishurl = phishurl.reset_index(drop=True)
phishurl.head()
```

	phish_id	url	phish_detail_url	submission_time	verified	verification_time	online	target
0	6514946	http://confirmprofileaccount.com/	http://www.phishtank.com/phish_detail.php?phis...	2020-04-19T11:06:55+00:00	yes	2020-04-19T13:42:41+00:00	yes	Other
1	4927651	http://www.marreme.com/MasterAdmin/04mop.html	http://www.phishtank.com/phish_detail.php?phis...	2017-04-04T19:35:54+00:00	yes	2017-05-03T23:00:42+00:00	yes	Other
2	5116976	http://modsecpaststudents.com/review/	http://www.phishtank.com/phish_detail.php?phis...	2017-07-25T18:48:30+00:00	yes	2017-07-28T16:01:36+00:00	yes	Other
3	6356131	https://docs.google.com/forms/d/e/1FAIpQLScL6L...	http://www.phishtank.com/phish_detail.php?phis...	2020-01-13T20:13:37+00:00	yes	2020-01-17T01:55:38+00:00	yes	Other
4	6535965	https://oportunidadedasemana.com/americanas//?...	http://www.phishtank.com/phish_detail.php?phis...	2020-04-29T00:01:03+00:00	yes	2020-05-01T10:55:35+00:00	yes	Other

```
#Collecting 5,000 Legitimate URLs randomly
legiurl = data1.sample(n = 5000, random_state = 12).copy()
legiurl = legiurl.reset_index(drop=True)
legiurl.head()
```

ADDRESS BAR BASED FEATURES

DOMAIN OF URL

```
# 1.Domain of the URL (Domain)
def getDomain(url):
  domain = urlparse(url).netloc
  if re.match(r"^www.",domain):
           domain = domain.replace("www.","")
  return domain
```

IP OF URL

```
# 2.Checks for IP address in URL (Have_IP)
def havingIP(url):
  try:
    ipaddress.ip_address(url)
    ip = 1
  except:
    ip = 0
  return ip
```

@ SYMBO IN URL

```
# 3.Checks the presence of @ in URL (Have_At)
def haveAtSign(url):
  if "@" in url:
    at = 1
  else:
    at = 0
  return at
```

LENGTH OF URL

```
# 4.Finding the length of URL and categorizing (URL_Length)
def getLength(url):
  if len(url) < 54:
    length = 0
  else:
    length = 1
  return length
```

DEPTH OF URL

```
# 5.Gives number of '/' in URL (URL_Depth)
def getDepth(url):
  s = urlparse(url).path.split('/')
  depth = 0
  for j in range(len(s)):
    if len(s[j]) != 0:
      depth = depth+1
  return depth
```

REDIRECTION OF URL

```
# 6.Checking for redirection '//' in the url (Redirection)
def redirection(url):
  pos = url.rfind('//')
  if pos > 6:
    if pos > 7:
      return 1
    else:
      return 0
  else:
    return 0
```

HTTP/HTTPS DOMAIN NAME

```
# 7.Existence of “HTTPS” Token in the Domain Part of the URL (https_Domain)
def httpDomain(url):
  domain = urlparse(url).netloc
  if 'https' in domain:
    return 1
  else:
    return 0
```

PREFIX/ SUFFIX “-“ IN DOMAIN

```
# 9.Checking for Prefix or Suffix Separated by (-) in the Domain (Prefix/Suffix)
def prefixSuffix(url):
    if '-' in urlparse(url).netloc:
        return 1            # phishing
    else:
        return 0            # legitimate
```

DOMAIN BASED FEATURES

AGE OF DOMAIN

```
# 13.Survival time of domain: The difference between termination time and creation time (Domain_Age)
def domainAge(domain_name):
  creation_date = domain_name.creation_date
  expiration_date = domain_name.expiration_date
  if (isinstance(creation_date,str) or isinstance(expiration_date,str)):
    try:
      creation_date = datetime.strptime(creation_date,'%Y-%m-%d')
      expiration_date = datetime.strptime(expiration_date,"%Y-%m-%d")
    except:
      return 1
  if ((expiration_date is None) or (creation_date is None)):
      return 1
  elif ((type(expiration_date) is list) or (type(creation_date) is list)):
      return 1
  else:
    ageofdomain = abs((expiration_date - creation_date).days)
    if ((ageofdomain/30) < 6):
      age = 1
    else:
      age = 0
  return age
```

END PERIOD OF DOMAIN

```
# 14.End time of domain: The difference between termination time and current time (Domain_End)
def domainEnd(domain_name):
  expiration_date = domain_name.expiration_date
  if isinstance(expiration_date,str):
    try:
      expiration_date = datetime.strptime(expiration_date,"%Y-%m-%d")
    except:
      return 1
  if (expiration_date is None):
      return 1
  elif (type(expiration_date) is list):
      return 1
  else:
    today = datetime.now()
    end = abs((expiration_date - today).days)
    if ((end/30) < 6):
      end = 0
    else:
      end = 1
  return end
```

C. HTML AND JAVASCRIPT BASED FEATURES

IFRAME REDIRECTION

```
# 15. IFrame Redirection (iFrame)
def iframe(response):
  if response == "":
      return 1
  else:
      if re.findall(r"[<iframe>|<frameBorder>]", response.text):
          return 0
      else:
          return 1
```

STATUS BAR CUSTOMIZATION

```
# 16.Checks the effect of mouse over on status bar (Mouse_Over)
def mouseOver(response):
  if response == "" :
    return 1
  else:
    if re.findall("<script>.+onmouseover.+</script>", response.text):
      return 1
    else:
      return 0
```

DISABLING RIGHT CLICK

```
# 17.Checks the status of the right click attribute (Right_Click)
def rightClick(response):
  if response == "":
    return 1
  else:
    if re.findall(r"event.button ?== ?2", response.text):
      return 0
    else:
      return 1
```

WEBSITE FORWORDING

```
# 18.Checks the number of forwardings (Web_Forwards)
def forwarding(response):
  if response == "":
    return 1
  else:
    if len(response.history) <= 2:
      return 0
    else:
      return 1
```

COMPUTING URL FEATURES

```python
#Function to extract features
def featureExtraction(url,label):

  features = []
  #Address bar based features (10)
  features.append(getDomain(url))
  features.append(havingIP(url))
  features.append(haveAtSign(url))
  features.append(getLength(url))
  features.append(getDepth(url))
  features.append(redirection(url))
  features.append(httpDomain(url))
  features.append(tinyURL(url))
  features.append(prefixSuffix(url))

  #Domain based features (4)
  dns = 0
  try:
    domain_name = whois.whois(urlparse(url).netloc)
  except:
    dns = 1

  features.append(dns)
  features.append(web_traffic(url))
  features.append(1 if dns == 1 else domainAge(domain_name))
  features.append(1 if dns == 1 else domainEnd(domain_name))

  # HTML & Javascript based features (4)
  try:
    response = requests.get(url)
  except:
    response = ""
  features.append(iframe(response))
  features.append(mouseOver(response))
  features.append(rightClick(response))
  features.append(forwarding(response))
  features.append(label)

  return features
```

FINAL DATASET AFTER FEATURE EXTRACTION:

```
#Concatenating the dataframes into one
urldata = pd.concat([legitimate, phishing]).reset_index(drop=True)
urldata.head()
```

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

II. PHISHING WEBSITE DETECTION BY ML TECHNIQUES

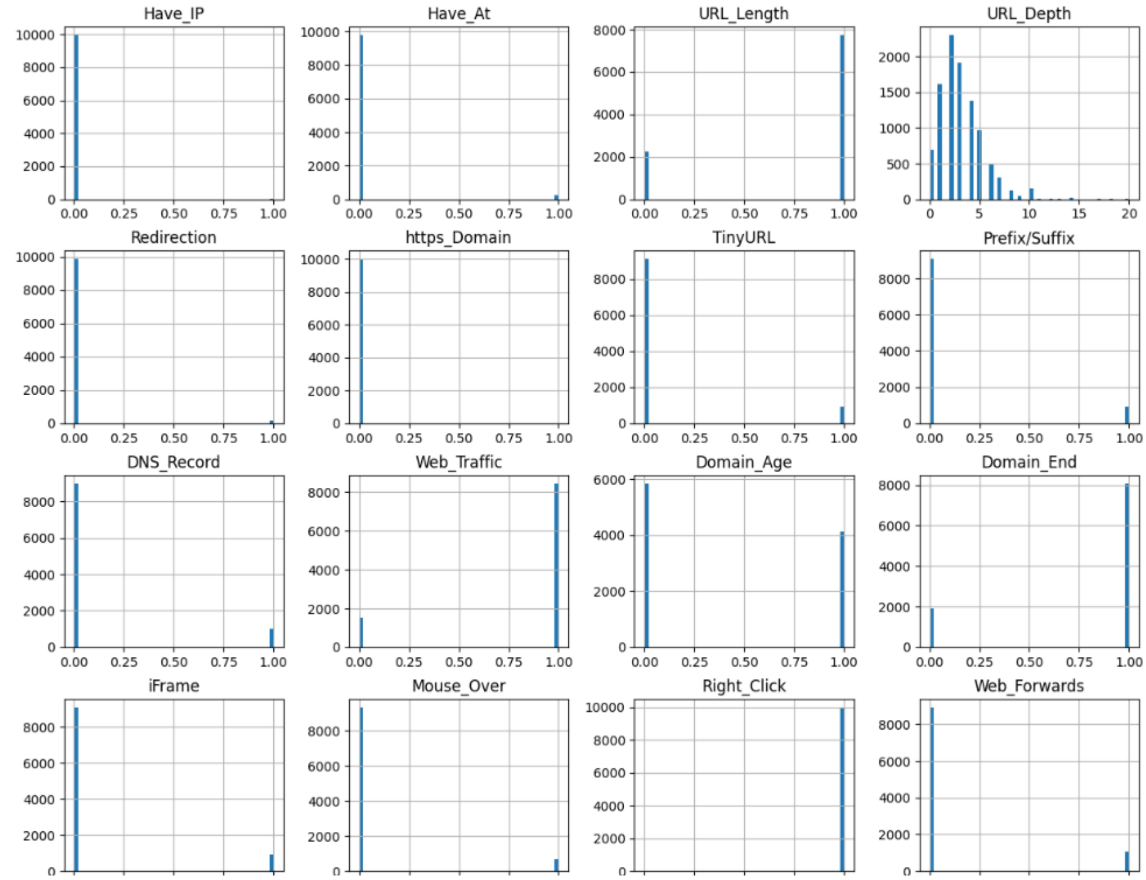
Now we will use the Data set which we got after extracting the features to train our ML models

```
#Loading the data
data0 = pd.read_csv('urldata.csv')
data0.head()
```

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

DATA VISUALIZING

```
#Plotting the data distribution
data0.hist(bins = 50,figsize = (15,15))
plt.show()
```



DATA PRE-PROCESSING

```
#Dropping the Domain column
data = data0.drop(['Domain'], axis = 1).copy()
```

```
#checking the data for null or missing values
data.isnull().sum()
```

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

```
# shuffling the rows in the dataset so that when splitting the train and test set are equally distributed
data = data.sample(frac=1).reset_index(drop=True)
data.head()
```

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

SPLITTING THE DATA

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

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

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size = 0.2, random_state = 12)
X_train.shape, X_test.shape
```

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

MACHINE LEARNING MODELS AND TRANING

From the dataset above, it is clear that this is a supervised machine learning task. There are two major types of supervised machine learning problems, called classification and regression.

This data set comes under classification problem, as the input URL is classified as phishing (1) or legitimate (0). The supervised machine learning models (classification) considered to train the dataset in this notebook are:

- Decision Tree
- Random Forest
- Multilayer Perceptrons
- XGBoost
- Autoencoder Neural Network
- Support Vector Machines

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

```
# Creating holders to store the model performance results
ML_Model = []
acc_train = []
acc_test = []

#function to call for storing the results
def storeResults(model, a,b):
  ML_Model.append(model)
  acc_train.append(round(a, 3))
  acc_test.append(round(b, 3))
```

A. DECISION TREE CLASSIFIER

```
# Decision Tree model
from sklearn.tree import DecisionTreeClassifier

# instantiate the model
tree = DecisionTreeClassifier(max_depth = 5)
# fit the model
tree.fit(X_train, y_train)
```

```
#predicting the target value from the model for the samples
y_test_tree = tree.predict(X_test)
y_train_tree = tree.predict(X_train)
```

PERFORMANCE

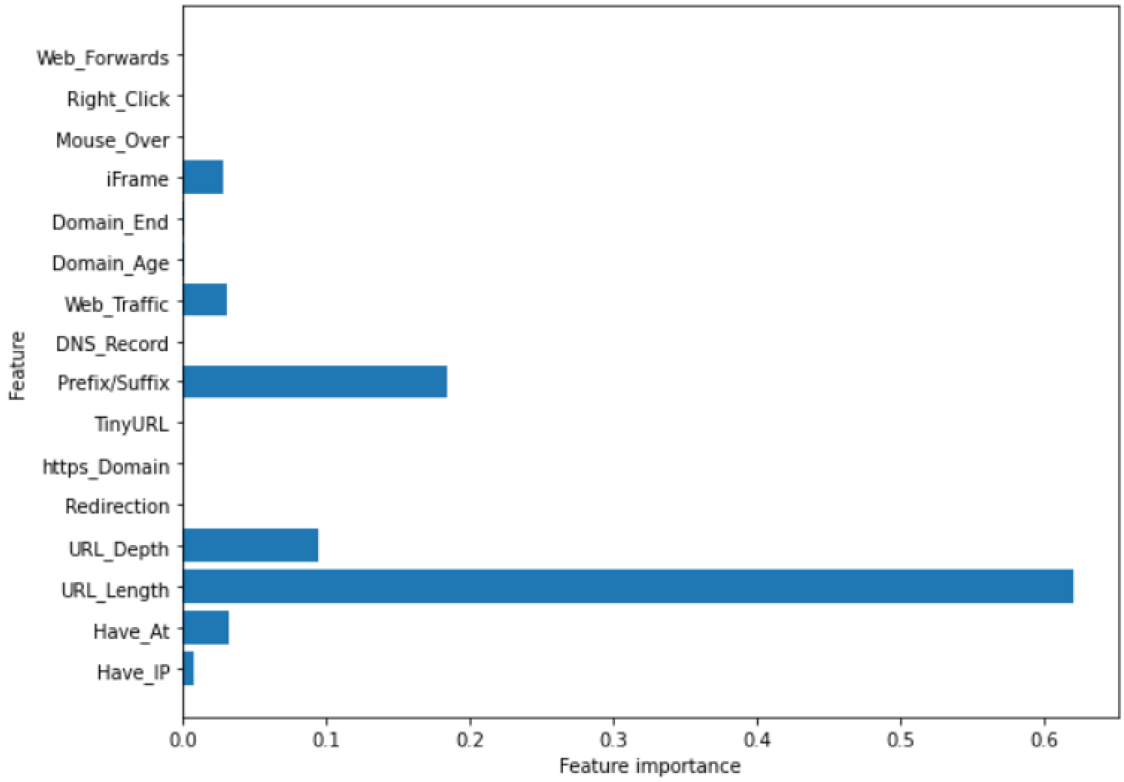
```
#computing the accuracy of the model performance
acc_train_tree = accuracy_score(y_train,y_train_tree)
acc_test_tree = accuracy_score(y_test,y_test_tree)

print("Decision Tree: Accuracy on training Data: {:.3f}".format(acc_train_tree))
print("Decision Tree: Accuracy on test Data: {:.3f}".format(acc_test_tree))
```

```
Decision Tree: Accuracy on training Data: 0.813
Decision Tree: Accuracy on test Data: 0.813
```

FEATURE IMPORTANCE

```
#checking the feature improtance in the model
plt.figure(figsize=(9,7))
n_features = X_train.shape[1]
plt.barh(range(n_features), tree.feature_importances_, align='center')
plt.yticks(np.arange(n_features), X_train.columns)
plt.xlabel("Feature importance")
plt.ylabel("Feature")
plt.show()
```



B. RANDOM FOREST

```
# Random Forest model
from sklearn.ensemble import RandomForestClassifier

# instantiate the model
forest = RandomForestClassifier(max_depth=5)

# fit the model
forest.fit(X_train, y_train)
```

PERFORMANCE:

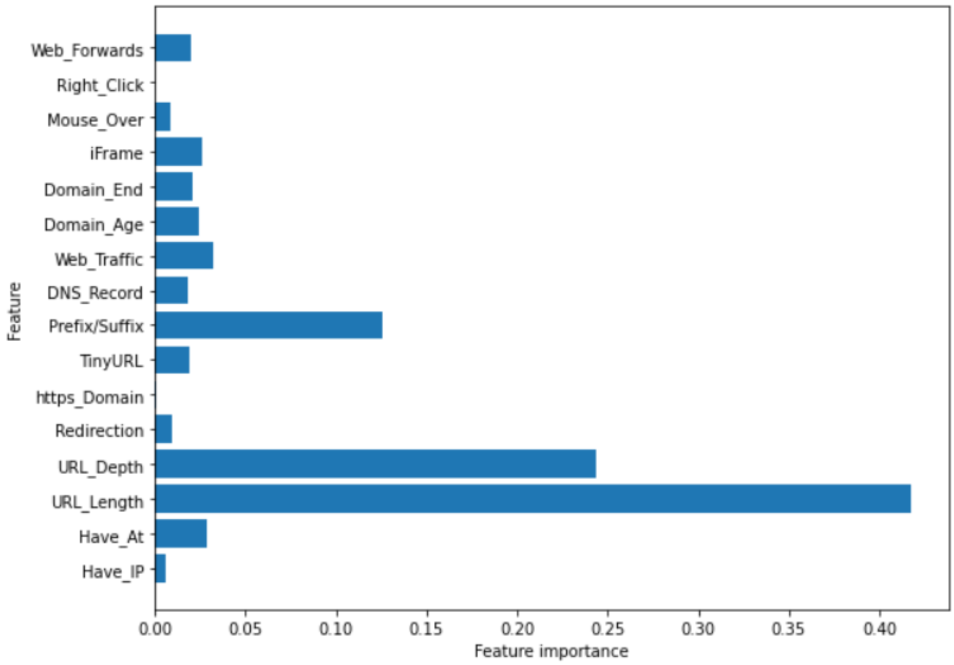
```
#computing the accuracy of the model performance
acc_train_forest = accuracy_score(y_train,y_train_forest)
acc_test_forest = accuracy_score(y_test,y_test_forest)

print("Random forest: Accuracy on training Data: {:.3f}".format(acc_train_forest))
print("Random forest: Accuracy on test Data: {:.3f}".format(acc_test_forest))
```

```
Random forest: Accuracy on training Data: 0.814
Random forest: Accuracy on test Data: 0.817
```

FEATURE IMPORTANCE

```
#checking the feature improtance in the model
plt.figure(figsize=(9,7))
n_features = X_train.shape[1]
plt.barh(range(n_features), forest.feature_importances_, align='center')
plt.yticks(np.arange(n_features), X_train.columns)
plt.xlabel("Feature importance")
plt.ylabel("Feature")
plt.show()
```



C. MULTILAYER PERCEPTONS

```python
# Multilayer Perceptrons model
from sklearn.neural_network import MLPClassifier

# instantiate the model
mlp = MLPClassifier(alpha=0.001, hidden_layer_sizes=([100,100,100]))

# fit the model
mlp.fit(X_train, y_train)
```

PERFORMANCE

```python
#computing the accuracy of the model performance
acc_train_mlp = accuracy_score(y_train,y_train_mlp)
acc_test_mlp = accuracy_score(y_test,y_test_mlp)

print("Multilayer Perceptrons: Accuracy on training Data: {:.3f}".format(acc_train_mlp))
print("Multilayer Perceptrons: Accuracy on test Data: {:.3f}".format(acc_test_mlp))
```

```
Multilayer Perceptrons: Accuracy on training Data: 0.865
Multilayer Perceptrons: Accuracy on test Data: 0.862
```

D. XGBOOST CLASSIFER

```python
#XGBoost Classification model
from xgboost import XGBClassifier

# instantiate the model
xgb = XGBClassifier(learning_rate=0.4,max_depth=7)
#fit the model
xgb.fit(X_train, y_train)
```

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.4, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=7, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              n_estimators=100, n_jobs=None, num_parallel_tree=None,
              predictor=None, random_state=None, ...)
```

PERFORMANCE

```python
#computing the accuracy of the model performance
acc_train_xgb = accuracy_score(y_train,y_train_xgb)
acc_test_xgb = accuracy_score(y_test,y_test_xgb)

print("XGBoost: Accuracy on training Data: {:.3f}".format(acc_train_xgb))
print("XGBoost : Accuracy on test Data: {:.3f}".format(acc_test_xgb))
```

```
XGBoost: Accuracy on training Data: 0.867
XGBoost : Accuracy on test Data: 0.863
```

E. AUTOENCODER NEURAL NETWORK

```
#building autoencoder model

input_dim = X_train.shape[1]
encoding_dim = input_dim

input_layer = Input(shape=(input_dim, ))
encoder = Dense(encoding_dim, activation="relu",
                activity_regularizer=regularizers.l1(10e-4))(input_layer)
encoder = Dense(int(encoding_dim), activation="relu")(encoder)

encoder = Dense(int(encoding_dim-2), activation="relu")(encoder)
code = Dense(int(encoding_dim-4), activation='relu')(encoder)
decoder = Dense(int(encoding_dim-2), activation='relu')(code)

decoder = Dense(int(encoding_dim), activation='relu')(encoder)
decoder = Dense(input_dim, activation='relu')(decoder)
autoencoder = Model(inputs=input_layer, outputs=decoder)
autoencoder.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 16)]              0

 dense (Dense)               (None, 16)                272

 dense_1 (Dense)             (None, 16)                272

 dense_2 (Dense)             (None, 14)                238

 dense_5 (Dense)             (None, 16)                240

 dense_6 (Dense)             (None, 16)                272

=================================================================
Total params: 1,294
Trainable params: 1,294
Non-trainable params: 0
_________________________________________________________________
```

COMPILING AND TRANING MODEL

```
#compiling the model
autoencoder.compile(optimizer='adam',
                    loss='binary_crossentropy',
                    metrics=['accuracy'])

#Training the model
history = autoencoder.fit(X_train, X_train, epochs=10, batch_size=64, shuffle=True, validation_split=0.2)
```

PERFORMANCE

```
acc_train_auto = autoencoder.evaluate(X_train, X_train)[1]
acc_test_auto = autoencoder.evaluate(X_test, X_test)[1]

print('\nAutoencoder: Accuracy on training Data: {:.3f}' .format(acc_train_auto))
print('Autoencoder: Accuracy on test Data: {:.3f}' .format(acc_test_auto))
```

```
250/250 [==============================] - 1s 2ms/step - loss: -1.6582 - accuracy: 0.8050
63/63 [==============================] - 0s 2ms/step - loss: -1.6795 - accuracy: 0.8030

Autoencoder: Accuracy on training Data: 0.805
Autoencoder: Accuracy on test Data: 0.803
```

F. SUPPORT VECTOE MACHINES

```
#Support vector machine model
from sklearn.svm import SVC

# instantiate the model
svm = SVC(kernel='linear', C=1.0, random_state=12)
#fit the model
svm.fit(X_train, y_train)
```

PERFORMANCE

```
#computing the accuracy of the model performance
acc_train_svm = accuracy_score(y_train,y_train_svm)
acc_test_svm = accuracy_score(y_test,y_test_svm)

print("SVM: Accuracy on training Data: {:.3f}".format(acc_train_svm))
print("SVM : Accuracy on test Data: {:.3f}".format(acc_test_svm))
```

```
SVM: Accuracy on training Data: 0.800
SVM : Accuracy on test Data: 0.808
```

RESULTS AND ANALYSIS

COMPARISION OF MODELS

```
#creating dataframe
results = pd.DataFrame({ 'ML Model': ML_Model,
    'Train Accuracy': acc_train,
    'Test Accuracy': acc_test})
results
```

	ML Model	Train Accuracy	Test Accuracy
0	Decision Tree	0.813	0.813
1	Random Forest	0.814	0.817
2	Multilayer Perceptrons	0.865	0.862
3	XGBoost	0.867	0.863
4	AutoEncoder	0.805	0.803
5	SVM	0.800	0.808

SORTING THEM ACCORDING TO TEST ND TRAIN ACCURAY

```
results.sort_values(by=['Test Accuracy', 'Train Accuracy'], ascending=False)
```

	ML Model	Train Accuracy	Test Accuracy
3	XGBoost	0.867	0.863
2	Multilayer Perceptrons	0.865	0.862
1	Random Forest	0.814	0.817
0	Decision Tree	0.813	0.813
5	SVM	0.800	0.808
4	AutoEncoder	0.805	0.803

In the comprehensive results and analysis, a systematic evaluation of machine learning models was conducted, focusing on both training and test accuracies. Notably, XGBoost emerged as the standout performer in accurately detecting phishing websites, showcasing its robustness for real-world cybersecurity applications. The comparative analysis offered a nuanced understanding of each model's strengths and limitations. Upon sorting and ranking based on accuracies, XGBoost secured the top position, further emphasizing its effectiveness. In conclusion, the results highlight XGBoost as a compelling choice, underscoring its capability to differentiate between phishing and legitimate websites with exceptional accuracy, thereby reinforcing its significance in the realm of cybersecurity.

CONCLUSION AND FUTURE WORK:

Undertaking this project has provided valuable insights into the complex landscape of phishing detection. Distinguishing phishing websites from legitimate ones was approached innovatively by integrating machine learning and deep neural networks. The project's primary objective was to predict and preemptively identify phishing websites, achieved by extracting relevant features from a diverse dataset that encompassed both legitimate and phishing URLs. Rigorous evaluation of supervised machine learning models highlighted XGBoost as the standout performer, emphasizing its effectiveness in addressing the challenge of phishing detection.

Looking ahead, the project opens avenues for impactful future work. The development of user-centric applications, such as browser extensions or graphical user interfaces (GUIs), equipped with the trained machine learning model, could empower users to assess URL legitimacy in real-time. Bridging the gap between advanced machine learning techniques and everyday users, these applications could serve as educational tools, raising awareness and fostering a safer online environment. Moreover, future enhancements could focus on improving model explainability, exploring real-time monitoring strategies, and ensuring continuous updates and collaboration with industry experts for ongoing refinement and adaptability against emerging phishing techniques. The success of XGBoost suggests its potential for broader applications in real-world cybersecurity contexts, emphasizing the significance of proactive cybersecurity measures.

In summary, this project not only contributes to fortifying cybersecurity defenses but also provides a foundation for ongoing research and development in the dynamic field of phishing mitigation.