Cairo University

Faculty of Engineering
Cairo University

Big Data and Cloud Computing

#CMP4011

Project Report

Team 10

Submitted to:

Eng. Omar Samir

Submitted By:

NAME	SEC	BN	ID
Sarah Mohamed Hossam	1	29	9202618
Abdelrahman Fathy	2	3	9202846
Yasmine Ashraf Ghanem	2	37	9203707
Yasmin Abdullah Nasser	2	38	9203717

Brief problem description Idea:

The dataset is designed for the detection and analysis of phishing URLs. Phishing attacks involve malicious websites pretending to be legitimate with the aim of deceiving individuals into proclaiming personal and sensitive information. It contains various features of URLs, including the lexical characteristics of the web addresses, website content features, and host-based features. This can give us a comprehensive framework for distinguishing between phishing and legitimate websites.

Project pipeline:

- Data Preprocessing
- Data Descriptive statistics and analysis
- Data Visualization
- Model & Training
- Evaluation

Analysis and solution:

Data Preprocessing (Step 1):

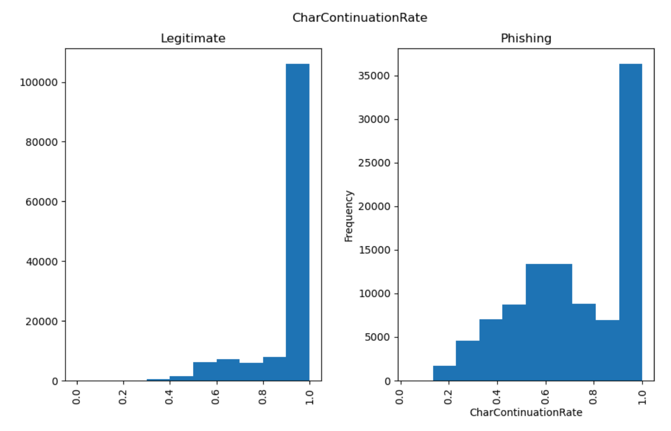
1. Read Dataset
2. Check for missing or null values
3. Check for the unique values of some columns {URL}
4. Transfer categorical features to ONE-HOT encoding vector
5. Splitting the data into *Training, Testing & Validation*

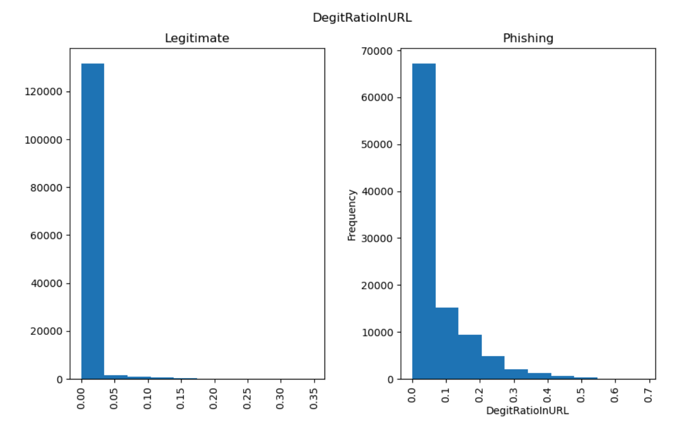
Data visualization:

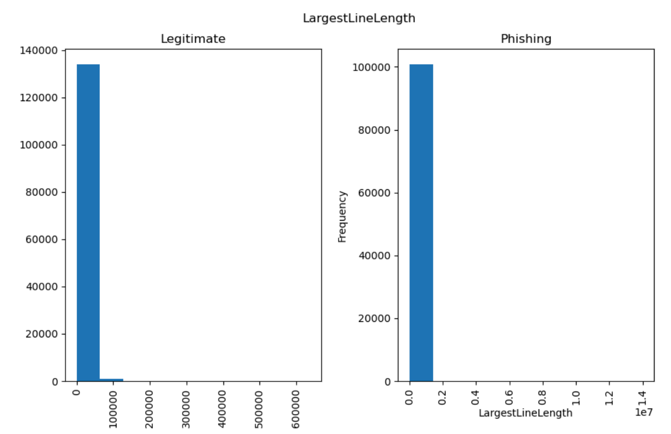
The problem we address aims to classify a URL as either Phishing or Legitimate, so we only have 2 classes.

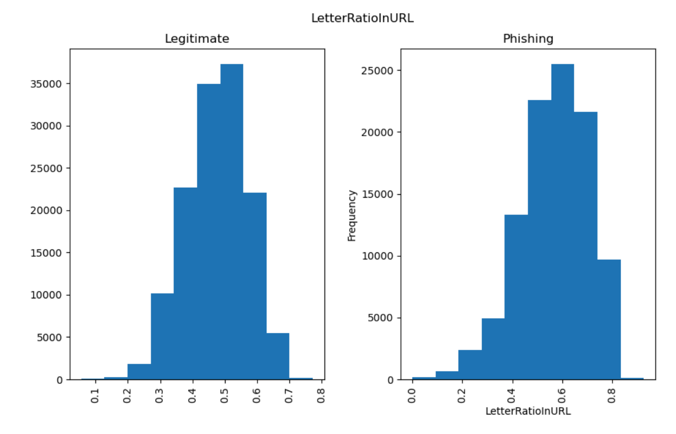
After carefully studying the data, we noticed that the features consisted of binary features (IsHTTPS, HasTitle… etc), numeric values (URLLength, URLSimilarityIndex… etc) and some strings (Domain, TLD… etc).
We manually split the feature columns into binary and numeric ones, to visualize each type in a certain way.

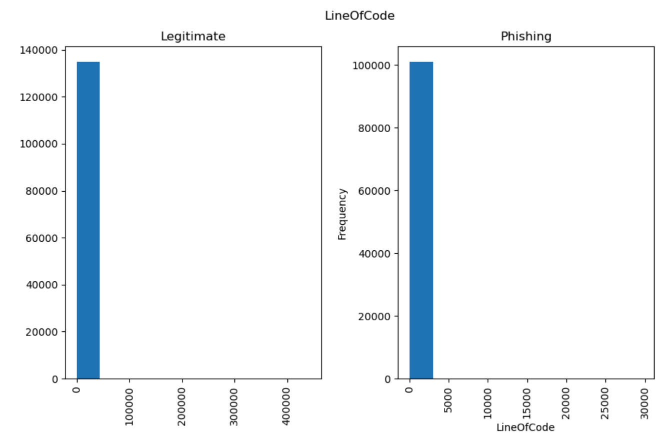
For numeric features, we used histograms to show the distribution of each column over the labels (phishing or legitimate).
The histograms are shown below.

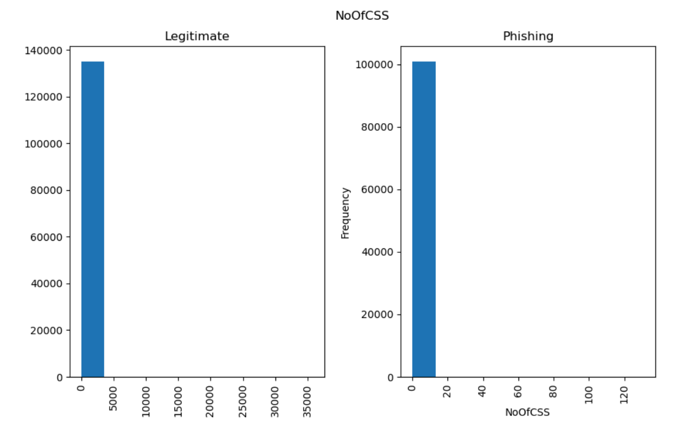


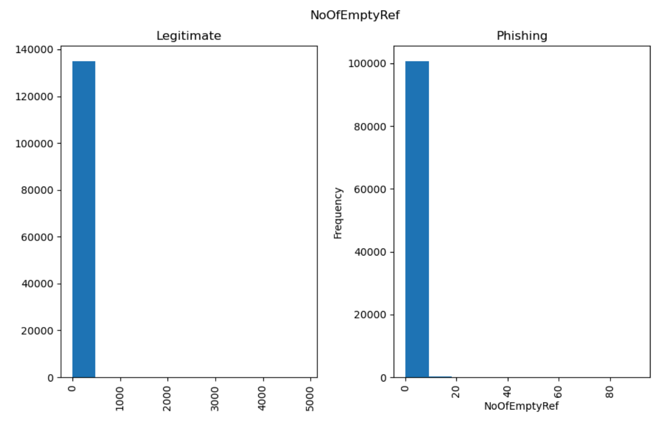


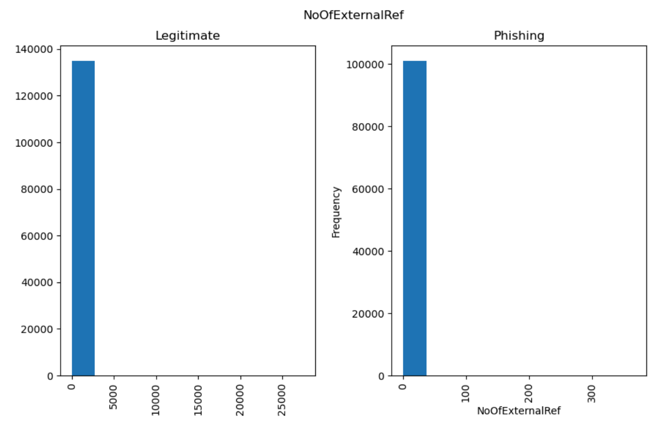


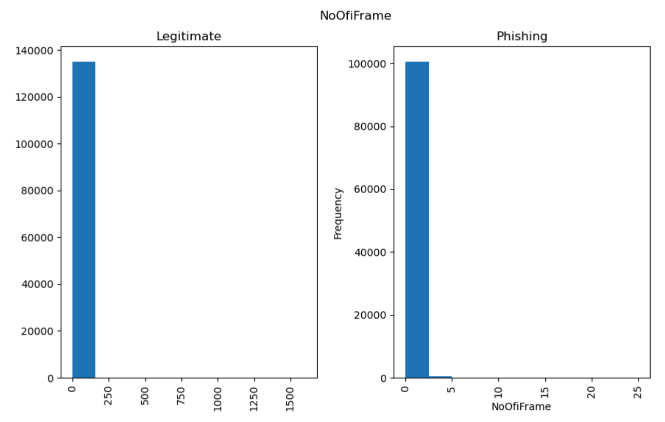


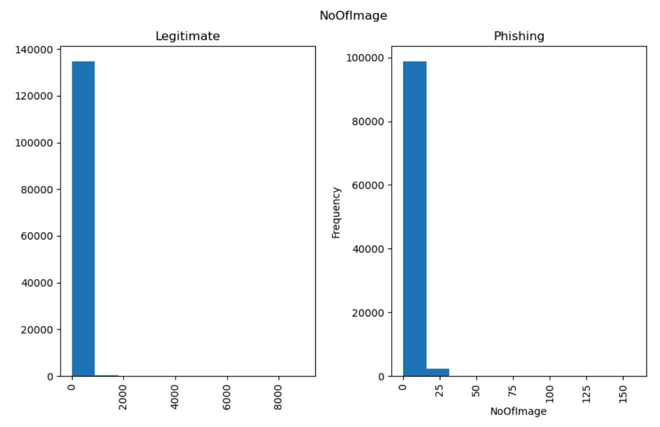


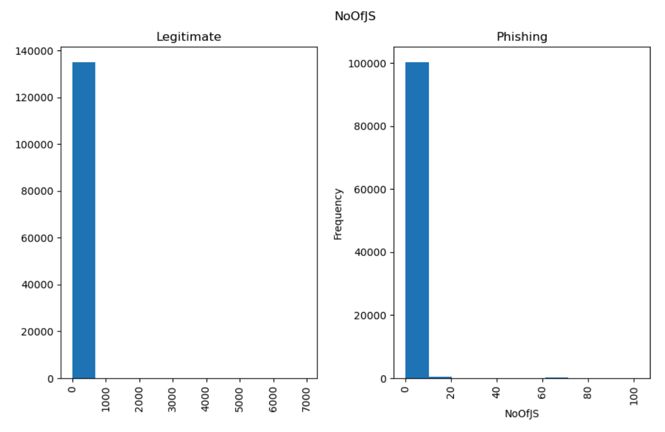


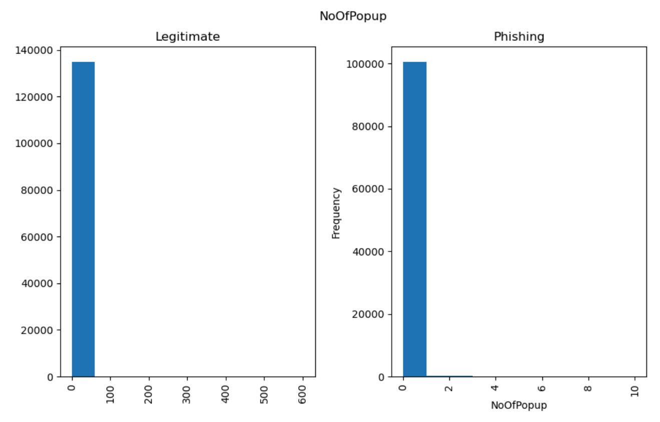

NoOfEmptyRef
Legitimate
Phishing
Frequency
NoOfEmptyRef

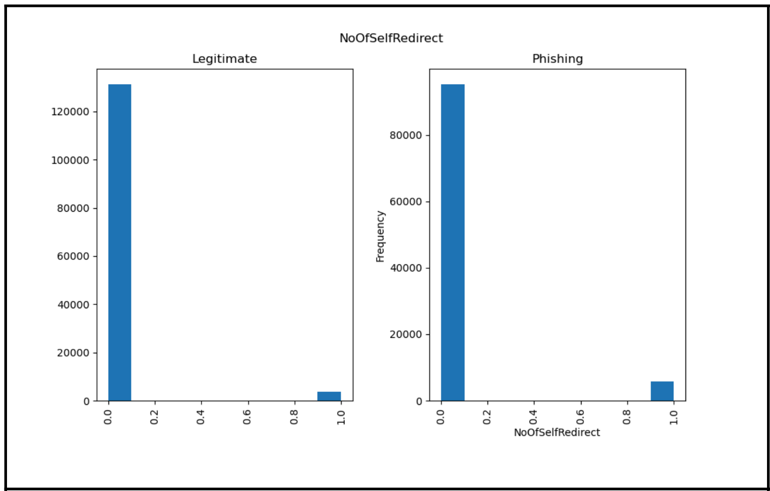

NoOfExternalRef
Legitimate
Phishing
Frequency
NoOfExternalRef

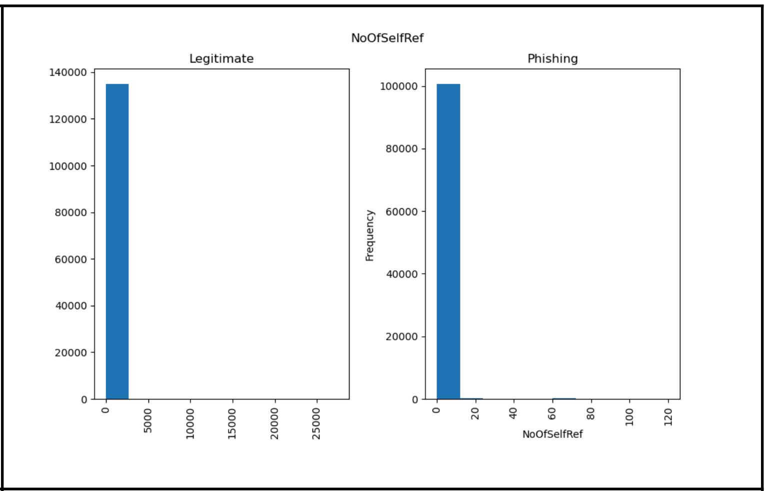

NoOfiFrame
Legitimate
Phishing
Frequency
NoOfiFrame

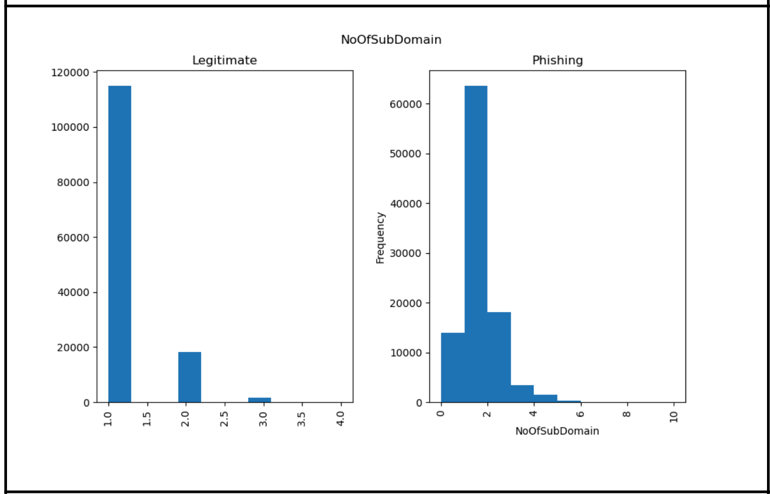

NoOfImage
Legitimate
Phishing
Frequency
NoOfImage

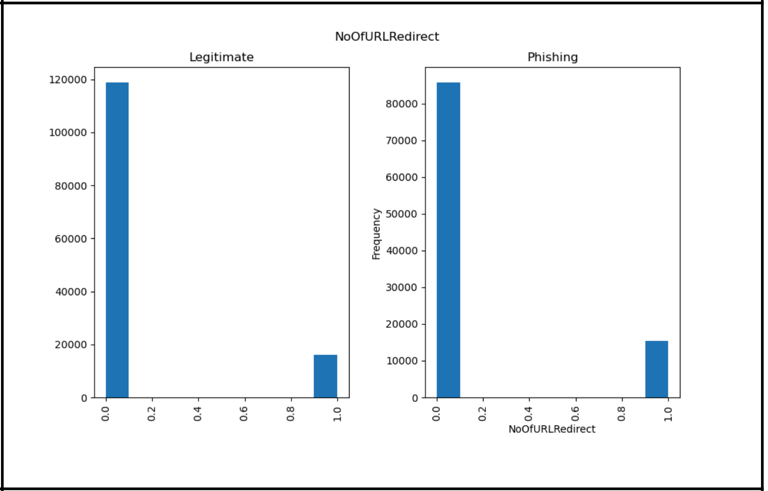

NoOfJS
Legitimate
Phishing
Frequency
NoOfJS

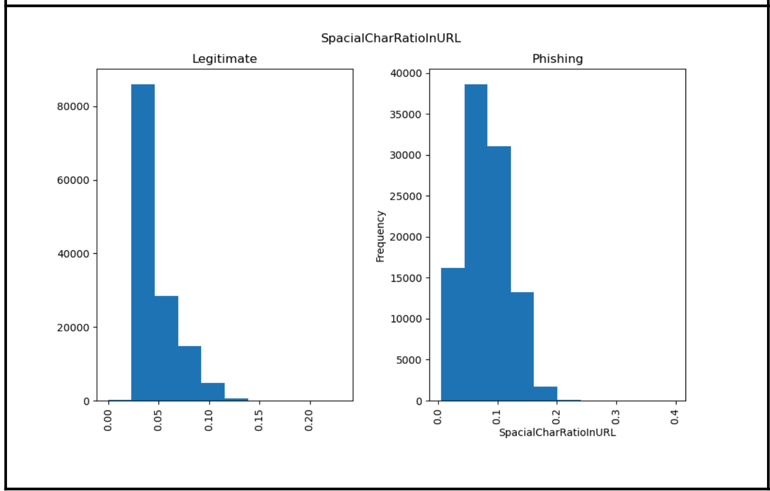

NoOfPopup
Legitimate
Phishing
Frequency
NoOfPopup

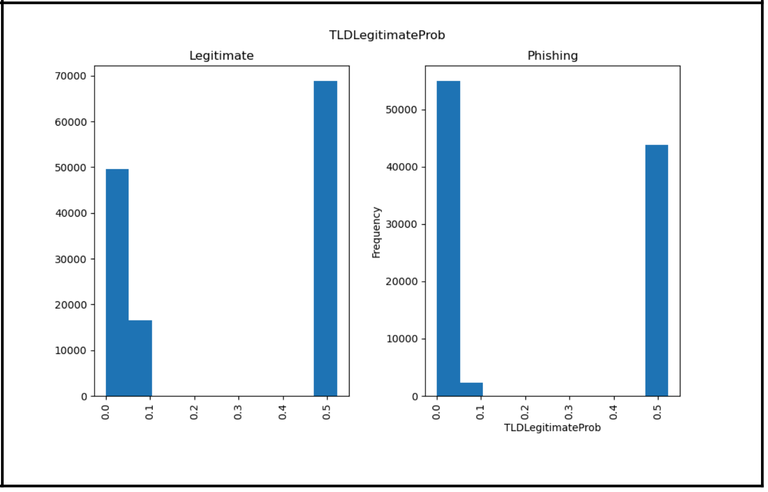

NoOfSelfRedirect
Legitimate
Phishing
Frequency
NoOfSelfRedirect

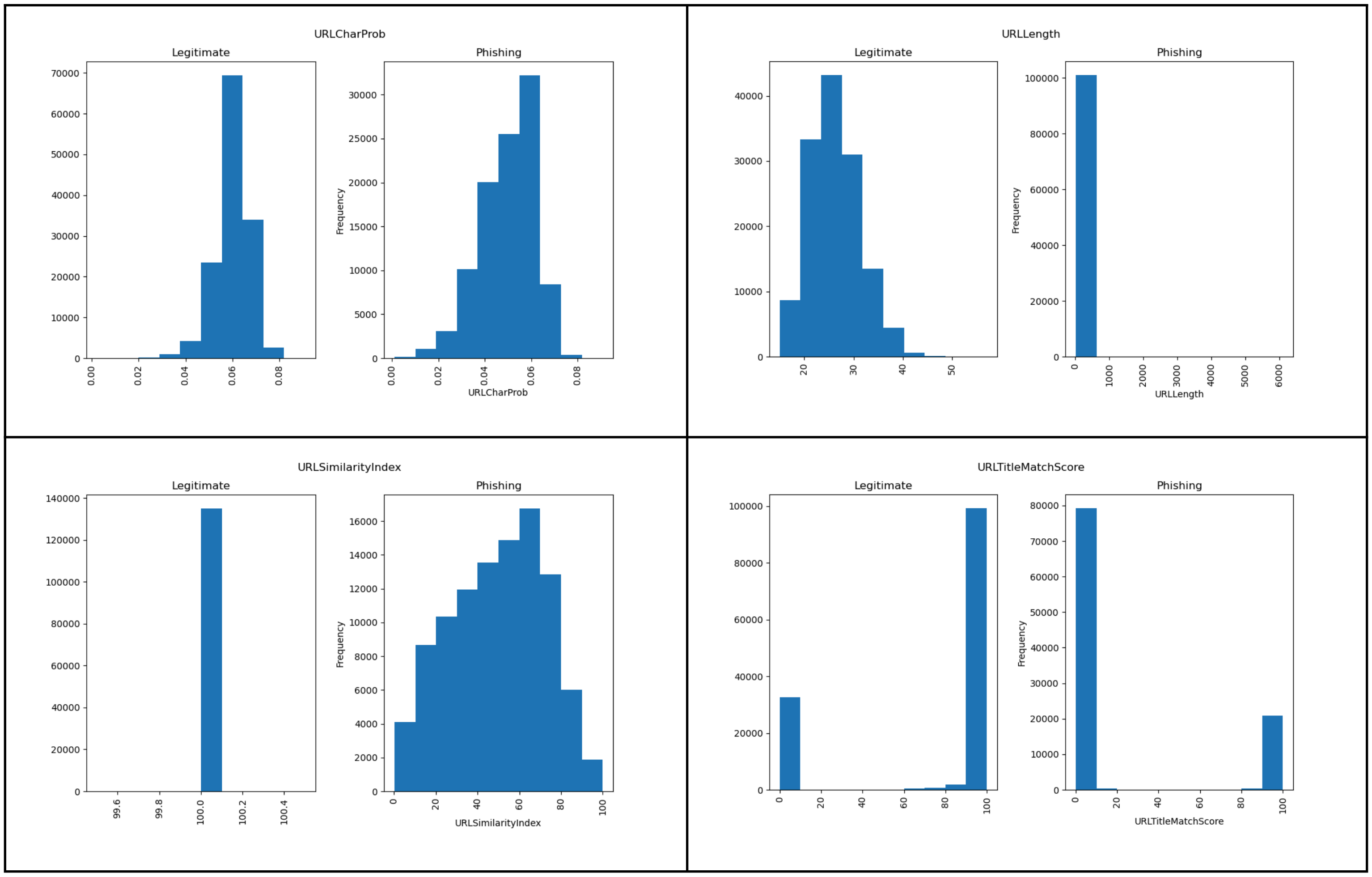

NoOfSelfRef
Legitimate
Phishing
Frequency
NoOfSelfRef


NoOfSubDomain
Legitimate
Phishing
Frequency
NoOfSubDomain


NoOfURLRedirect
Legitimate
Phishing
Frequency
NoOfURLRedirect

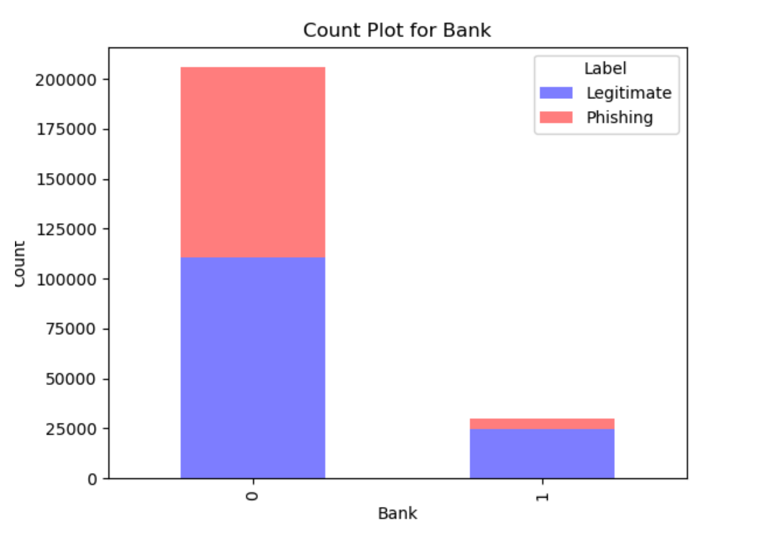

SpacialCharRatioInURL
Legitimate
Phishing
Frequency
SpacialCharRatioInURL

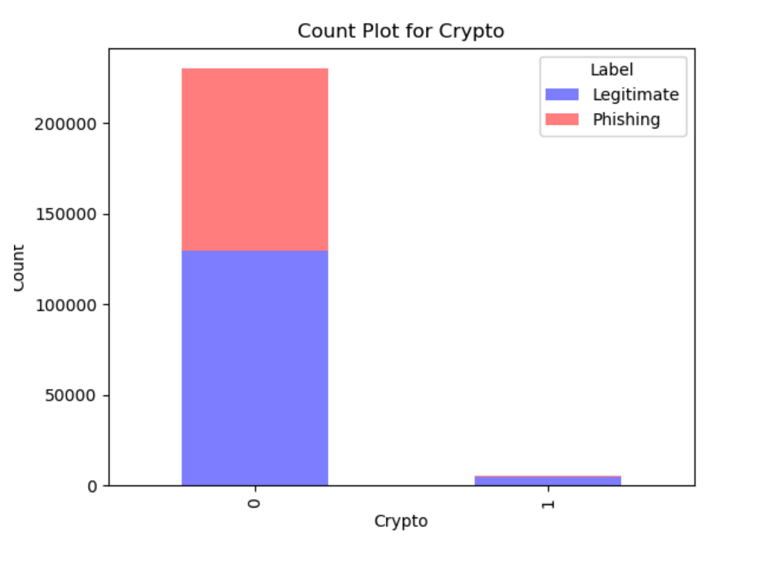

TLDLegitimateProb
Legitimate
Phishing
Frequency
TLDLegitimateProb

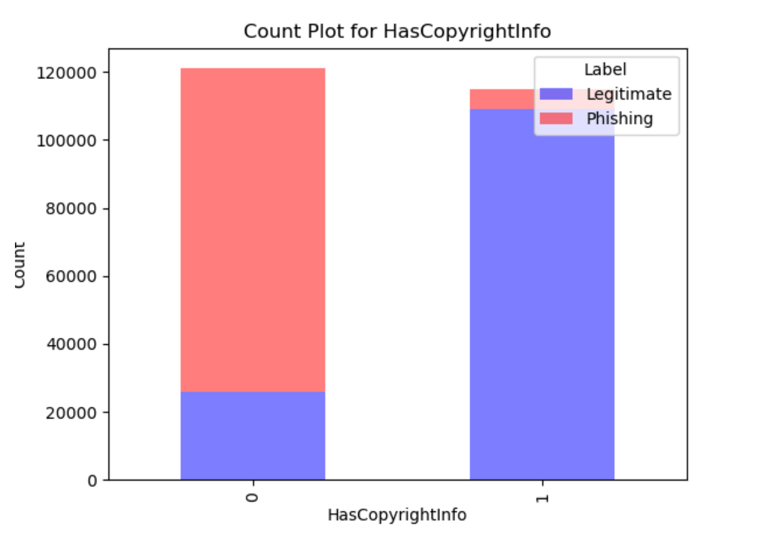


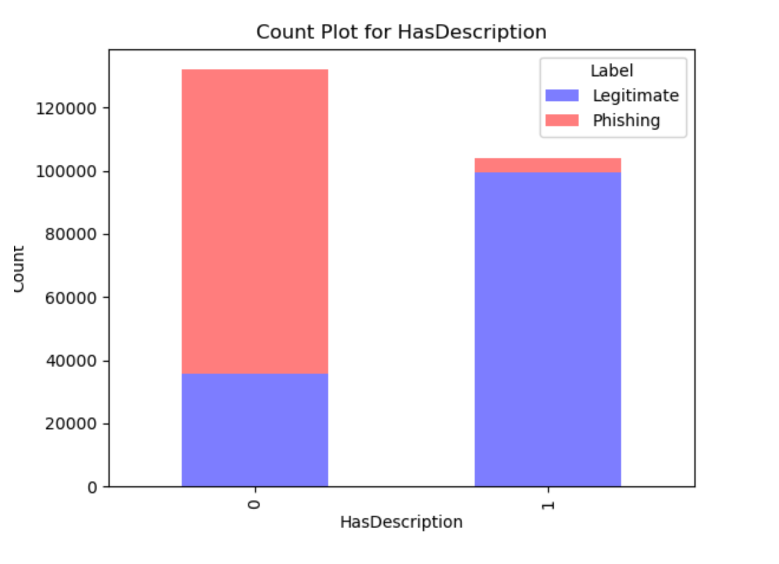
By observing the data distributions, we dropped some features we found that don't differentiate enough between the classes, like the *NoOfURLRedirect*.
Also, there are some features that strongly differentiate between the classes like the *URLSimilarityIndex*.

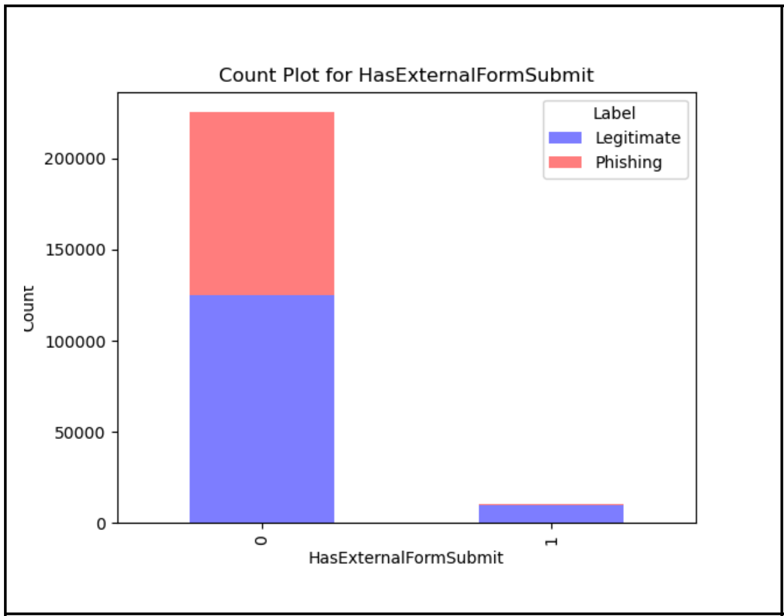
For binary features, we used count plots to show the distribution of each column over the labels (phishing or legitimate).
The plots are shown below.

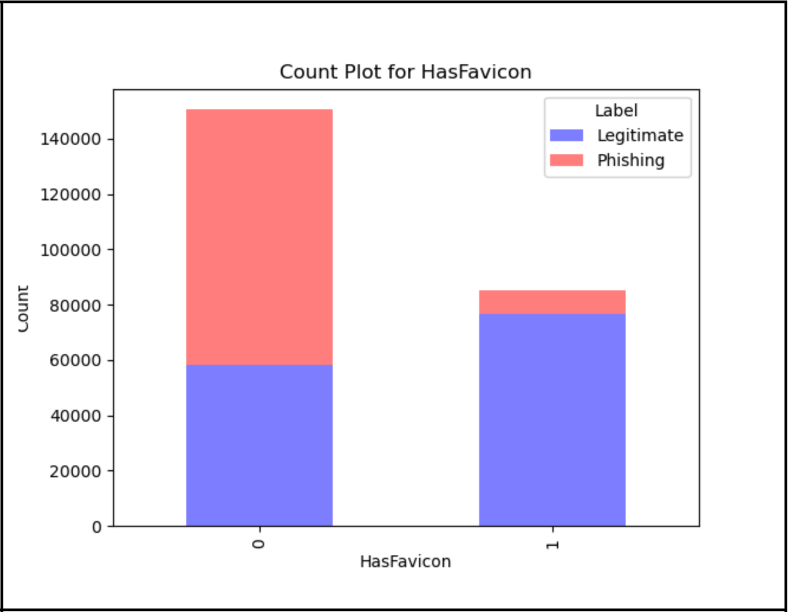


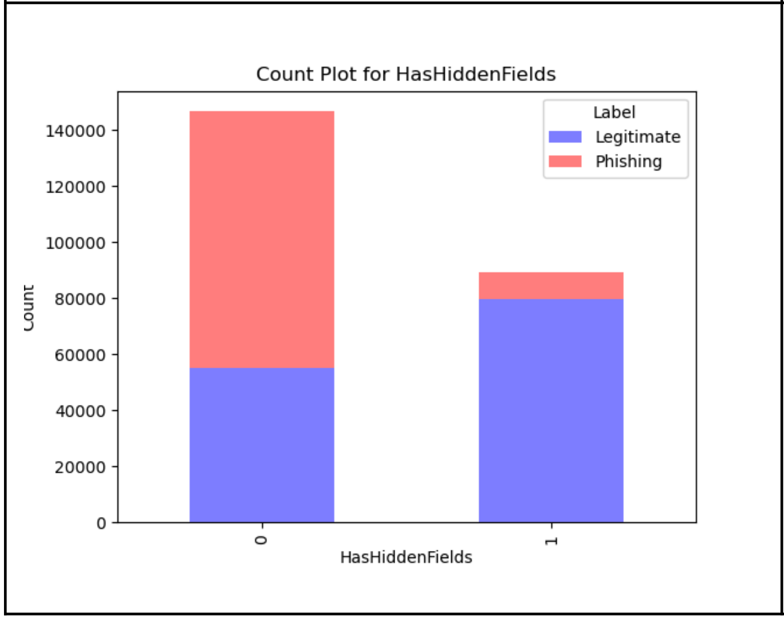


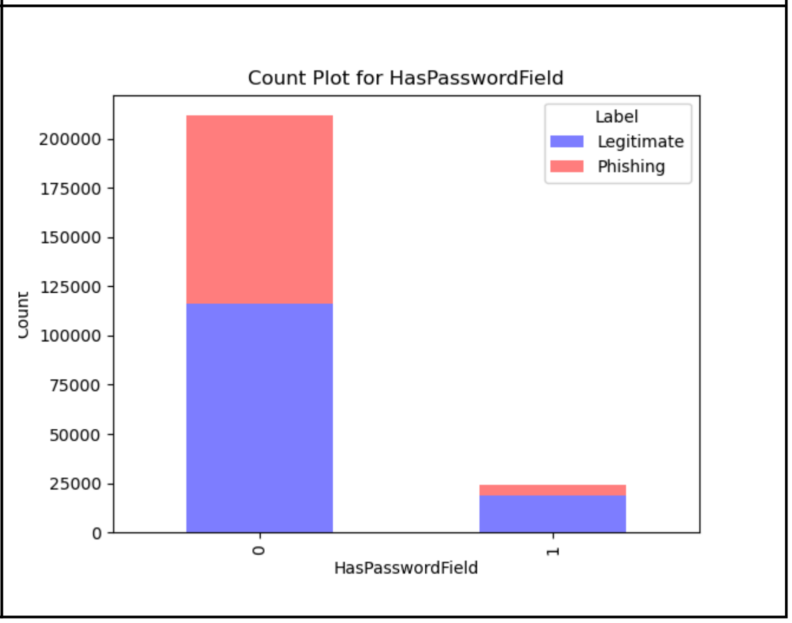


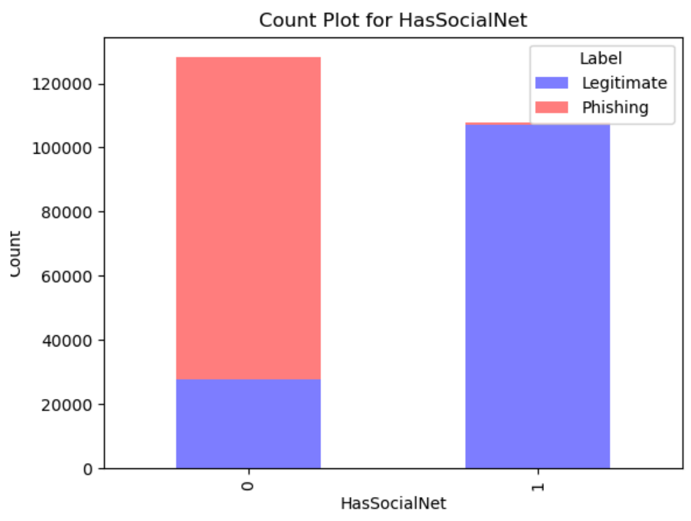


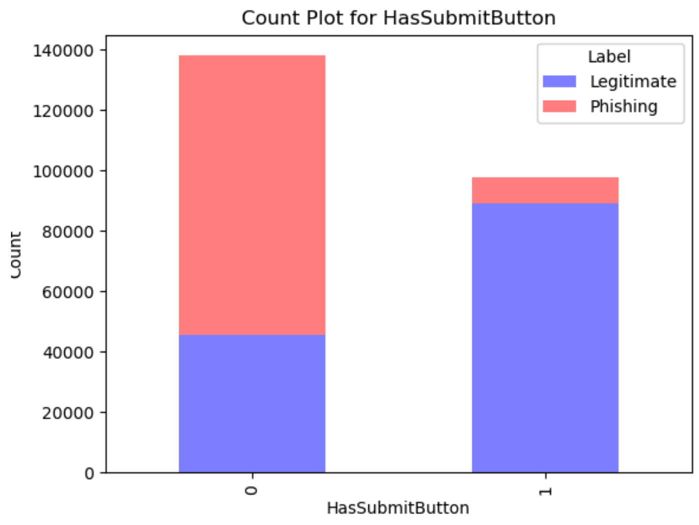

Count Plot for HasExternalFormSubmit
Label
Legitimate
Phishing
Count
0
50000
100000
150000
200000
0
1
HasExternalFormSubmit

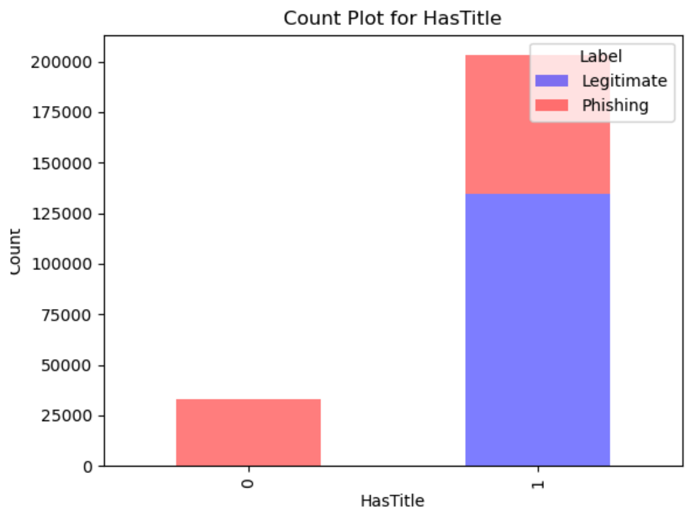

Count Plot for HasFavicon
Label
Legitimate
Phishing
Count
0
20000
40000
60000
80000
100000
120000
140000
0
1
HasFavicon

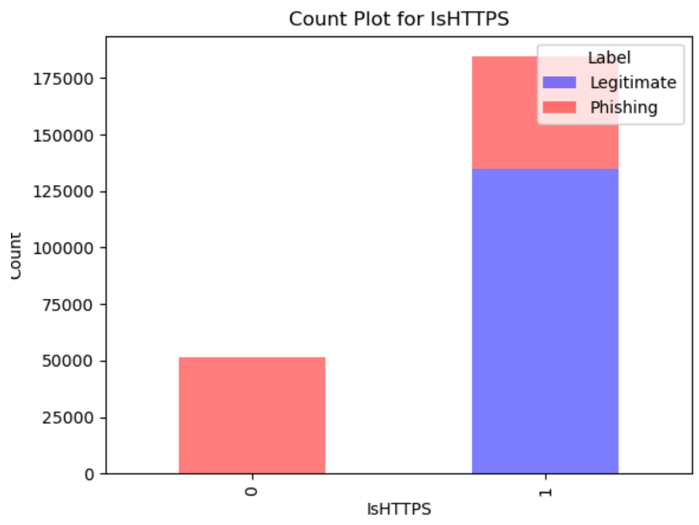

Count Plot for HasHiddenFields
Label
Legitimate
Phishing
Count
0
20000
40000
60000
80000
100000
120000
140000
0
1
HasHiddenFields

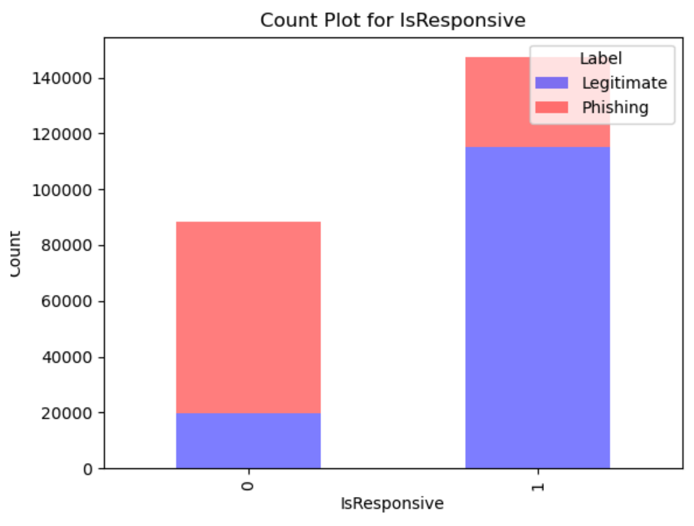

Count Plot for HasPasswordField
Label
Legitimate
Phishing
Count
0
25000
50000
75000
100000
125000
150000
175000
200000
0
1
HasPasswordField

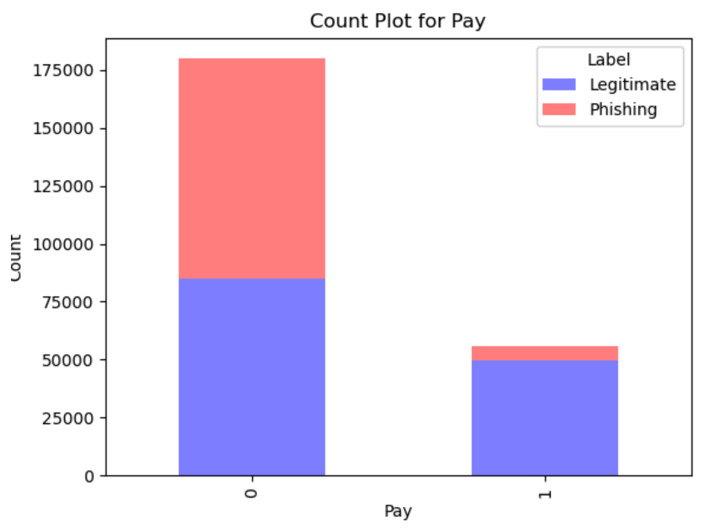

Count Plot for HasSocialNet
Label
Legitimate
Phishing
Count
HasSocialNet
0
1
0
20000
40000
60000
80000
100000
120000

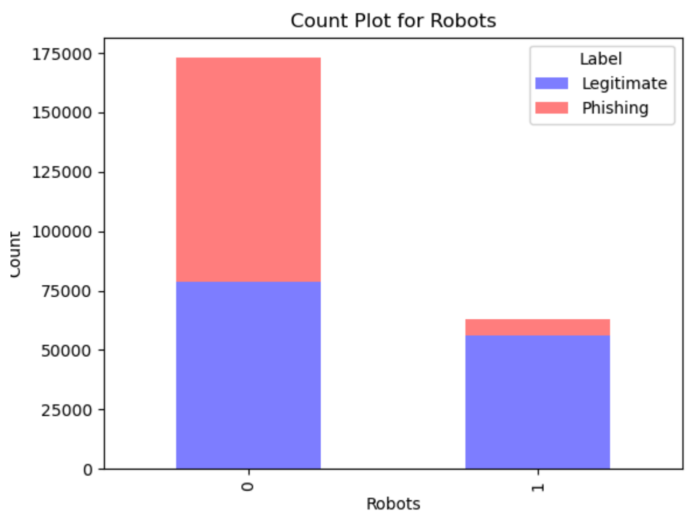

Count Plot for HasSubmitButton
Label
Legitimate
Phishing
Count
HasSubmitButton
0
1
0
20000
40000
60000
80000
100000
120000
140000


Count Plot for HasTitle
Label
Legitimate
Phishing
Count
HasTitle
0
1
0
25000
50000
75000
100000
125000
150000
175000
200000


Count Plot for IsHTTPS
Label
Legitimate
Phishing
Count
IsHTTPS
0
1
0
25000
50000
75000
100000
125000
150000
175000







Based on the above shown count plots, some features weren't very informative, like *Robots* and *HasExternalFormSubmit*.

Only around 50% of the columns were kept after the visualization. *"URLLength", "URLSimilarityIndex", "CharContinuationRate", "URLCharProb", "NoOfSubDomain", "DigitRatioInURL", "SpacialCharRatioInURL", "URLTitleMatchScore", "NoOfImage", "NoOfJS", "IsHTTPS", "HasTitle", "IsResponsive", "HasDescription", "HasExternalFormSubmit", "HasSocialNet", "HasSubmitButton", "HasHiddenFields", "HasPasswordField", "Bank", "Pay", "Crypto" and "HasCopyrightInfo".*

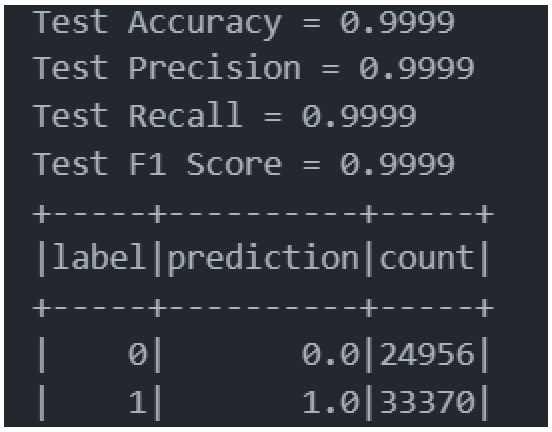
We saved a new copy of the data with the selected columns for convenience, and also found that the string columns weren't really important.
This is the first "*round*" of data preprocessing and visualization.

Model/Classifier training:

Random Forest:

Starting with a simple random forest classifier using the selected 23 columns, the model's evaluation metrics were exceedingly high.

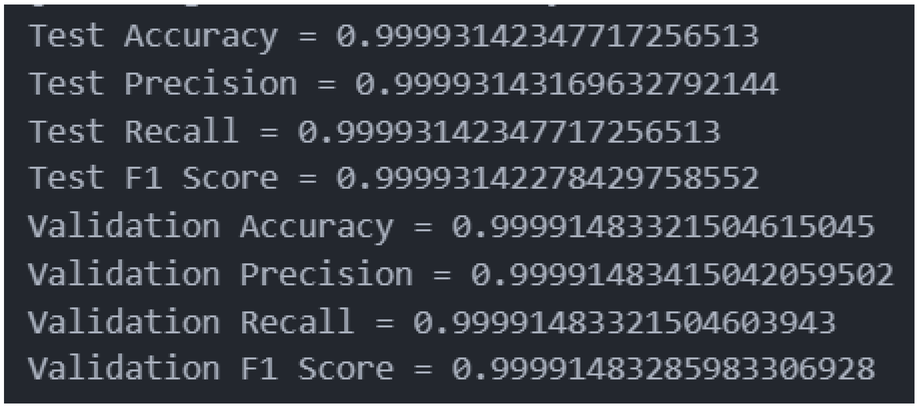
- Using cross validation, with 5 folds.



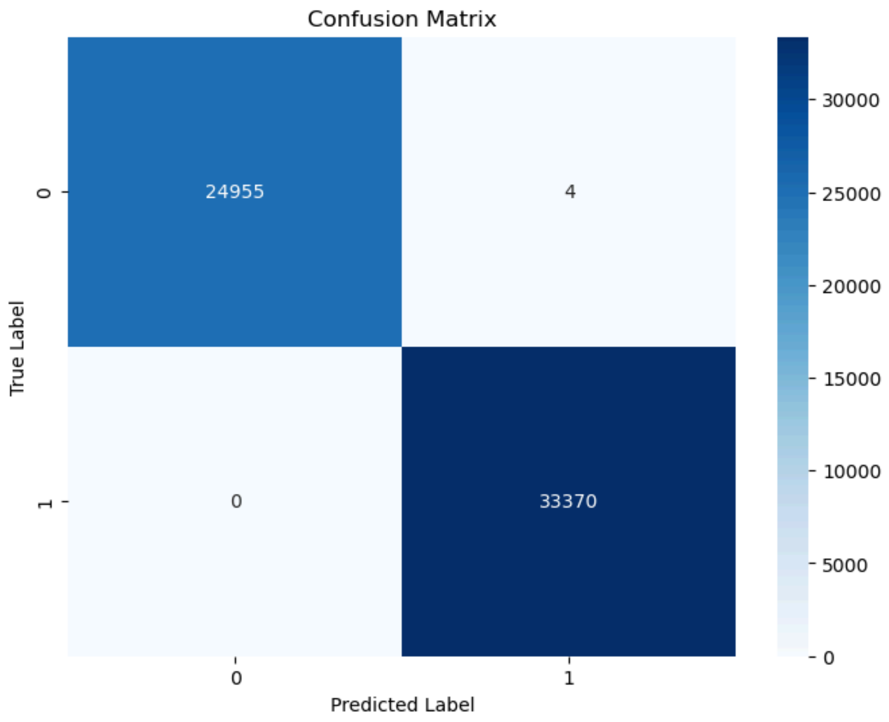
```
Test Accuracy = 0.9999
Test Precision = 0.9999
Test Recall = 0.9999
Test F1 Score = 0.9999
+-----+----------+-----+
|label|prediction|count|
+-----+----------+-----+
|    0|       0.0|24956|
|    1|       1.0|33370|
```

The results were suspicious, so we broke it down into pieces.

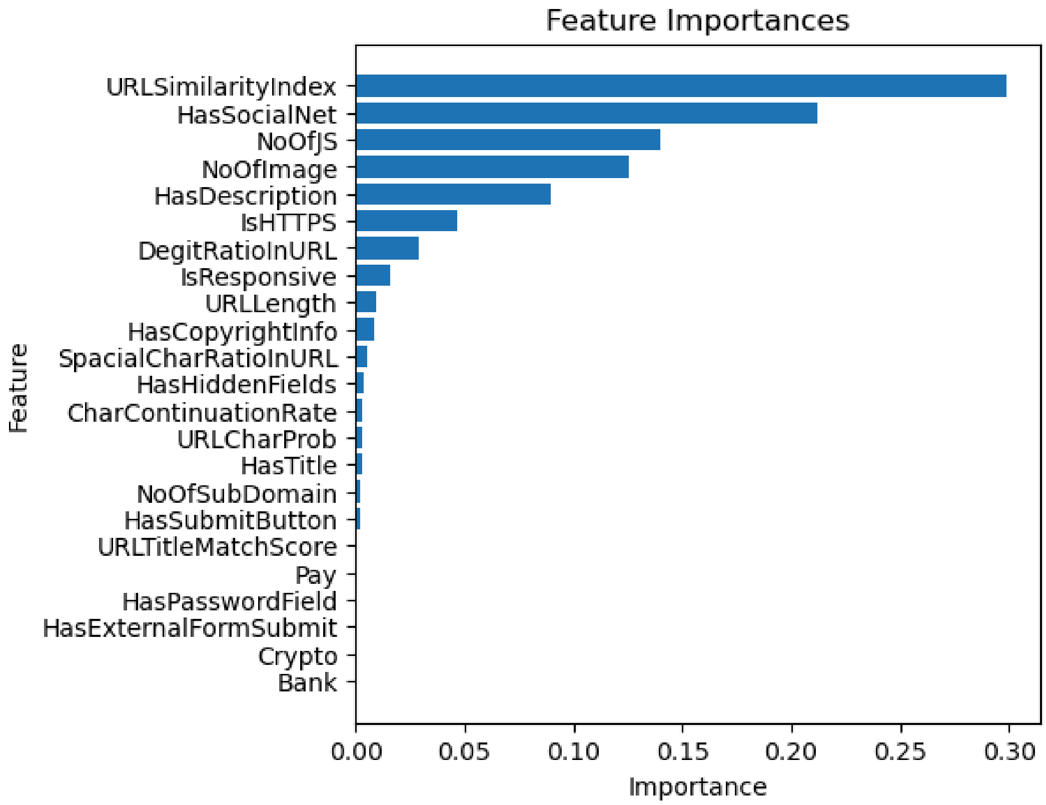
- Training the model using its default parameters, and all the feature columns



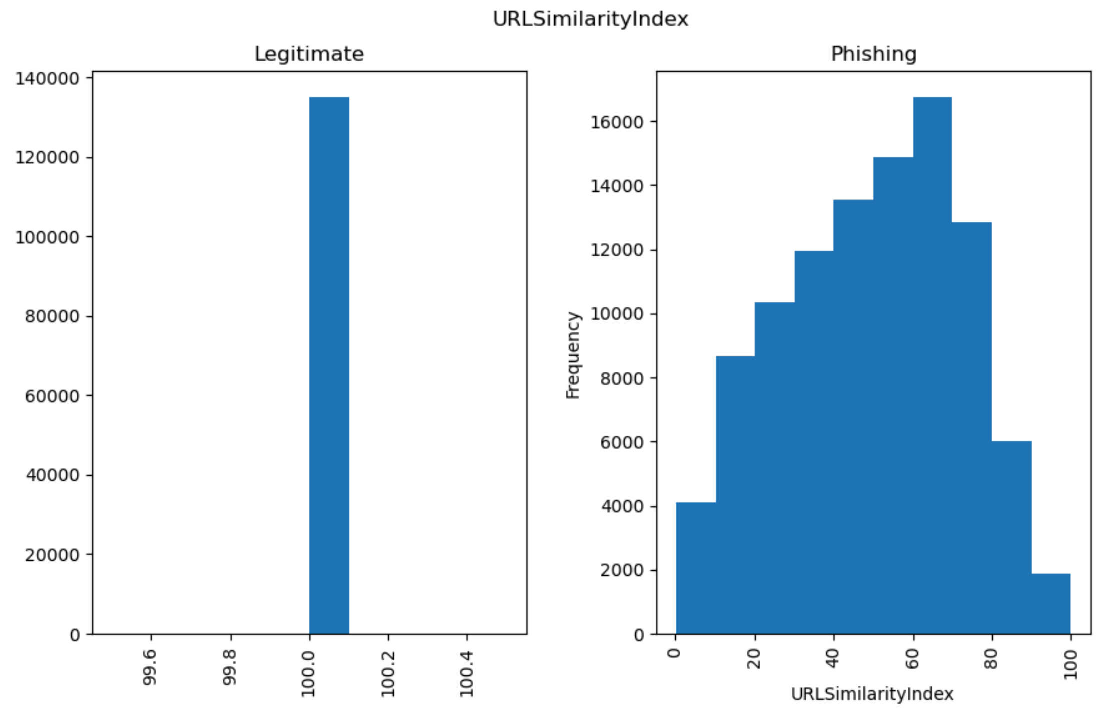
```
Test Accuracy = 0.9999314234771725651 3
Test Precision = 0.99993143169632792144
Test Recall = 0.99993142347717256513
Test F1 Score = 0.99993142278429758552
Validation Accuracy = 0.99991483321504615045
Validation Precision = 0.99991483415042059502
Validation Recall = 0.99991483321504603943
Validation F1 Score = 0.99991483285983306928
```



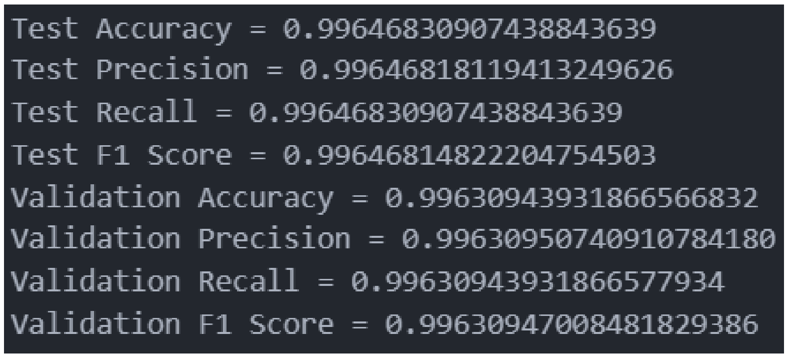
- Plotting the feature importances to see which ones have the most effect



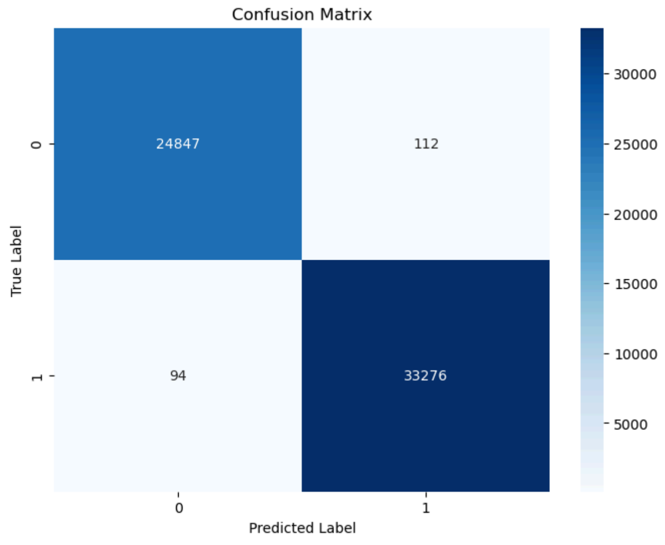
As shown in the above plot, a few features have such a high weight, especially the URLSimilarityIndex.
By referring to the paper that was published regarding the dataset, the URL similarity index is calculated between a source URL and a target URL. The source URL is the given URL for calculating USI. The target URL is a URL from the list of the top 10 million legitimate websites downloaded from Open PageRank Initiative.
And by observing the histogram distributions, the URL similarity index almost completely classifies the URL's correctly, *any* legitimate URL has a similarity index of 100, and a very few number of the phishing ones also have a a similarity index of 100, but that's a very small percentage of the data set.

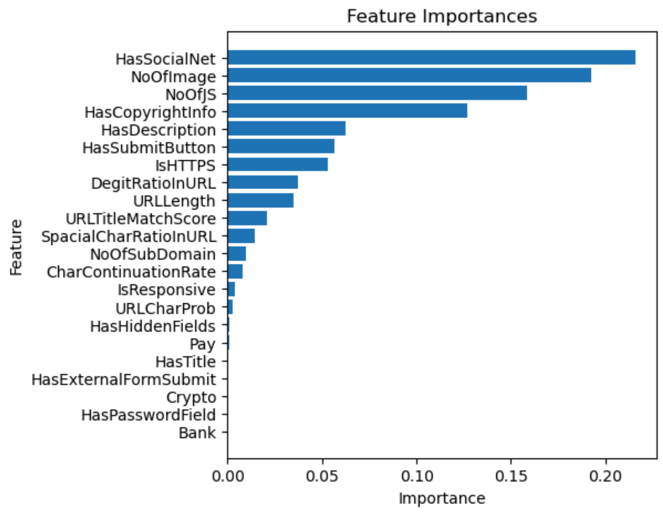


So we tried training the model without that column

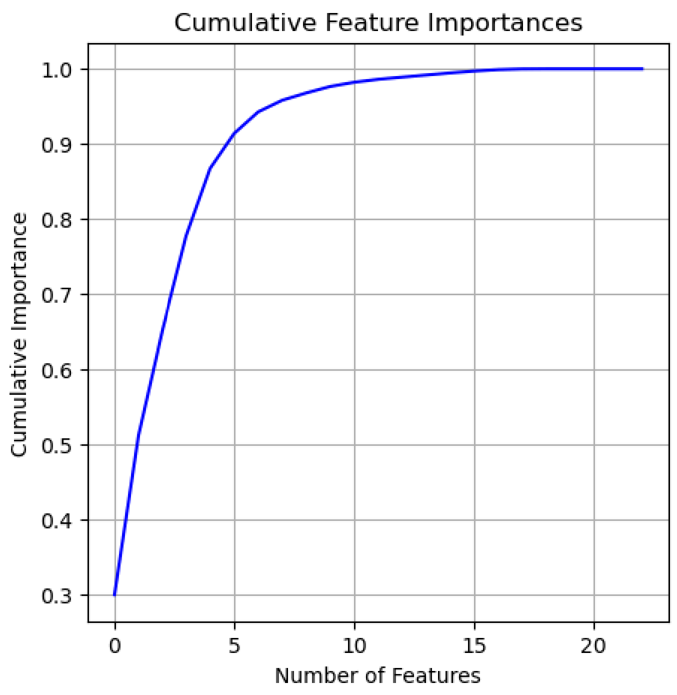


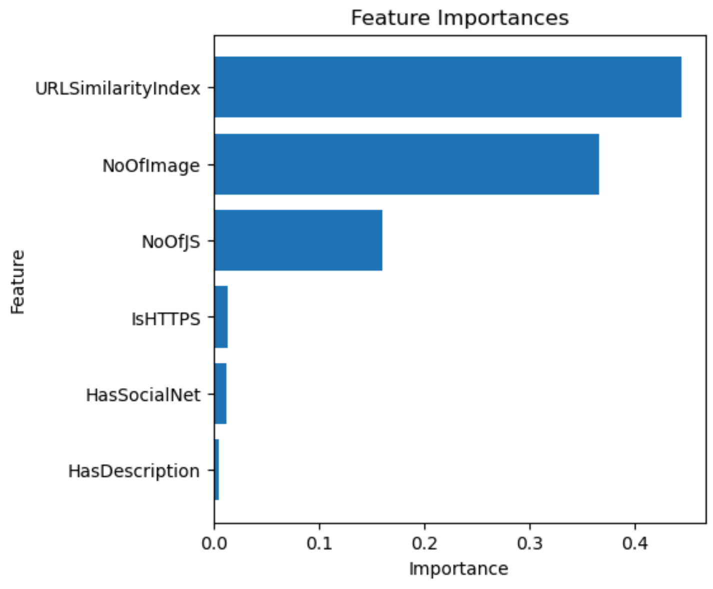
```
Test Accuracy = 0.99646830907438843639
Test Precision = 0.99646818119413249626
Test Recall = 0.99646830907438843639
Test F1 Score = 0.99646814822204754503
Validation Accuracy = 0.99630943931866566832
Validation Precision = 0.99630950740910784180
Validation Recall = 0.99630943931866577934
Validation F1 Score = 0.99630947008481829386
```



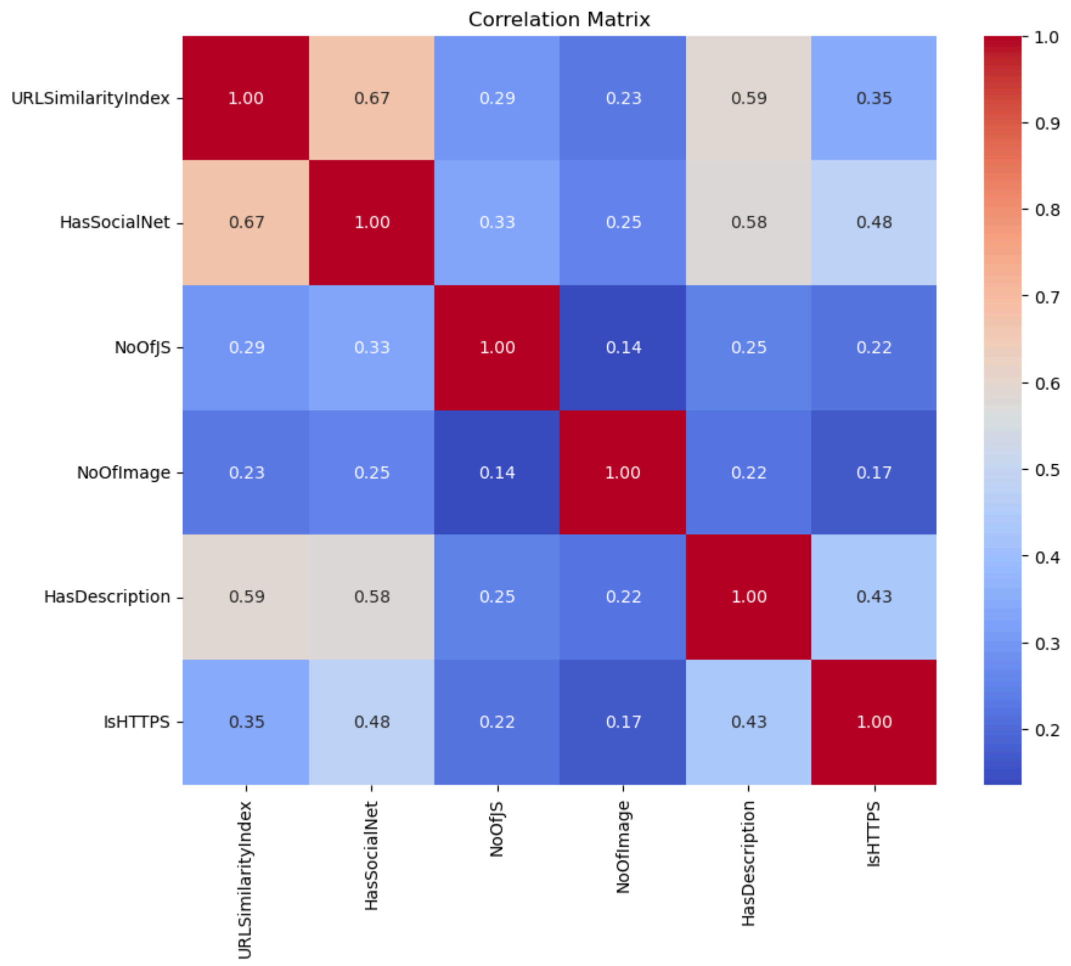


Using all the features, and by plotting the cumulative feature importances, we chose to cap it down to the features that support up to 90% of the importance, which narrowed it to about 6 features. We used those features for the other models.





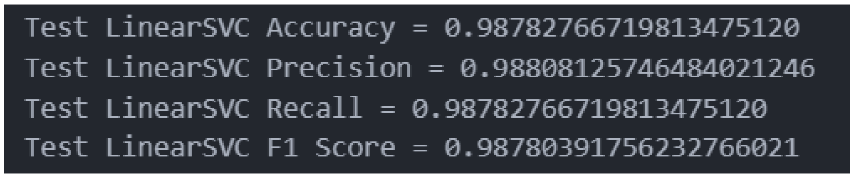
The correlation matrix based on the selected features



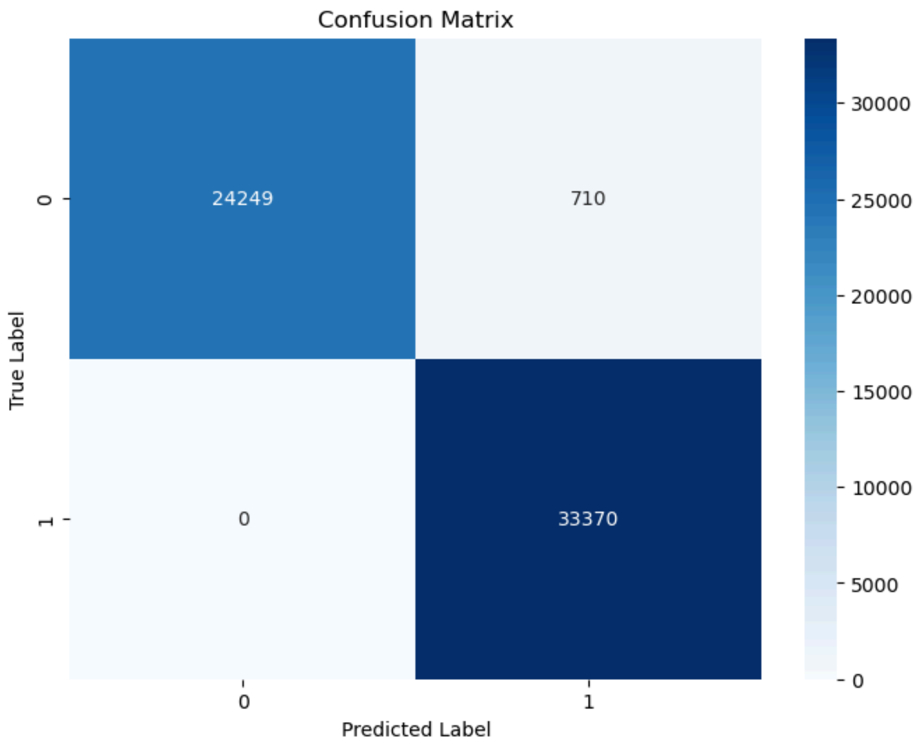
Linear SVM:

Using a Linear Support Vector Machine Classifier with the chosen 6 columns

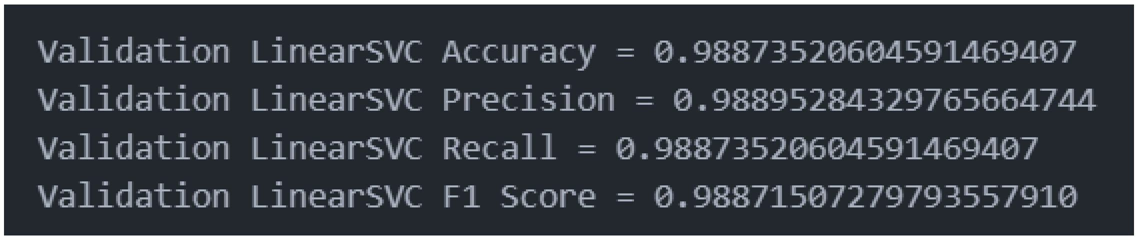
- Test set



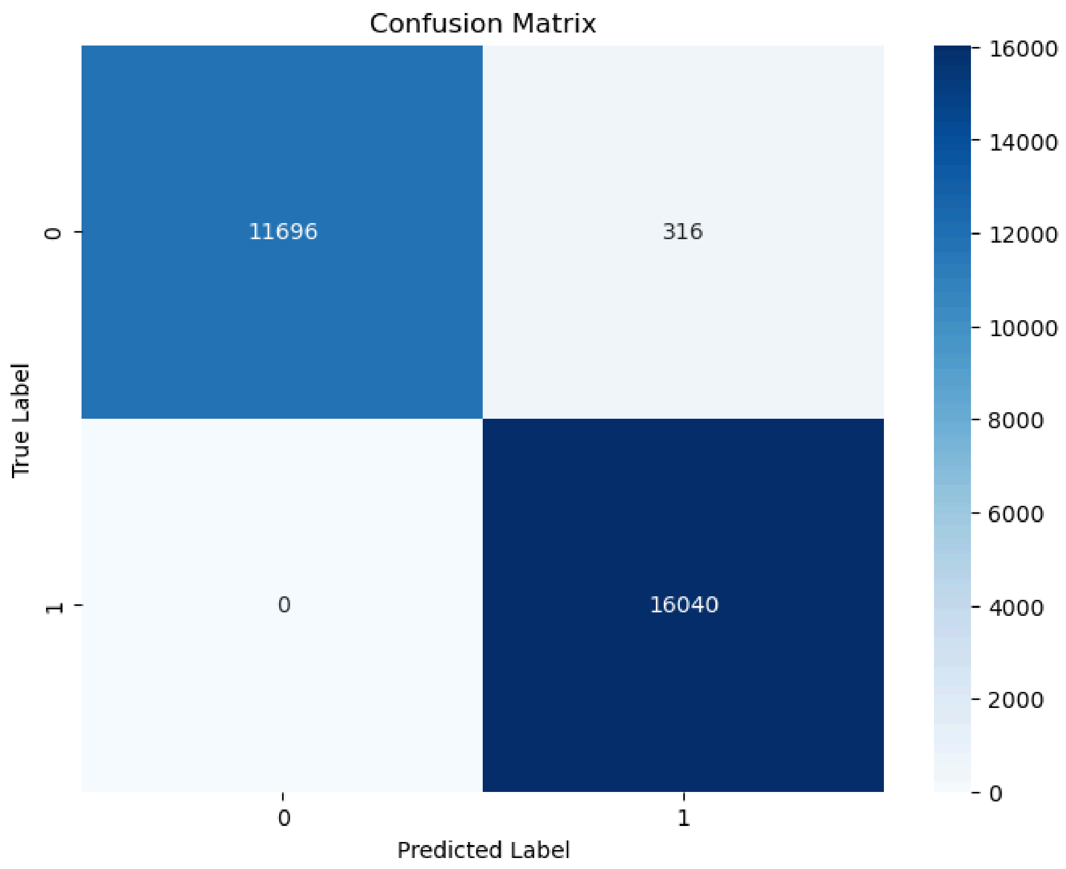
```
Test LinearSVC Accuracy = 0.98782766719813475120
Test LinearSVC Precision = 0.98808125746484021246
Test LinearSVC Recall = 0.98782766719813475120
Test LinearSVC F1 Score = 0.98780391756232766021
```



- Validation set



```
Validation LinearSVC Accuracy = 0.9887352060459146907
Validation LinearSVC Precision = 0.9889528432976566474
Validation LinearSVC Recall = 0.9887352060459146907
Validation LinearSVC F1 Score = 0.9887150727979355791
```



<u>Other Model/Classifiers training:</u>

We used other models such as *KNN and Naive Bayes.*

We also implemented 2 versions one with *MapReduce* and other without to enhance performance.

KNN:

- First we chose the KNN classifier to implement it with and without MapReduce to compare the results.
- The choice was based on the lecture as well as some intuition that for large datasets, since KNN doesn't have a training phase and the classification is made for each test row by calculating the distance between the test row and all the training row and extracting the k nearest neighbors which would consume a lot of time for large datasets.
- The idea was to map the test set as well as the training set but the attempt failed due to reasons that will be discussed in the failed attempt section.
- On the other hand the built-in classifier achieved an accuracy of 99.8% on the test set in seconds.

Naive Bayes:

- The second classifier proposed in the lecture to use MapReduce was Naive Bayes since it also requires to calculate the conditional probability for each value in each feature given each class and to classify the sample based on the maximum posterior probability for each class.
- The approach we took was first to calculate the class priors and the conditional probabilities for each feature for both classes.
- The idea is to use mapping functions to split the dataset and calculate each split on its own then reduce based on the output we want.
- The output of the training phase is a dictionary with an outer key for each class and an nested dictionary where the inner key is the feature index and the conditional probability of the feature value for this class.
- The final phase is to predict which we do by creating a mapping function to calculate the maximum posterior probability of the test sample given both classes and returns the actual and predicted classification.

Results and Evaluation:

Random Forest Model Accuracy:

Test: *99.64683%* Validation: *99.69623%*

SVM Model Accuracy:

Train: *98.85%* Test: *98.78%* Validation: *98.87%*

KNN Model Accuracy:

Test: 98.62 *%*

Naive Bayes Model accuracy:

Without MapReduce:

Test: *99.98%*

With MapReduce:

Test: *92%*

Unsuccessful trials:

MapReduce Attempts:

- When trying to implement the KNN using MapReduce we were able to either map the training dataset to calculate the distances or map the test set but not both since we can’t apply two nested RDD transformations.
- So it was either loop on all test rows and map the training data with each test row or map the test set and loop on all training data to calculate the similarity between them.
- Since the dataset is large both implementations would consume a huge amount of time to classify all the points in the test set.
- After some computations we came to the conclusion that in order to classify all the points in the test set it would require approximately 200 hours which is not worth it since the built-in KNN classifier trained the data in seconds with accuracy 99.9%.

Enhancements & Future work:

Using a larger dataset that covers more diverse phishing URL’s, extracting more descriptive features and using more models for comparison purposes.