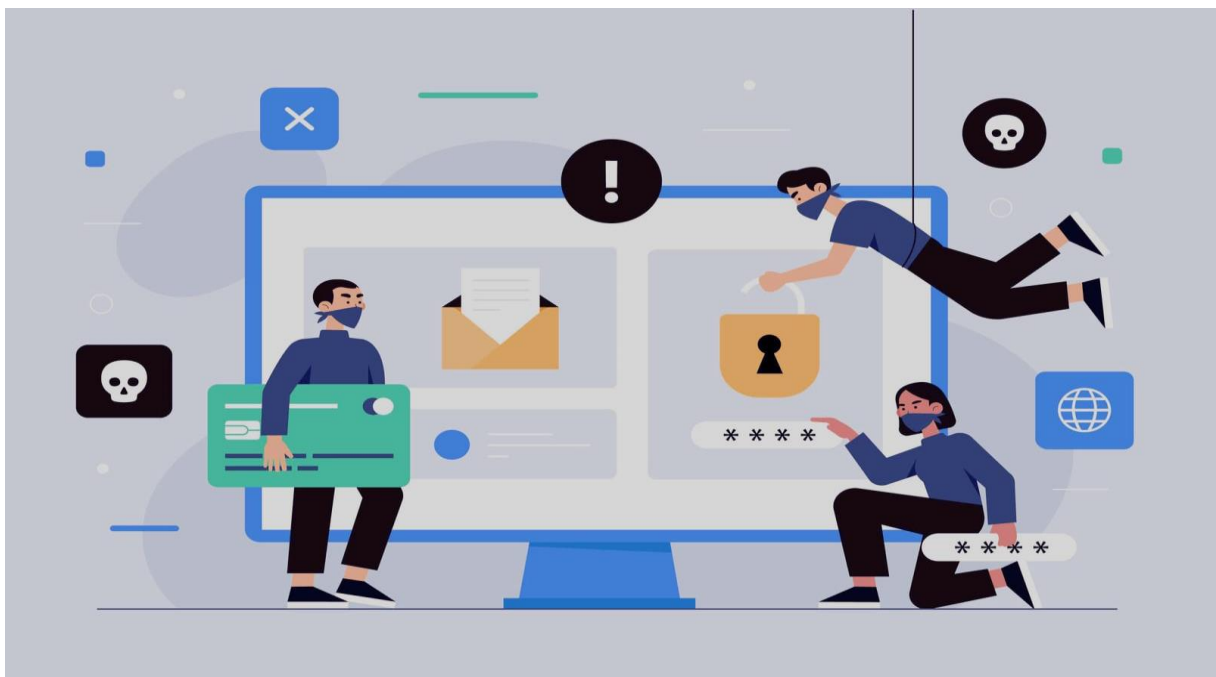




URL-Based Phishing Detection Using Machine Learning



Team Members

B.Sumanjali(20481A1221) (Team Lead)

A. Jahnavi (20481A1205)

Ch. Sushma (20481A1227)

D. Srivalli (20481A1237)

Table of Contents

	Topic	Page No.
1	Introduction	1
	1.1 Overview	1
	1.2 Purpose	1
2	Literature Survey	2
	2.1 Existing problem	2
	2.2 Proposed solution	2
3	Theoretical Analysis	3
	3.1 Block diagram	3
	3.2 Software designing	3
4	Experimental Investigations	4-18
5	Flowchart	19
6	Result	20-21
7	Advantages & Disadvantages	22
8	Applications	23
9	Conclusion	23
10	Future Scope	24
11	Biblography	24
	11.1 Source Code	25-26

1. INTRODUCTION

1.1 Overview: -

There are a number of users who purchase products online and make payments through e-banking. There are e-banking websites that ask users to provide sensitive data such as username, password & credit card details, etc., often for malicious reasons. This type of e-banking website is known as a phishing website. Web service is one of the key communications software services for the Internet. Web phishing is one of many security threats to web services on the Internet.

Common threats of web phishing:

- Web phishing aims to steal private information, such as usernames, passwords, and credit card details, by way of impersonating a legitimate entity.
- It will lead to information disclosure and property damage.
- Large organizations may get trapped in different kinds of scams.

1.2 Purpose: -

This Guided Project mainly focuses on applying a machine-learning algorithm to detect Phishing websites. In order to detect and predict e-banking phishing websites, we proposed an intelligent, flexible and effective system that is based on using classification algorithms. We implemented classification algorithms and techniques to extract the phishing datasets criteria to classify their legitimacy. The e-banking phishing website can be detected based on some important characteristics like URL and domain identity, and security and encryption criteria in the final phishing detection rate. Once a user makes a transaction online when he makes payment through an e-banking website our system will use a data mining algorithm to detect whether the e-banking website is a phishing website or not.

2. LITERATURE SURVEY

2.1 Existing problem: -

Imbalanced Datasets: The dataset used for training ML models may be imbalanced, meaning there might be significantly more legitimate URLs than phishing URLs, affecting the model's ability to accurately detect phishing attempts.

Data Privacy Concerns: The analysis of URLs may inadvertently reveal sensitive information or violate user privacy, raising ethical and legal concerns.

Scalability: As the volume of URLs increases, the computational resources and time required for processing and analysis can become a challenge.

2.2 Proposed solution: -

Feature Engineering: Extract relevant features from URLs, such as domain reputation, URL length, presence of certain keywords, and domain age. These features can provide valuable information for ML models to distinguish between legitimate and phishing URLs.

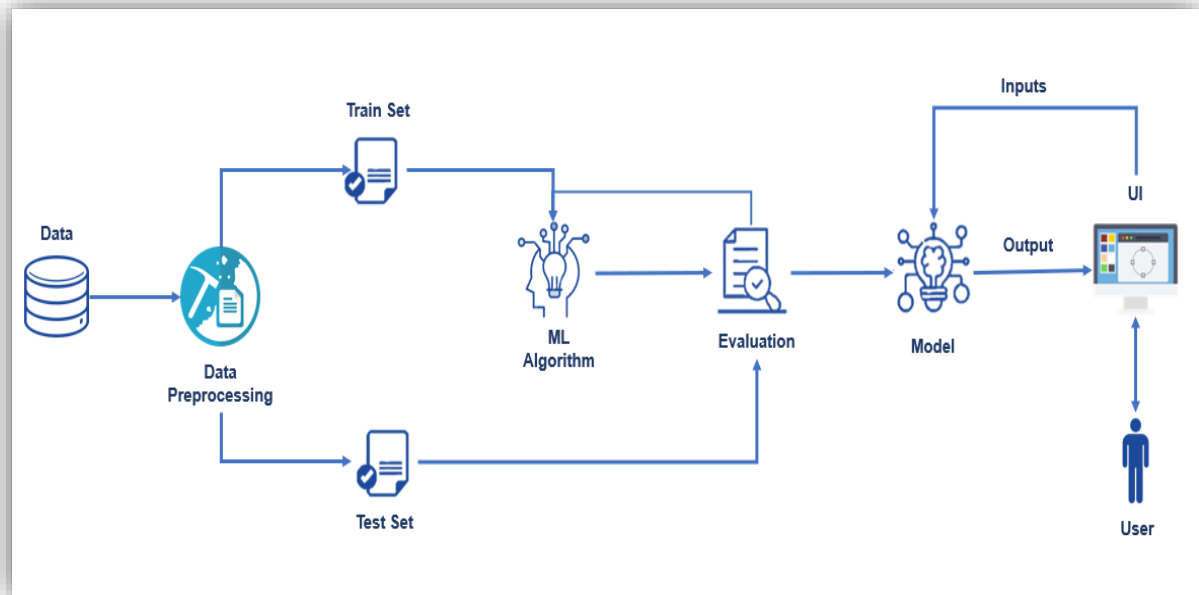
Ensemble Learning: Combine multiple ML models, such as Random Forests, Gradient Boosting, and Decision tree classification in an ensemble to leverage their strengths and mitigate individual weaknesses, improving overall detection performance.

Imbalanced Data Handling: Address imbalanced datasets by using techniques like oversampling, undersampling, or using synthetic data generation methods like SMOTE (Synthetic Minority Over-sampling Technique) to balance the proportion of legitimate and phishing URLs in the training data.

Transfer Learning: Utilize pre-trained models on large-scale datasets related to web data or URLs and fine-tune them with domain-specific phishing data to benefit from the knowledge and features learned from broader contexts.

3. THEORETICAL ANALYSIS

3.1 Block diagram



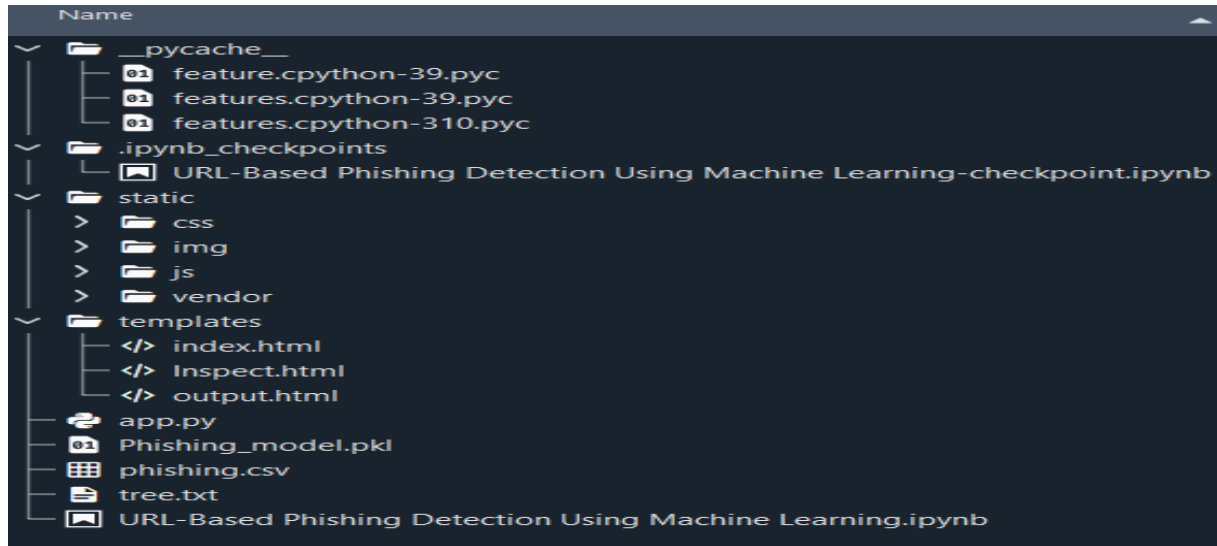
3.2 Software designing: -

To complete this project, you must required following software's, concepts and packages

- **Anaconda navigator:**
 - Refer the link below to download anaconda navigator.
 - Link : <https://youtu.be/1ra4zH2G4o0>
- **Python packages:**
 - Open anaconda prompt as administrator
 - Type "pip install numpy" and click enter.
 - Type "pip install pandas" and click enter.
 - Type "pip install scikit-learn" and click enter.
 - Type "pip install matplotlib" and click enter.
 - Type "pip install scipy" and click enter.
 - Type "pip install pickle-mixin" and click enter.
 - Type "pip install seaborn" and click enter.
 - Type "pip install Flask" and click enter.

4. EXPERIMENTAL INVESTIGATIONS

Create the Project folder which contains files as shown below



We are building a flask application which needs HTML pages stored in the templates folder.

➤ Milestone 1: Data Collection & Data Pre-processing

Activity 1: Importing Required Libraries:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from sklearn import metrics
import warnings
warnings.filterwarnings('ignore')
```

Collection Of Dataset

To start with, we have to select or identify a dataset that contains a set of features through which a phishing website can be identified.

Activity 2: Download the dataset

There are many popular open sources for collecting the data. Eg: kaggle.com, UCI repository, etc.

In this project we have used phishing.csv data. This data is downloaded from kaggle.com. Please refer the link given below to download the dataset.

Dataset Link: <https://www.kaggle.com/eswarchandt/phishing-website-detector>

As we have understood how the data is collected lets pre-process the collected data.

Activity 3: Data Pre-processing

The download data set is not suitable for training the machine learning model as it might have so much of randomness so we need to clean the dataset properly in order to fetch good results.

This activity includes the following steps.

- Handling missing values
- Handling categorical data
- Handling outliers
- Scaling Techniques
- Splitting dataset into training and test set

Note: These are the general steps of pre-processing the data before using it for machine learning. Depending on the condition of your dataset, you may or may not have to go through all these steps.

Activity 4: Checking for null values

Let's find the shape of our dataset first, To find the shape of our data, df.shape method is used.

To find the data type, df.info() function is used.

```
df.shape
```

```
(11054, 32)
```

```
: df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11054 entries, 0 to 11053
Data columns (total 32 columns):
 #   Column                                Non-Null Count  Dtype
---  -
 0   Index                                11054 non-null  int64
 1   UsingIP                              11054 non-null  int64
 2   LongURL                              11054 non-null  int64
 3   ShortURL                             11054 non-null  int64
 4   Symbol@                              11054 non-null  int64
 5   Redirecting//                        11054 non-null  int64
 6   PrefixSuffix-                        11054 non-null  int64
 7   SubDomains                           11054 non-null  int64
 8   HTTPS                                11054 non-null  int64
 9   DomainRegLen                         11054 non-null  int64
10   Favicon                              11054 non-null  int64
11   NonStdPort                           11054 non-null  int64
12   HTTPSDomainURL                       11054 non-null  int64
13   RequestURL                           11054 non-null  int64
14   AnchorURL                            11054 non-null  int64
15   LinksInScriptTags                   11054 non-null  int64
16   ServerFormHandler                    11054 non-null  int64
17   InfoEmail                            11054 non-null  int64
18   AbnormalURL                          11054 non-null  int64
19   WebsiteForwarding                   11054 non-null  int64
20   StatusBarCust                        11054 non-null  int64
21   DisableRightClick                   11054 non-null  int64
22   UsingPopupWindow                    11054 non-null  int64
23   IFrameRedirection                   11054 non-null  int64
24   AgeofDomain                         11054 non-null  int64
25   DNSRecording                         11054 non-null  int64
26   WebsiteTraffic                       11054 non-null  int64
27   PageRank                            11054 non-null  int64
28   GoogleIndex                         11054 non-null  int64
29   LinksPointingToPage                 11054 non-null  int64
30   StatsReport                          11054 non-null  int64
31   class                               11054 non-null  int64
dtypes: int64(32)
memory usage: 2.7 MB
```

For checking the null values, `df.isnull()` function is used. To sum those null values we use `.sum()` function to it. From the below image we found that there are some null values present in our dataset. So we have to handle the missing values.

```
df.isnull().sum()
```

```
Index          0
UsingIP        0
LongURL        0
ShortURL       0
Symbol@        0
Redirecting//   0
PrefixSuffix-   0
SubDomains     0
HTTPS          0
DomainRegLen   0
Favicon        0
NonStdPort     0
HTTPSDomainURL 0
RequestURL     0
AnchorURL      0
LinksInScriptTags 0
ServerFormHandler 0
InfoEmail      0
AbnormalURL    0
WebsiteForwarding 0
StatusBarCust   0
DisableRightClick 0
UsingPopupWindow 0
IframeRedirection 0
AgeofDomain    0
DNSRecording   0
WebsiteTraffic 0
PageRank       0
GoogleIndex    0
LinksPointingToPage 0
StatsReport    0
class          0
dtype: int64
```

There is no null values in our dataset

There is no categorical data in our dataset.

Activiy 5: Checking for duplicated data

```
df.duplicated().sum()
```

```
5205
```

we had duplicate data in our dataset so we have to remove it.

Handling duplicate data

```
df.drop_duplicates(inplace=True)
```

```
df.duplicated().sum()
```

```
0
```

```
df.shape
```

```
(5849, 22)
```


Activity 6: Outliers

We had outliers in our dataset in the columns 'PrefixSuffix-', 'NonStdPort', 'HTTPSDomainURL', 'AnchorURL', 'ServerFormHandler', 'InfoEmail', 'AbnormalURL', 'WebsiteForwarding', 'StatusBarCust', 'DisableRightClick', 'GoogleIndex', 'StatsReport'

Handling Outliers

```
# Z-score method
z_scores = np.abs((df - df.mean()) / df.std())
outliers = df[z_scores > 3]

# IQR method
Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3 - Q1
outliers = df[(df < (Q1 - 1.5 * IQR)) | (df > (Q3 + 1.5 * IQR))]

# Handle outliers
# Remove outliers
df_cleaned = df.drop(outliers.index)

# Replace outliers with median
df_cleaned = df.copy()
df_cleaned = np.where(z_scores > 3, df.median(), df)

# Apply log transformation
df_cleaned = np.log1p(df + 0.0001)

# Validate the results
print(df_cleaned.describe())
```

Activity 7: Checking data is balanced or not?

```
clmns=['class', 'HTTPS', 'SubDomains']
for i in clmns:
    print(f"The Value counts of {i} :")
    print(df[i].value_counts().to_string(), '\n')
```

The Value counts of class :

```
-1    3019
1     2830
```

The Value counts of HTTPS :

```
1     3000
-1    2128
0       721
```

The Value counts of SubDomains :

```
1     2088
0     2058
-1    1703
```

From the above we can understand that our data set is imbalanced

	UsingIP	PrefixSuffix-	SubDomains	HTTPS	NonStdPort
count	5849.000000	5849.000000	5849.000000	5849.000000	5849.000000
mean	-3.600763	-8.023406	-2.434195	-2.995374	-0.9034
std	4.908301	3.216801	4.353017	4.705445	3.642
min	-9.210340	-9.210340	-9.210340	-9.210340	-9.210340
25%	-9.210340	-9.210340	-9.210340	-9.210340	0.693197
50%	0.693197	-9.210340	0.000100	0.693197	0.693197
75%	0.693197	-9.210340	0.693197	0.693197	0.693197
max	0.693197	0.693197	0.693197	0.693197	0.693197

	HTTPSDomainURL	RequestURL	AnchorURL	LinksInScriptTags	\
count	5849.000000	5849.000000	5849.000000	5849.000000	5849.000000
mean	-1.022016	-3.660025	-3.011206	-3.341746	-3.341746
std	3.747949	4.915881	4.479933	4.606370	4.606370
min	-9.210340	-9.210340	-9.210340	-9.210340	-9.210340
25%	0.693197	-9.210340	-9.210340	-9.210340	-9.210340
50%	0.693197	0.693197	0.000100	0.693197	0.000100
75%	0.693197	0.693197	0.000100	0.693197	0.000100
max	0.693197	0.693197	0.693197	0.693197	0.693197

	ServerFormHandler	WebsiteForwarding	StatusBarCust	\
count	5849.000000	5849.000000	5849.000000	5849.000000
mean	-6.775943	0.091462	-0.678296	-0.678296
std	4.197679	0.234489	3.421059	3.421059
min	-9.210340	0.000100	-9.210340	-9.210340
25%	-9.210340	0.000100	0.693197	0.693197
50%	-9.210340	0.000100	0.693197	0.693197
75%	0.000100	0.000100	0.693197	0.693197
max	0.693197	0.693197	0.693197	0.693197

	DisableRightClick	AgeofDomain	DNSRecording	WebsiteTraffic	\
count	5849.000000	5849.000000	5849.000000	5849.000000	5849.000000
mean	0.207248	-4.152746	-1.905868	-1.979872	-1.979872
std	2.139451	4.951061	4.357527	4.201293	4.201293
min	-9.210340	-9.210340	-9.210340	-9.210340	-9.210340
25%	0.693197	-9.210340	-9.210340	-9.210340	-9.210340
50%	0.693197	0.693197	0.693197	0.000100	0.000100
75%	0.693197	0.693197	0.693197	0.693197	0.693197
max	0.693197	0.693197	0.693197	0.693197	0.693197

	PageRank	GoogleIndex	StatsReport	class
count	5849.000000	5849.000000	5849.000000	5849.000000
mean	-6.357295	-0.981379	-0.806980	-4.418579
std	4.485400	3.712463	3.550871	4.949606
min	-9.210340	-9.210340	-9.210340	-9.210340
25%	-9.210340	0.693197	0.693197	-9.210340
50%	-9.210340	0.693197	0.693197	-9.210340
75%	0.693197	0.693197	0.693197	0.693197
max	0.693197	0.693197	0.693197	0.693197

[8 rows x 22 columns]

Activity 8: Scaling

Splitting the data

The data is split into train & test sets, 80-20 split.

```
X = df.drop(["class"],axis =1)
y = df["class"]
```

Scaling

```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
clmn_names=X.columns
scaled=scaler.fit_transform(X)
features=pd.DataFrame(scaled,columns=clmn_names)
features
```

	UsingIP	PrefixSuffix-	SubDomains	HTTPS	NonStdPort	HTTPSDomainURL	RequestURL	AnchorURL	LinksInScriptTags	ServerFormHandler	...	Abn
0	0.874911	-0.369011	-0.082035	0.920511	0.438422	-2.184935	0.885618	0.190783	-1.108454	-0.550407
1	0.874911	-0.369011	-1.328329	-1.243069	0.438422	-2.184935	0.885618	0.190783	-1.108454	-0.550407
2	0.874911	-0.369011	-1.328329	-1.243069	0.438422	-2.184935	-1.129155	0.190783	0.190814	-0.550407
3	0.874911	-0.369011	1.164259	0.920511	0.438422	0.457680	0.885618	0.190783	0.190814	-0.550407
4	-1.142973	-0.369011	1.164259	0.920511	0.438422	-2.184935	0.885618	0.190783	0.190814	-0.550407
...
5844	0.874911	-0.369011	-1.328329	-1.243069	0.438422	-2.184935	0.885618	-1.186860	1.490083	-0.550407
5845	0.874911	-0.369011	1.164259	-1.243069	0.438422	0.457680	0.885618	0.190783	1.490083	-0.550407
5846	0.874911	-0.369011	-1.328329	0.920511	0.438422	0.457680	-1.129155	0.190783	0.190814	-0.550407
5847	-1.142973	-0.369011	1.164259	-1.243069	0.438422	-2.184935	-1.129155	0.190783	-1.108454	-0.550407
5848	-1.142973	-0.369011	-1.328329	-1.243069	0.438422	0.457680	-1.129155	-1.186860	0.190814	-0.550407

5849 rows × 21 columns

Activity 9: Train Test and Split

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(features, y, test_size = 0.2, random_state = 42)
X_train.shape, y_train.shape, X_test.shape, y_test.shape
```

```
((4679, 21), (4679,)), (1170, 21), (1170,))
```

Activity 10: Handling Balance Data

```
from imblearn.over_sampling import SMOTE
smote=SMOTE(sampling_strategy=1)
X_train,y_train=smote.fit_resample(X_train,y_train)
pd.DataFrame(y_train).value_counts()
```

```
class
-1      2388
1      2388
dtype: int64
```

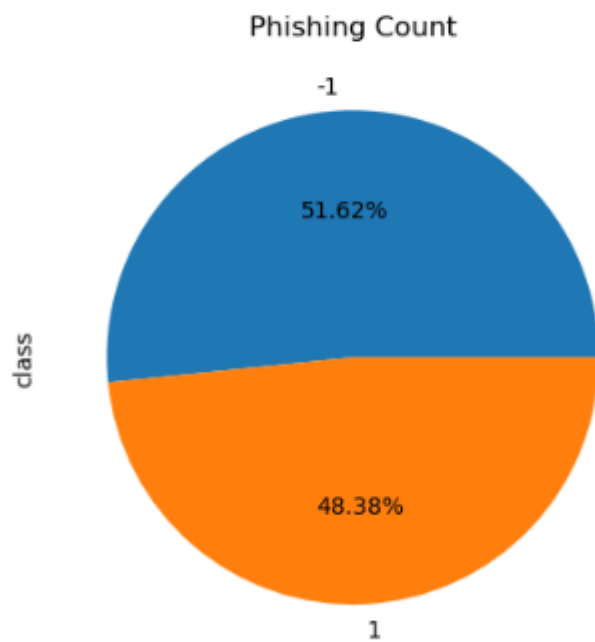
From the above we understand that our data is balanced.

➤ Milestone 2: Visualizing and analysing the data

Activity 1: Univariate analysis

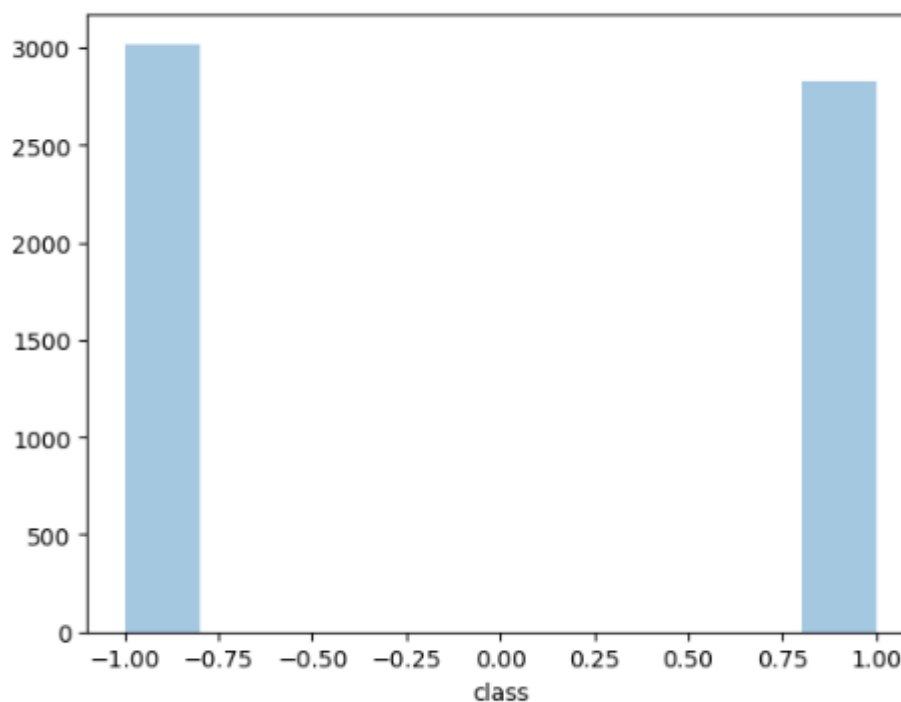
In simple words, univariate analysis is understanding the data with single feature. Here we have displayed two different graphs such as distplot and countplot.

```
df['class'].value_counts().plot(kind='pie', autopct='%1.2f%%')
plt.title("Phishing Count")
plt.show()
```



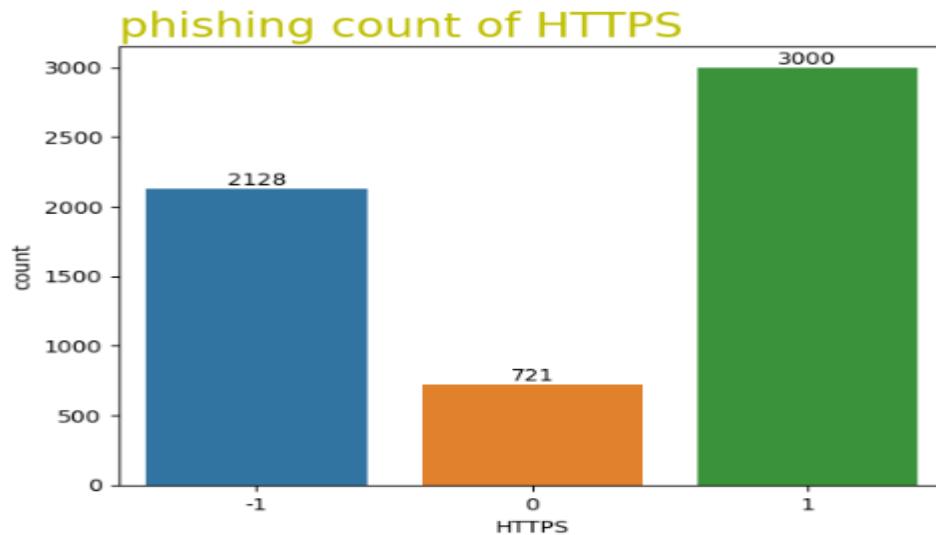
- From the above plot we came to know, the highest distribution of phishing is unsafe with 51.62%

```
sns.distplot( a=df["class"], hist=True, kde=False, rug=False )
<Axes: xlabel='class'>
```



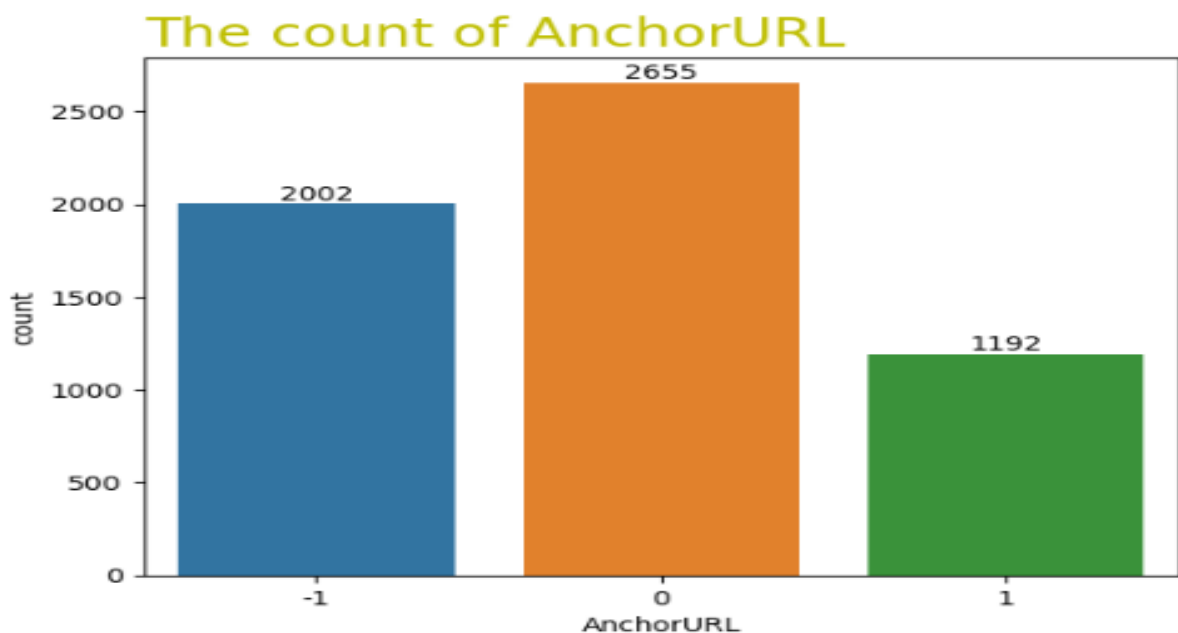
Phishing count is more here also as the class which has 1 is greater than the class which has -1

```
count=sns.countplot(x=df.HTTPS,data=df)
for i in count.containers:
    count.bar_label(i)
plt.title("phishing count of HTTPS",color='y',size=20,loc='left')
plt.show()
```



- phishing URLs are 2128.
- legitimate HTTPS URLs or non_phishing URLs are 3000.
- Suspicious URLs are 721 (0 indicates a potential risk of features that indicate a potential risk of phishing, but they are not confirmed phishing URLs).

```
count=sns.countplot(x=df.AnchorURL,data=df)
for i in count.containers:
    count.bar_label(i)
plt.title("The count of AnchorURL ",color='y',size=20,loc='left')
plt.show()
```

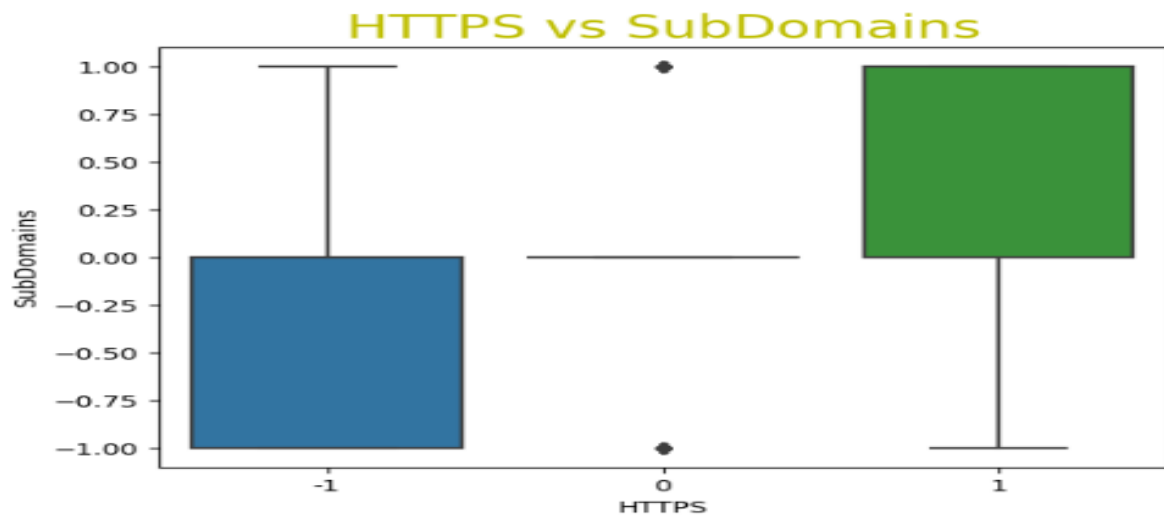


In the AnchorURL suspicious URLs are more than the Non-phishing and phishing URLs

Activity 2: Bivariate analysis

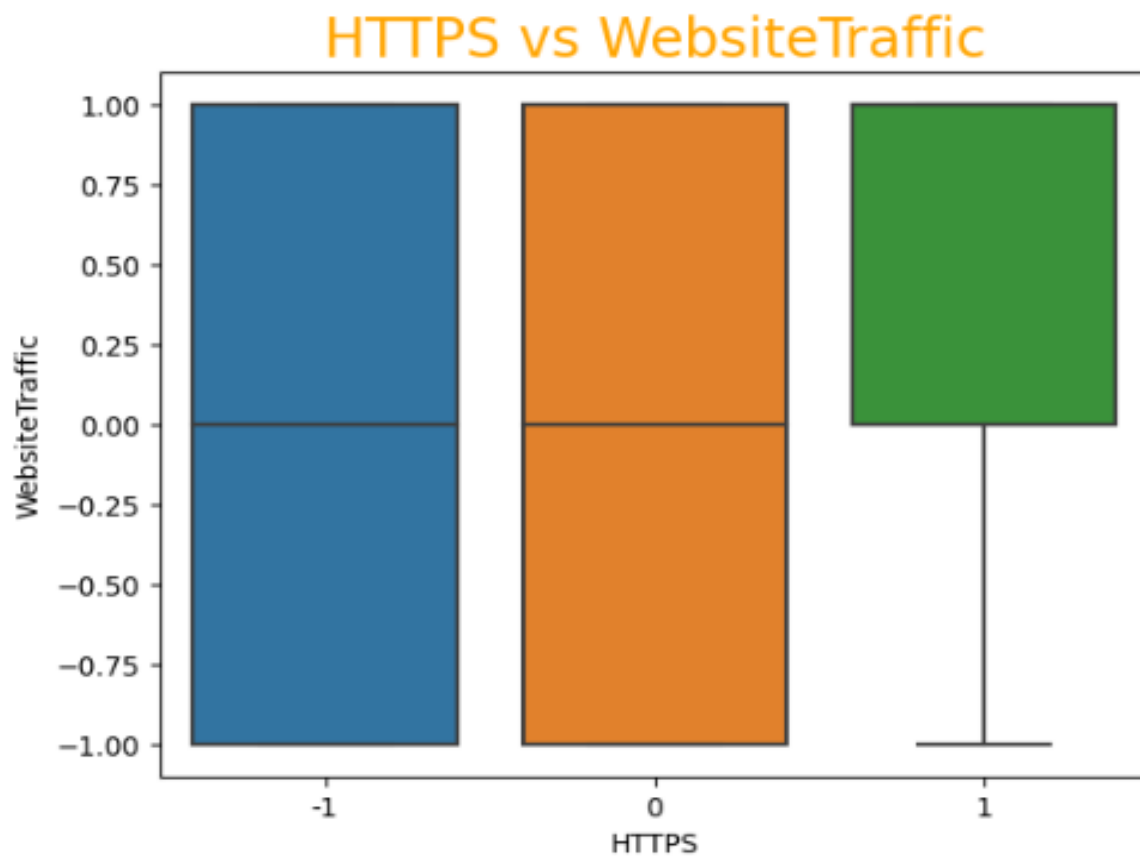
To find the relation between two features we use bivariate analysis.

```
sns.boxplot(x=df.HTTPS,y=df.SubDomains)
plt.title("HTTPS vs SubDomains",color='y',size=20)
plt.show()
```



There are outliers in my dataset

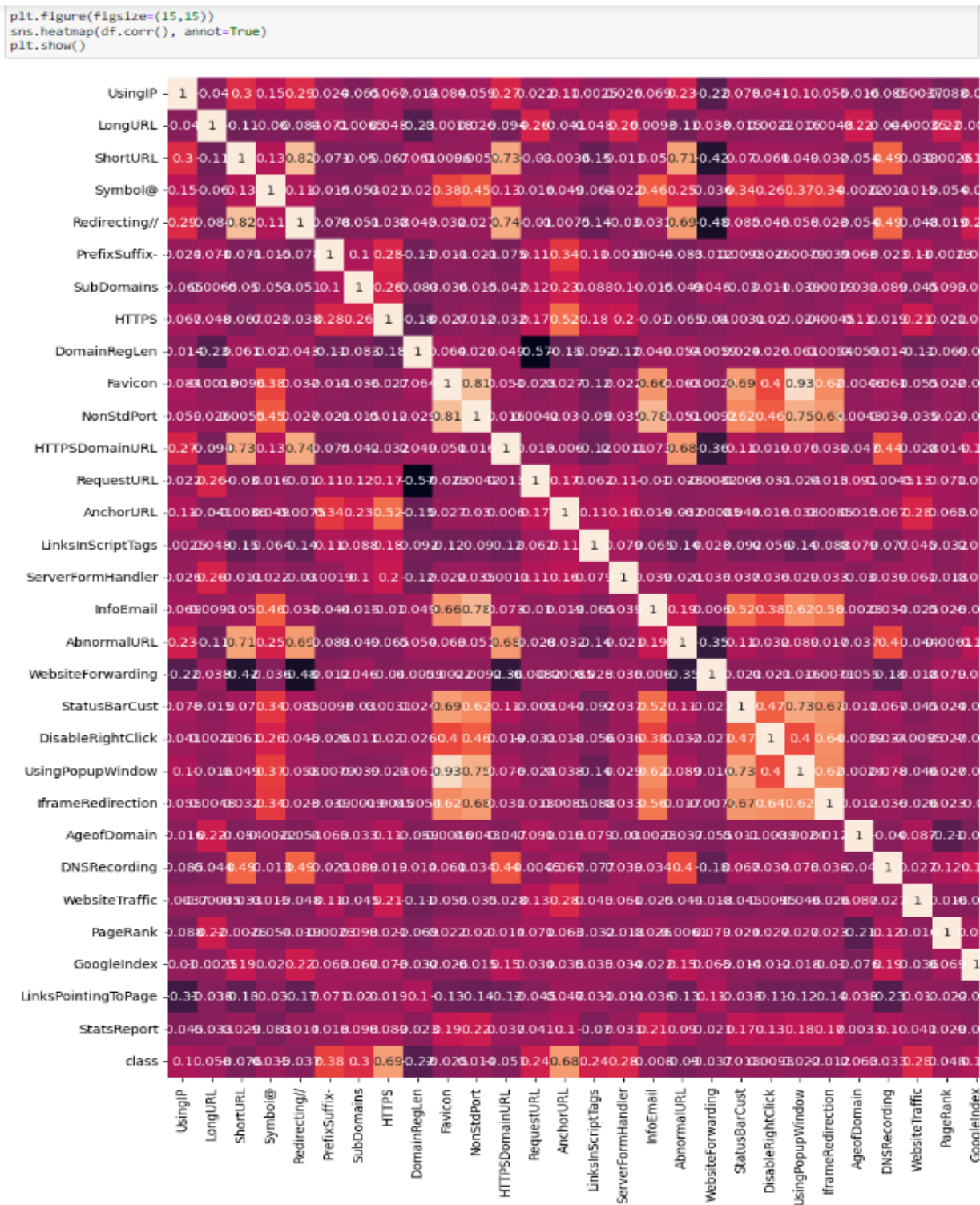
```
sns.boxplot(x=df.HTTPS,y=df.WebsiteTraffic)
plt.title("HTTPS vs WebsiteTraffic",color='orange',size=20)
plt.show()
```



Activity 3: Multivariate analysis

In simple words, multivariate analysis is to find the relation between multiple features. Here we have used heatmap from seaborn package.

- From the below image, we came to a conclusion that how data is distributed and how they are and how much they are correlated each other.
- All the features weather following the normal distribution or not ?



here in this dataset some features have no good relationship so we can delete those we can delete the columns based on dependent variable class So, here I'm going to delete LongURL,ShortURL,Symbol@, Redirecting //,DomainRegLen, Favicon, UsingPopupWindow, IframeRedirection, LinksPointingToPage

```
drop(columns=['LongURL','Symbol@','ShortURL','Redirecting//','DomainRegLen','Favicon','UsingPopupWindow','IframeRedirection'],
```

```
df.head()
```

	UsingIP	PrefixSuffix-	SubDomains	HTTPS	NonStdPort	HTTPSDomainURL	RequestURL	AnchorURL	LinksInScriptTags	ServerFormHandler	...	WebsiteFor
0	1	-1	0	1	1	-1	1	0	-1	-1	...	
1	1	-1	-1	-1	1	-1	1	0	-1	-1	...	
2	1	-1	-1	-1	1	-1	-1	0	0	-1	...	
3	1	-1	1	1	1	1	1	0	0	-1	...	
4	-1	-1	1	1	1	-1	1	0	0	-1	...	

5 rows × 22 columns

```
df.shape
```

```
(5849, 22)
```

After completion of training and splitting the data we had 21 column.

➤ Milestone 3

Model Building and Comparision of Models

There are two major types of supervised machine learning problems, called classification and regression. Our data set comes under regression problem, as the prediction of suicide rate is a continuous number, or a floating-point number in programming terms. The supervised machine learning models (regression) considered to train the dataset in this notebook are: Logistic Regression, K-Nearest Neighbors , Naive Bayes, Decision Tree, Random Forest, Gradient Boosting, Multi Layer perceptron Classifier, Support Vector Machine Classifier.

The metrics considered to evaluate the model performance are Accuracy & F1 score.

To compare the models performance, a dataframe is created. The columns of this dataframe are the lists created to store the results of the model.

```

# Instantiate the models
logistic_regression = LogisticRegression()
knn = KNeighborsClassifier()
naive_bayes = GaussianNB()
decision_tree = DecisionTreeClassifier(max_depth=30)
random_forest = RandomForestClassifier(n_estimators=10)
gradient_boosting = GradientBoostingClassifier(max_depth=4, learning_rate=0.7)
mlp = MLPClassifier()
svc = SVC()

# Train and predict for each model
models = [
    ("Logistic Regression", logistic_regression),
    ("K-Nearest Neighbors", knn),
    ("Naive Bayes", naive_bayes),
    ("Decision Tree", decision_tree),
    ("Random Forest", random_forest),
    ("Gradient Boosting", gradient_boosting),
    ("Multi-Layer Perceptron", mlp),
    ("Support Vector", svc)
]

for model_name, model in models:
    print(f"Model: {model_name}")

    # Fit the model
    model.fit(X_train, y_train)

    # Predict
    y_pred = model.predict(X_test)

    # Evaluation metrics
    accuracy = accuracy_score(y_test, y_pred)
    confusion = confusion_matrix(y_test, y_pred)
    classification = classification_report(y_test, y_pred)

    print(f"Accuracy: {accuracy}")
    print(f"Confusion Matrix:\n{confusion}")
    print(f"Classification Report:\n{classification}")
    print("-" * 50)

```

Model: Logistic Regression
 Accuracy: 0.9222222222222223
 Confusion Matrix:

```
[[573 58]
 [ 33 506]]
```

Classification Report:

	precision	recall	f1-score	support
-1	0.95	0.91	0.93	631
1	0.90	0.94	0.92	539
accuracy			0.92	1170
macro avg	0.92	0.92	0.92	1170
weighted avg	0.92	0.92	0.92	1170

Model: K-Nearest Neighbors
 Accuracy: 0.9222222222222223
 Confusion Matrix:

```
[[588 43]
 [ 48 491]]
```

Classification Report:

	precision	recall	f1-score	support
-1	0.92	0.93	0.93	631
1	0.92	0.91	0.92	539
accuracy			0.92	1170
macro avg	0.92	0.92	0.92	1170
weighted avg	0.92	0.92	0.92	1170


```

-----
Model: Naive Bayes
Accuracy: 0.6623931623931624
Confusion Matrix:
[[631  0]
 [395 144]]
Classification Report:
      precision  recall  f1-score  support

-1    0.62    1.00    0.76    631
 1    1.00    0.27    0.42    539

 accuracy              0.66    1170
macro avg    0.81    0.63    0.59    1170
weighted avg    0.79    0.66    0.61    1170

```

```

-----
Model: Decision Tree
Accuracy: 0.9145299145299145
Confusion Matrix:
[[591  40]
 [ 60 479]]
Classification Report:
      precision  recall  f1-score  support

-1    0.91    0.94    0.92    631
 1    0.92    0.89    0.91    539

 accuracy              0.91    1170
macro avg    0.92    0.91    0.91    1170
weighted avg    0.91    0.91    0.91    1170

```

```

-----
Model: Random Forest
Accuracy: 0.9282051282051282
Confusion Matrix:
[[586  45]
 [ 39 500]]
Classification Report:
      precision  recall  f1-score  support

-1    0.94    0.93    0.93    631
 1    0.92    0.93    0.92    539

 accuracy              0.93    1170
macro avg    0.93    0.93    0.93    1170
weighted avg    0.93    0.93    0.93    1170

```

```

-----
Model: Gradient Boosting
Accuracy: 0.9452991452991453
Confusion Matrix:
[[596  35]
 [ 29 510]]
Classification Report:
      precision  recall  f1-score  support

-1    0.95    0.94    0.95    631
 1    0.94    0.95    0.94    539

 accuracy              0.95    1170
macro avg    0.94    0.95    0.95    1170
weighted avg    0.95    0.95    0.95    1170

```

```

-----
Model: Multi-Layer Perceptron
Accuracy: 0.9401709401709402
Confusion Matrix:
[[587  44]
 [ 26 513]]
Classification Report:
      precision  recall  f1-score  support

```

-1	0.96	0.93	0.94	631
1	0.92	0.95	0.94	539

accuracy		0.94	1170
macro avg	0.94	0.94	0.94
weighted avg	0.94	0.94	0.94

Model: Support Vector

Accuracy: 0.9384615384615385

Confusion Matrix:

[[584 47]

[25 514]]

Classification Report:

	precision	recall	f1-score	support
-1	0.96	0.93	0.94	631
1	0.92	0.95	0.93	539

accuracy		0.94	1170
macro avg	0.94	0.94	0.94
weighted avg	0.94	0.94	0.94

Printing in sorted order

```
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.svm import SVC

logistic_regression = LogisticRegression()
knn = KNeighborsClassifier()
naive_bayes = GaussianNB()
decision_tree = DecisionTreeClassifier(max_depth=30)
random_forest = RandomForestClassifier(n_estimators=10)
gradient_boosting = GradientBoostingClassifier(max_depth=4, learning_rate=0.7)
mlp = MLPClassifier()
svc = SVC()

# Train and predict for each model
models = [
    ("Logistic Regression", logistic_regression),
    ("K-Nearest Neighbors", knn),
    ("Naive Bayes", naive_bayes),
    ("Decision Tree", decision_tree),
    ("Random Forest", random_forest),
    ("Gradient Boosting", gradient_boosting),
    ("Multi-Layer Perceptron", mlp),
    ("Support Vector", svc)
]

results = []

for model_name, model in models:
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    results.append((model_name, model, accuracy))

# Sort results based on accuracy
results.sort(key=lambda x: x[2], reverse=True)

for model_name, model, accuracy in results:
    print(f"Model: {model_name}")
    print(f"Accuracy: {accuracy}")
    y_pred = model.predict(X_test)
    confusion = confusion_matrix(y_test, y_pred)
    classification_report = classification_report(y_test, y_pred)
    print(f"Confusion Matrix:\n{confusion}")
    print(f"Classification Report:\n{classification_report}")
    print("-" * 50)
```

Model: Gradient Boosting
 Accuracy: 0.9452991452991453
 Confusion Matrix:
 [[596 35]
 [29 510]]
 Classification Report:

	precision	recall	f1-score	support
-1	0.95	0.94	0.95	631
1	0.94	0.95	0.94	539
accuracy			0.95	1170
macro avg	0.94	0.95	0.95	1170
weighted avg	0.95	0.95	0.95	1170

Model: Random Forest
 Accuracy: 0.941025641025641
 Confusion Matrix:
 [[596 35]
 [34 505]]
 Classification Report:

	precision	recall	f1-score	support
-1	0.95	0.94	0.95	631
1	0.94	0.94	0.94	539
accuracy			0.94	1170
macro avg	0.94	0.94	0.94	1170
weighted avg	0.94	0.94	0.94	1170

Model: Multi-Layer Perceptron
 Accuracy: 0.9393162393162393
 Confusion Matrix:
 [[594 37]
 [34 505]]
 Classification Report:

	precision	recall	f1-score	support
-1	0.95	0.94	0.94	631
1	0.93	0.94	0.93	539
accuracy			0.94	1170
macro avg	0.94	0.94	0.94	1170
weighted avg	0.94	0.94	0.94	1170

Model: Support Vector
 Accuracy: 0.9384615384615385
 Confusion Matrix:
 [[584 47]
 [25 514]]
 Classification Report:

	precision	recall	f1-score	support
-1	0.96	0.93	0.94	631
1	0.92	0.95	0.93	539
accuracy			0.94	1170
macro avg	0.94	0.94	0.94	1170
weighted avg	0.94	0.94	0.94	1170

Model: Logistic Regression
 Accuracy: 0.9222222222222223
 Confusion Matrix:
 [[573 58]
 [33 506]]
 Classification Report:

	precision	recall	f1-score	support
--	-----------	--------	----------	---------

-1	0.95	0.91	0.93	631
1	0.90	0.94	0.92	539

accuracy		0.92	1170
macro avg	0.92	0.92	0.92 1170
weighted avg	0.92	0.92	0.92 1170

Model: K-Nearest Neighbors
Accuracy: 0.922222222222223
Confusion Matrix:
[[588 43]
[48 491]]
Classification Report:

	precision	recall	f1-score	support
-1	0.92	0.93	0.93	631
1	0.92	0.91	0.92	539

accuracy		0.92	1170
macro avg	0.92	0.92	0.92 1170
weighted avg	0.92	0.92	0.92 1170

Model: Decision Tree
Accuracy: 0.9128205128205128
Confusion Matrix:
[[589 42]
[60 479]]
Classification Report:

	precision	recall	f1-score	support
-1	0.91	0.93	0.92	631
1	0.92	0.89	0.90	539

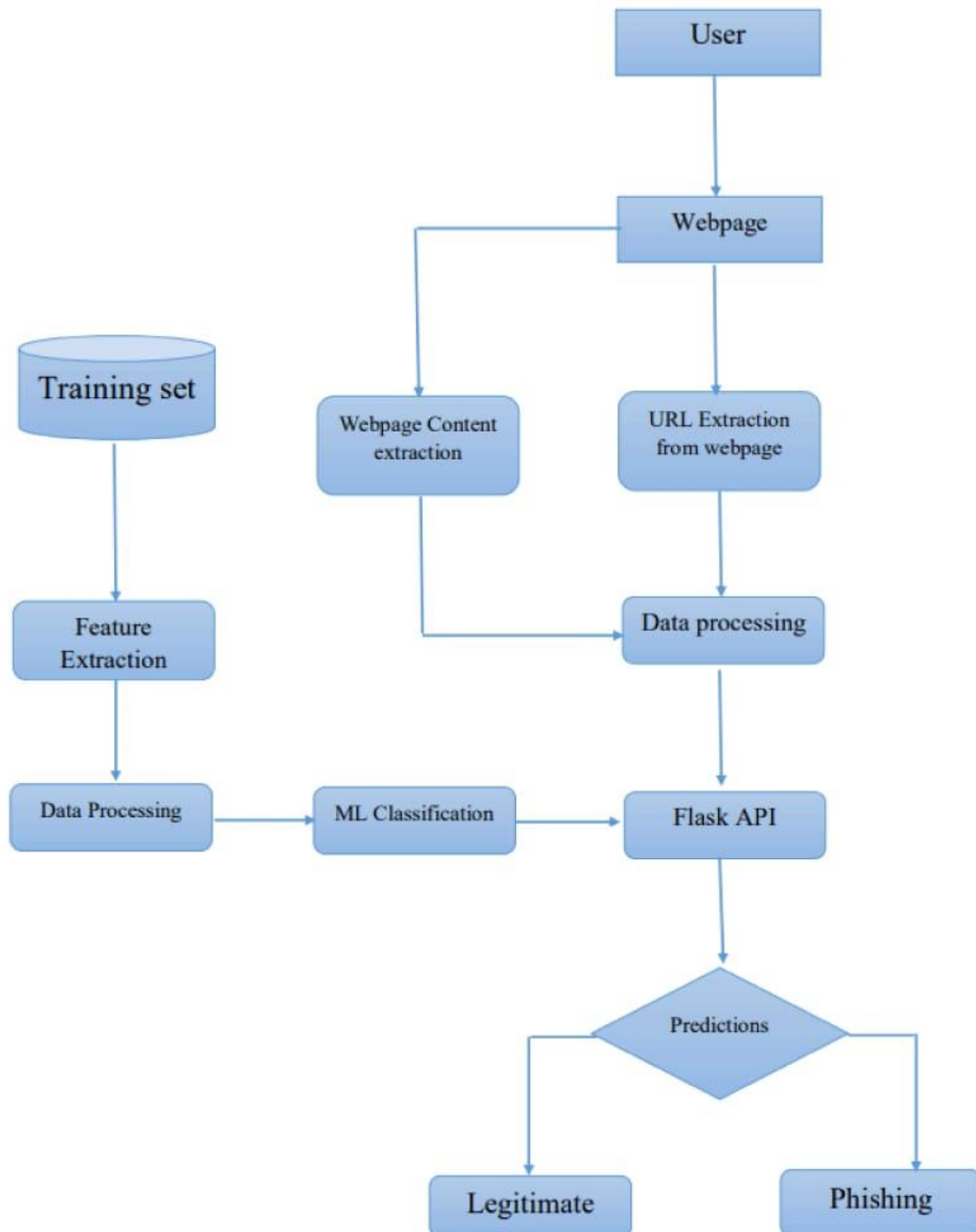
accuracy		0.91	1170
macro avg	0.91	0.91	0.91 1170
weighted avg	0.91	0.91	0.91 1170

Model: Naive Bayes
Accuracy: 0.6623931623931624
Confusion Matrix:
[[631 0]
[395 144]]
Classification Report:

	precision	recall	f1-score	support
-1	0.62	1.00	0.76	631
1	1.00	0.27	0.42	539

accuracy		0.66	1170
macro avg	0.81	0.63	0.59 1170
weighted avg	0.79	0.66	0.61 1170

5. FLOWCHART



6. RESULT

For this project create three HTML files namely

- index.html
- inspect.html
- output.html

and save them in templates folder.

This is how our index.html page looks like:



Now when you click on inspect button from top right corner you will get redirected to Inspect.html

Lets look how our Inspect.html file looks like:

[Home](#)
[About](#)
[Contact](#)

URL-Based Phishing Detection

UsingIP

PrefixSuffix-

SubDomains

HTTPS

NonStdPort

HTTPSDomainURL

RequestURL

AnchorURL

LinksInScriptTags

ServerFormHandler

InfoEmail

AbnormalURL

WebsiteForwarding

StatusBarCust

DisableRightClick

AgeofDomain

DNSRecording

WebsiteTraffic

PageRank

GoogleIndex

StatsReport

Predict

[Home](#)
[About](#)
[Contact](#)

[Home](#)
[About](#)
[Contact](#)

URL-Based Phishing Detection

UsingIP

PrefixSuffix-

SubDomains

HTTPS

NonStdPort

HTTPSDomainURL

RequestURL

AnchorURL

LinksInScriptTags

ServerFormHandler

InfoEmail

AbnormalURL

WebsiteForwarding

StatusBarCust

DisableRightClick

AgeofDomain

DNSRecording

WebsiteTraffic

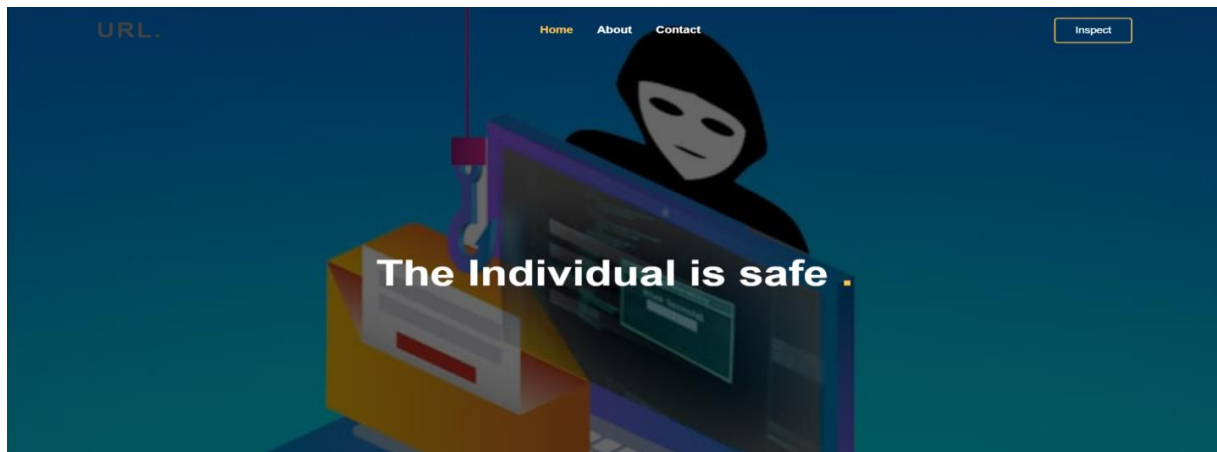
PageRank

GoogleIndex

StatsReport

Predict

[Home](#)
[About](#)
[Contact](#)

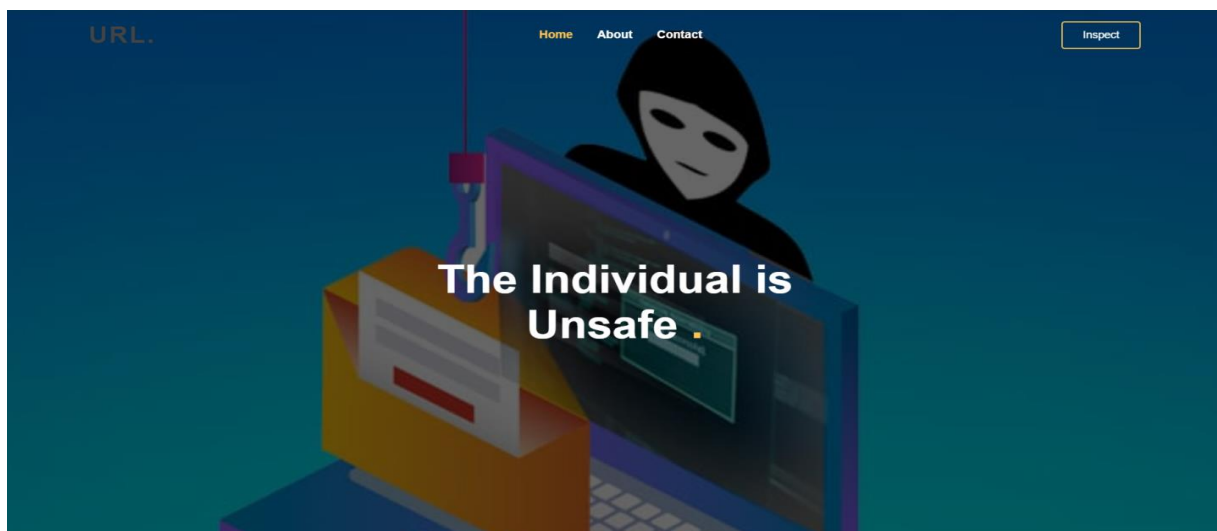


Home About Contact

URL-Based Phishing Detection

UsingIP	NonStdPort	LinksInScriptTags	StatusBarCust	WebsiteTraffic
<input type="text" value="-1"/>	<input type="text" value="-1"/>	<input type="text" value="-1"/>	<input type="text" value="-1"/>	<input type="text" value="-1"/>
PrefixSuffix-	HTTPSDomainURL	ServerFormHandler	DisableRightClick	PageRank
<input type="text" value="1"/>	<input type="text" value="1"/>	<input type="text" value="1"/>	<input type="text" value="-1"/>	<input type="text" value="-1"/>
SubDomains	RequestURL	InfoEmail	AgeofDomain	GoogleIndex
<input type="text" value="1"/>	<input type="text" value="1"/>	<input type="text" value="1"/>	<input type="text" value="-1"/>	<input type="text" value="-1"/>
HTTPS	AnchorURL	AbnormalURL	DNSRecording	StatsReport
<input type="text" value="-1"/>	<input type="text" value="-1"/>	<input type="text" value="-1"/>	<input type="text" value="-1"/>	<input type="text" value="-1"/>
		WebsiteForwarding		
		<input type="text" value="-1 "/>		
		<input type="button" value="Predict"/>		

Will try with different numbers and then click on predict button.



7. ADVANTAGES & DISADVANTAGES

Advantages: -

- *Real-time Detection:* Machine learning models can analyze URLs quickly and in real-time, enabling rapid identification of phishing links, reducing the risk of falling victim to scams.
- *Scalability:* ML-based systems can handle a large number of URLs simultaneously, making them scalable to protect a vast user base or an entire organization from phishing threats.
- *Continuous Learning:* Machine learning models can adapt and improve over time by continuously learning from new phishing patterns, staying up-to-date with emerging threats.
- *Accuracy:* Advanced ML algorithms can achieve high accuracy in detecting phishing URLs, minimizing false positives and false negatives, leading to more reliable protection.
- And some other advantages are automation, customizability, early warning, multi-platform support, enhanced security, data driven insights.

Disadvantages:

- *False Positives and False Negatives:* Machine learning models may occasionally misclassify legitimate URLs as phishing or fail to detect sophisticated phishing attempts, leading to false positives and false negatives, respectively.
- *Data Privacy and Security:* ML-based phishing detection often involves analyzing URLs, which could raise privacy concerns if sensitive or personal data is unintentionally processed during the analysis.
- *Rapidly Evolving Phishing Techniques:* As phishing techniques evolve, ML models might struggle to keep up with the latest strategies used by attackers.
- *Dependency on Data Sources:* ML models for phishing detection depend on timely access to relevant data sources, and any disruption in data availability could impact their performance.

8. APPLICATIONS

URL-based phishing detection using machine learning has various practical applications, including:

1. Email Security: Integrating ML-based phishing detection in email systems helps identify and block phishing links, protecting users from clicking on malicious URLs sent via emails.
2. Web Browsers: Browser extensions or built-in features that utilize ML can alert users about potential phishing websites when they attempt to visit suspicious URLs.
3. Network Security: Employing ML models in network security systems can help detect and block phishing URLs in real-time, safeguarding users and organizations from cyber threats.
4. Mobile Security: Mobile apps can leverage ML-based phishing detection to warn users about fraudulent links, ensuring safer browsing on smartphones and tablets.

Some other applications are cloud services, anti-phishing solutions, social media platforms, anti-virus and security software.

By utilizing machine learning in URL-based phishing detection, these applications can effectively mitigate phishing risks and enhance overall cybersecurity.

9. CONCLUSION

1. The final take away from this project is to explore various machine learning models, perform Exploratory Data Analysis on phishing dataset and understanding their features.
2. Creating this notebook helped me to learn a lot about the features affecting the models to detect whether URL is safe or not, also I came to know how to tune model and how they affect the model performance.
3. The final conclusion on the Phishing dataset is that some features like "HTTPS", "AnchorURL", "WebsiteTraffic" have more importance to classify URL is phishing URL or not.
4. Gradient Boosting Classifier correctly classifies URL up to 94.52% respective classes and hence reduces the chance of malicious attachments.

10. FUTURE SCOPE

In future if we get structured dataset of phishing we can perform phishing detection much more faster than any other technique. In future we can use a combination of any other two or more classifier to get maximum accuracy.

11. BIBLIOGRAPHY

- **ML Concepts**

- Supervised learning: <https://www.javatpoint.com/supervised-machine-learning>
- Unsupervised learning: <https://www.javatpoint.com/unsupervised-machine-learning>
- Decision tree: <https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm>
- Random forest: <https://www.javatpoint.com/machine-learning-random-forest-algorithm>
- KNN: <https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning>
- Support vector machine algorithm: <https://www.javatpoint.com/machine-learning-support-vector-machine-algorithm>
- Logistic Regression: <https://www.javatpoint.com/logistic-regression-in-machine-learning>
- Naïve Bayes Classifier : <https://www.javatpoint.com/machine-learning-naive-bayes-classifier>
- Gradient boosting: <https://www.javatpoint.com/gbm-in-machine-learning>
- Multi-layer Perceptron: <https://www.javatpoint.com/multi-layer-perceptron-in-tensorflow>
- Evaluation metrics: <https://www.analyticsvidhya.com/blog/2019/08/11-important-model-evaluation-error-metrics/>

- **Flask Basics :** https://www.youtube.com/watch?v=lj4I_CvBnt0

APPENDIX

11.1. Source Code

Import the libraries

```
from flask import Flask, request, render_template, url_for
import pickle
import numpy as np
import json
import requests
```

Load the saved model. Importing flask module in the project is mandatory. An object of Flask class is our WSGI application. Flask constructor takes the name of the current module (`__name__`) as argument.

```
app = Flask(__name__)
with open('Phishing_model.pkl', 'rb') as file:
    model = pickle.load(file)
```

Render HTML page:

```
@app.route("/")
def f():
    return render_template("index.html")

@app.route("/inspect")
def inspect():
    return render_template("inspect.html")
```

Here we will be using declared constructor to route to the HTML page which we have created earlier.

In the above example, '/' URL is bound with index.html function. Hence, when the index page of the web server is opened in browser, the html page will be rendered. Whenever you enter the values from the html page the values can be retrieved using POST Method.

Retrieves the value from UI:

```

@app.route("/output", methods=["GET", "POST"])
def output():
    if request.method == 'POST':
        var1 = request.form["UsingIP"]
        var2 = request.form["PrefixSuffix-"]
        var3 = request.form["SubDomains"]
        var4 = request.form["HTTPS"]
        var5 = request.form["NonStdPort"]
        var6 = request.form["HTTPSDomainURL"]
        var7 = request.form["RequestURL"]
        var8 = request.form["AnchorURL"]
        var9 = request.form["LinksInScriptTags"]
        var10 = request.form["ServerFormHandler"]
        var11 = request.form["InfoEmail"]
        var12 = request.form["AbnormalURL"]
        var13 = request.form["WebsiteForwarding"]
        var14 = request.form["StatusBarCust"]
        var15 = request.form["DisableRightClick"]
        var16 = request.form["AgeofDomain"]
        var17 = request.form["DNSRecording"]
        var18 = request.form["WebsiteTraffic"]
        var19 = request.form["PageRank"]
        var20 = request.form["GoogleIndex"]
        var21 = request.form["StatsReport"]
        # Convert the input data into a numpy array
        predict_data = np.array([var1, var2, var3, var4, var5, var6, var7, var8, var9, var10,
                                var11, var12, var13, var14, var15, var16, var17, var18, var19, var20,
                                var21]).reshape(1, -1)

        # Use the loaded model to make predictions
        predict = model.predict(predict_data)
        # 1 is safe
        # -1 is unsafe
        if (predict == 1):
            return render_template('output.html', predict="safe")
        else:
            return render_template('output.html', predict="Unsafe")
    return render_template("output.html")
if __name__ == "__main__":
    app.run(debug=False)

```

Here we are routing our app to output() function. This function retrieves all the values from the HTML page using Post request. That is stored in an array. This array is passed to the model.predict() function. This function returns the prediction. And this prediction value will be rendered to the text that we have mentioned in the output.html page earlier.

Main function:

```

if __name__ == "__main__":
    app.run(debug=False)

```

To run the application:

- Open anaconda prompt from the start menu
- Navigate to the folder where your python script is.
- Now type “python app.py” command
- Navigate to the localhost where you can view your web page.
- Click on the inspect button from the top right corner, enter the inputs, click on the predict button, and see the result/prediction on the web.