PG AIML - AI and Machine Learning Capstone Project

Course-End Project Problem Statement

simplilearn

Get Certified. Get Ahead.

<u>Course-End Project: Cyber Security</u>

<u>Problem statement:</u>
Book-My-Show will enable the ads on their website, but they are also very cautious about their user privacy and information who visit their website. Some ads URL could contain a malicious link that can trick any recipient and lead to a malware installation, freezing the system as part of a ransomware attack or revealing sensitive information. Book-My-Show now wants to analyze that whether the particular URL is prone to phishing (malicious) or not.

<u>Dataset description:</u>
Dataset name: **dataset.csv**

The input dataset contains an 11k sample corresponding to the 11k URL. Each sample contains 32 features that give a different and unique description of URL ranging from -1,0,1.
1: Phishing
0: Suspicious
1: Legitimate
The sample could be either legitimate or phishing.

<u>Task to be performed:</u>
Exploratory Data Analysis:

1. Each sample has 32 features ranging from -1,0,1. Explore the data using histogram, heatmaps.
2. Determine the number of samples present in the data, unique elements in all the features.
3. Check if there is any null value in any features.

Correlation of features and feature selection:
4. Next, we have to find if there are any correlated features present in the data. Remove the feature which might be correlated with some threshold.

Building Classification Model

1. Finally, build a robust classification system that classifies whether the URL sample is a phishing site or not.

- Build classification models using a binary classifier to detect malicious or phishing URLs.
- Illustrate the diagnostic ability of this binary classifier by plotting the ROC curve.
- Validate the accuracy of data by the K-Fold cross-validation technique.
- The final output consists of the model, which will give maximum accuracy on the validation dataset with selected attributes.

<u>Solution :</u>

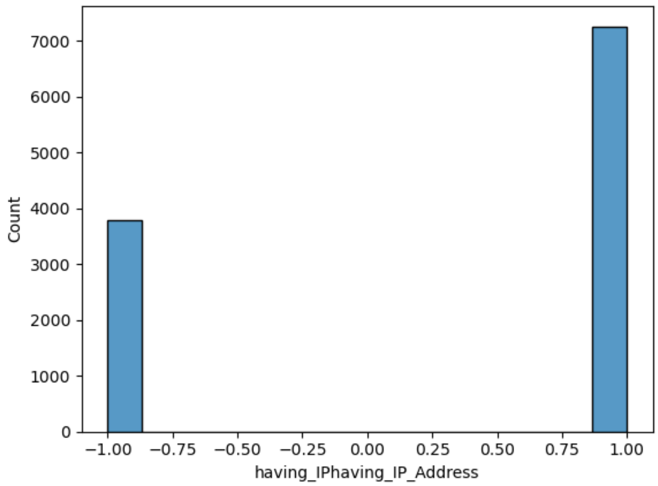
Exploratory Data Analysis:

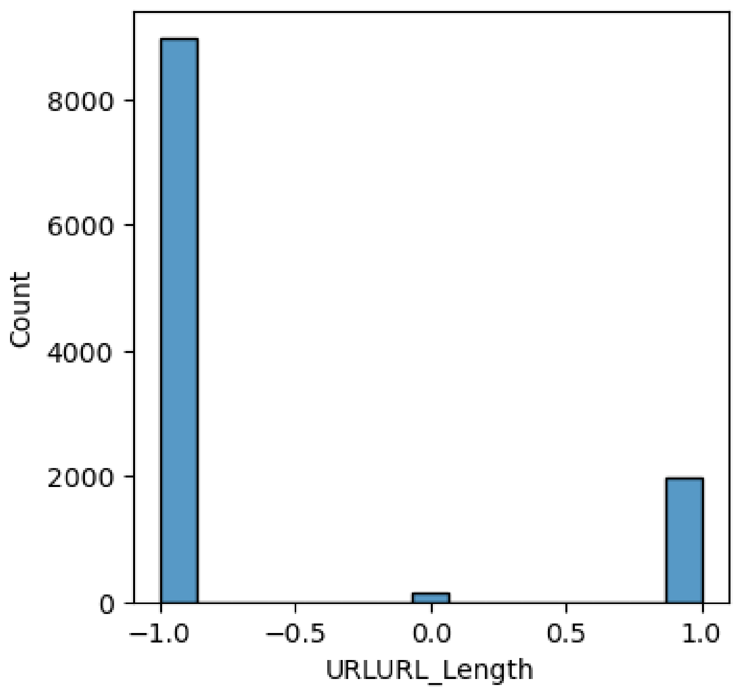
– importing and reading dataset

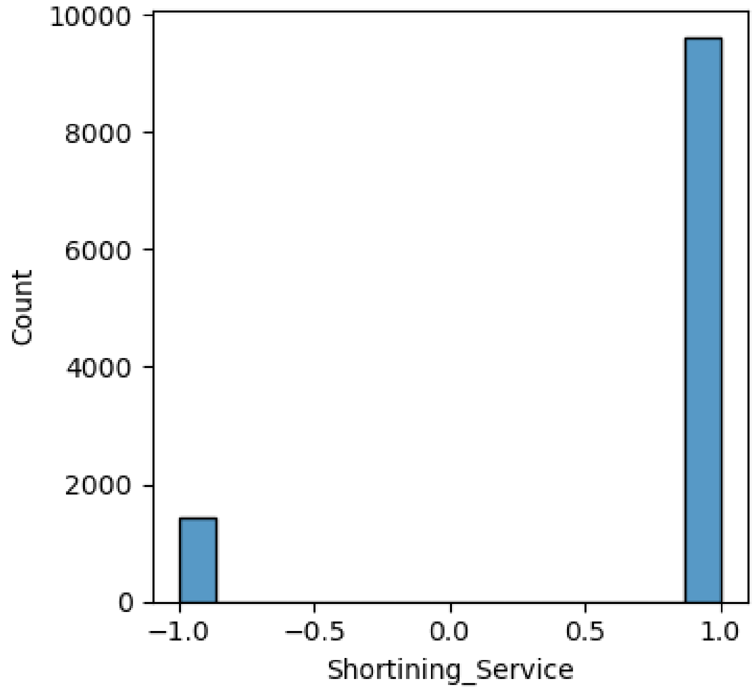
```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')

for each in os.listdir():
    print(each)

df= pd.read_csv("dataset.csv")
df

df.head()
```

1. Each sample has 32 features ranging from -1,0,1. Explore the data using histogram, heatmaps.

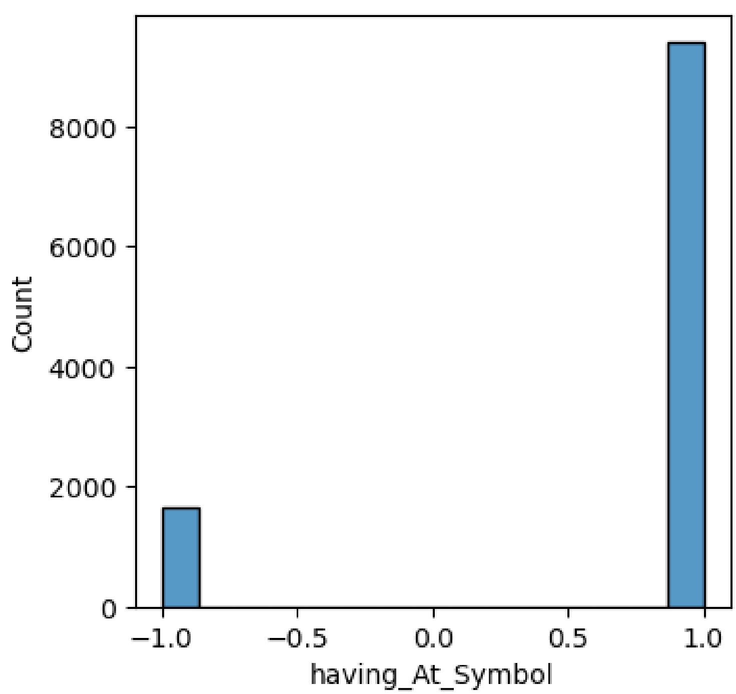
```
df1 = df.drop('index', axis=1)
for i, col in enumerate(df1.columns):
    plt.figure(figsize=(4,4))
    plt.figure(i)
    sns.histplot(df1[col])
    sns.histplot()
```



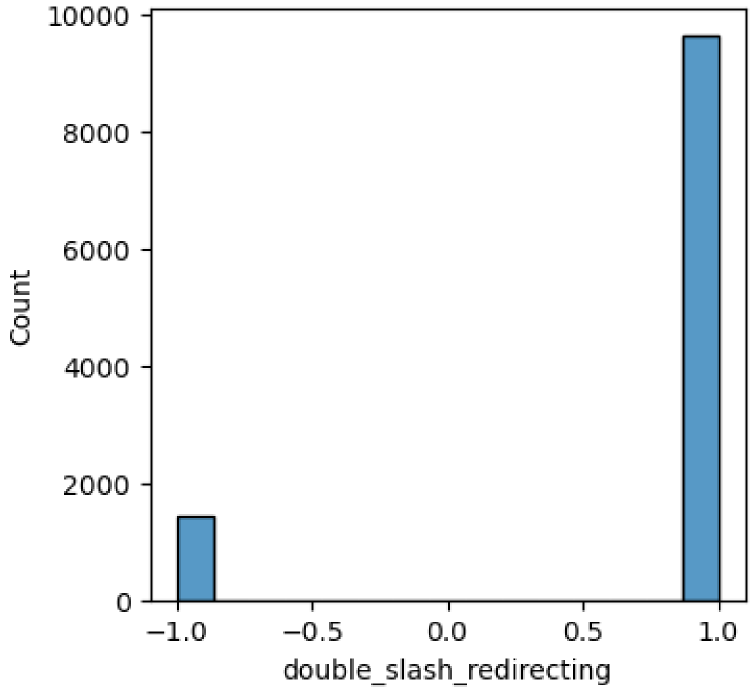




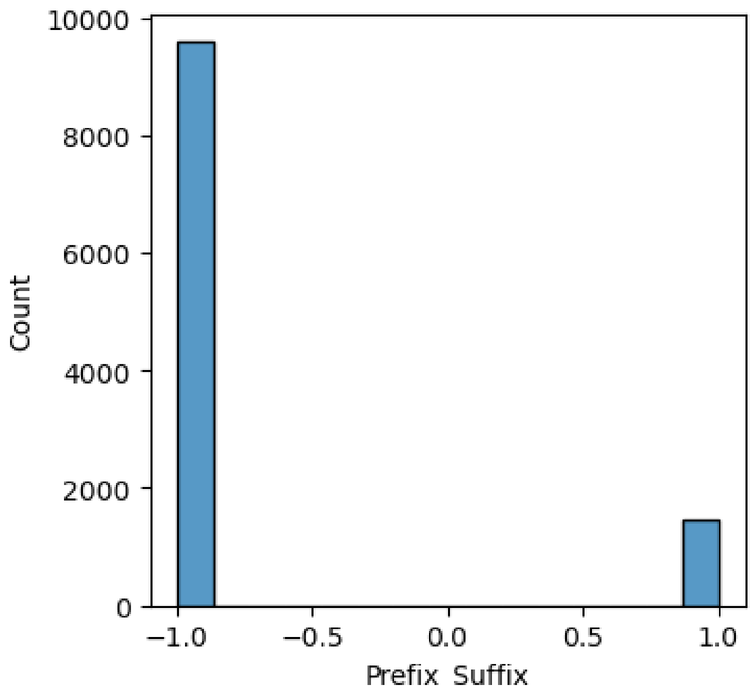
10000
8000
6000
4000
2000
0
Count
−1.0
−0.5
0.0
0.5
1.0
Shortining_Service



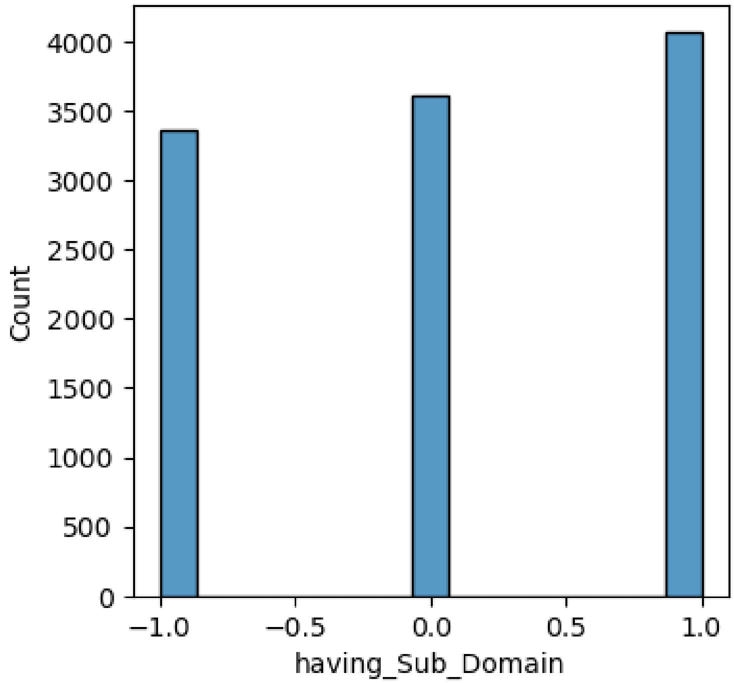
8000
6000
4000
2000
0
Count
−1.0
−0.5
0.0
0.5
1.0
having_At_Symbol



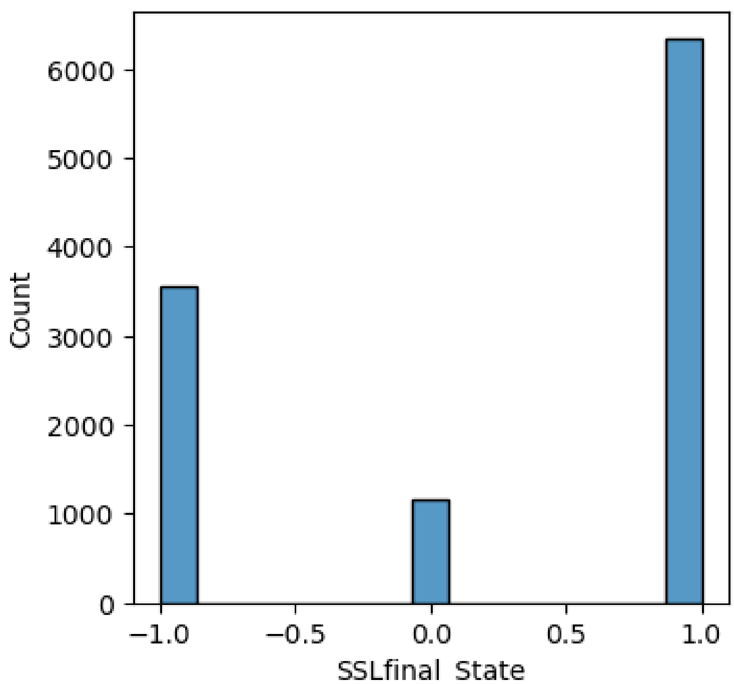
10000
8000
6000
4000
2000
0
Count
−1.0
−0.5
0.0
0.5
1.0
double_slash_redirecting



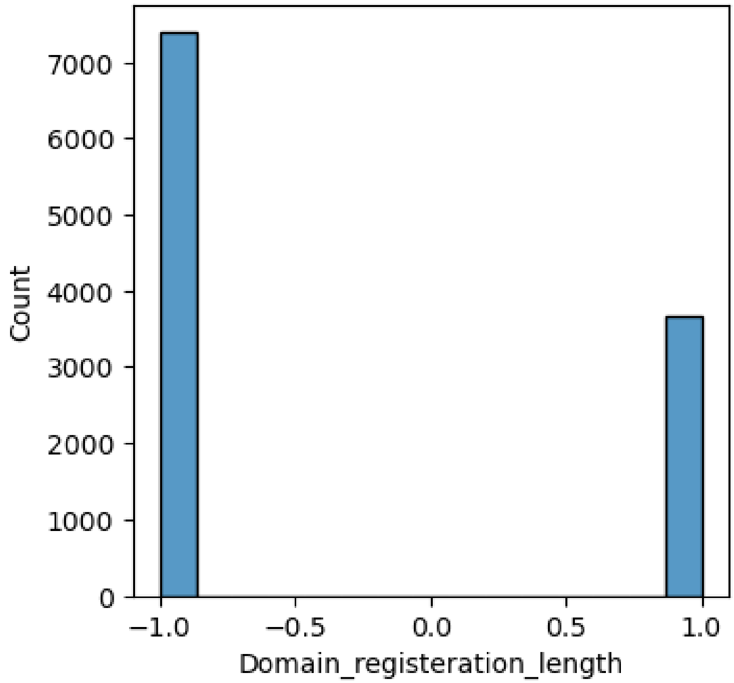
10000
8000
6000
4000
2000
0
Count
−1.0
−0.5
0.0
0.5
1.0
Prefix_Suffix

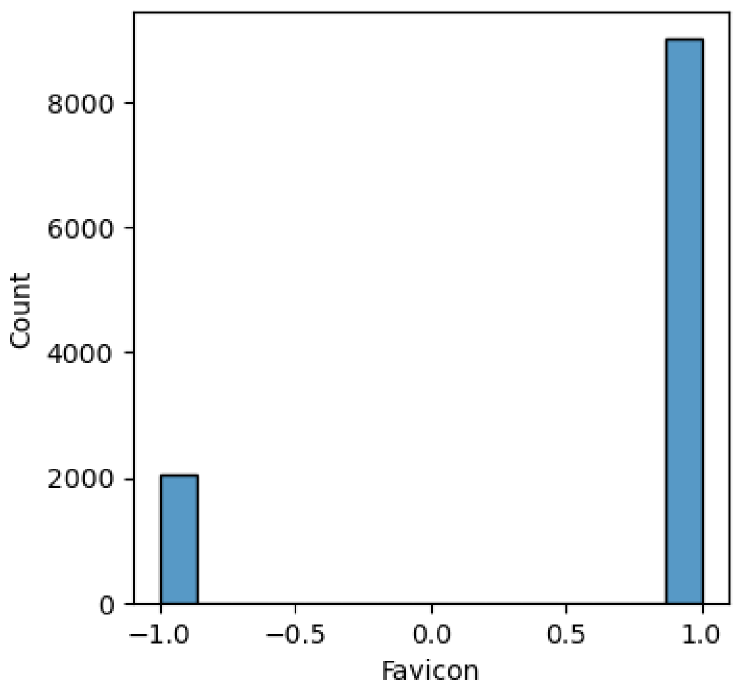


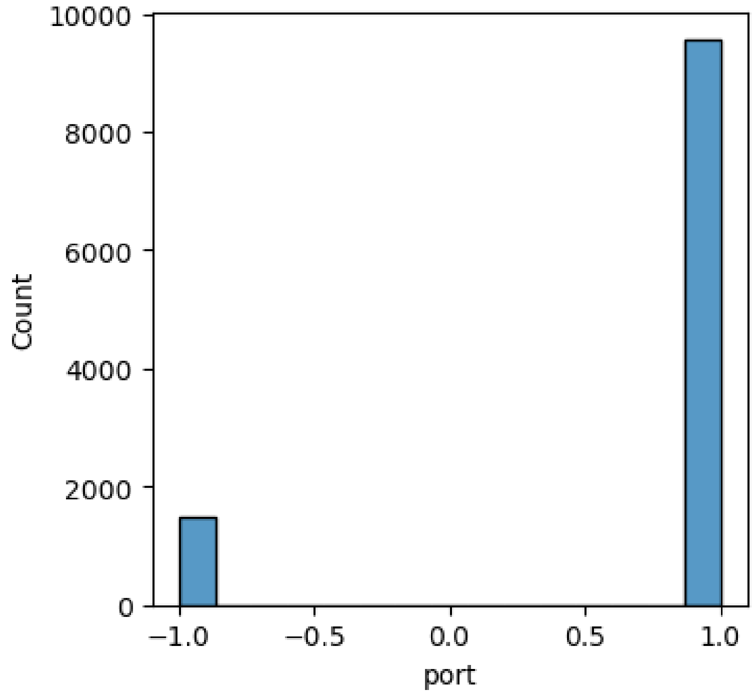
4000
3500
3000
2500
2000
1500
1000
500
0
Count
−1.0
−0.5
0.0
0.5
1.0
having_Sub_Domain



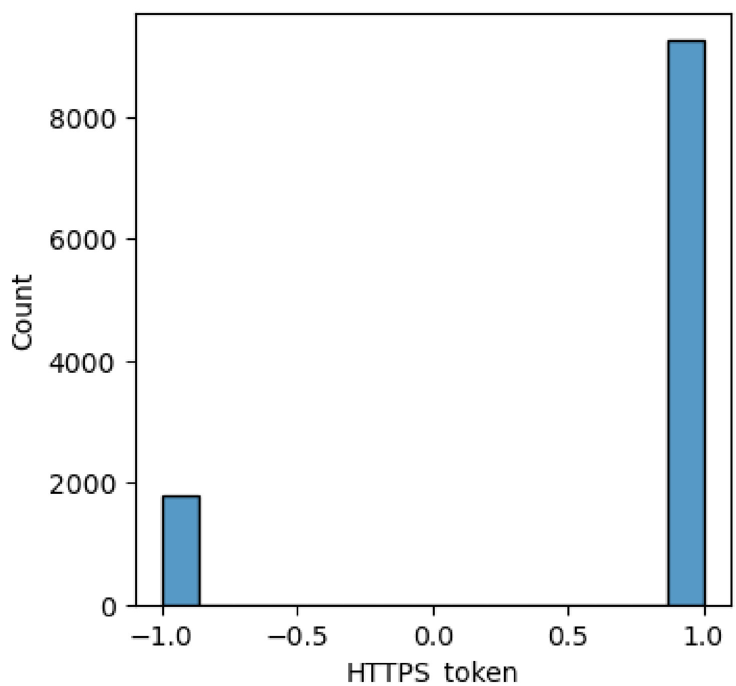
6000
5000
4000
3000
2000
1000
0
Count
−1.0
−0.5
0.0
0.5
1.0
SSLfinal_State


Count
7000
6000
5000
4000
3000
2000
1000
0
−1.0
−0.5
0.0
0.5
1.0
Domain_registeration_length

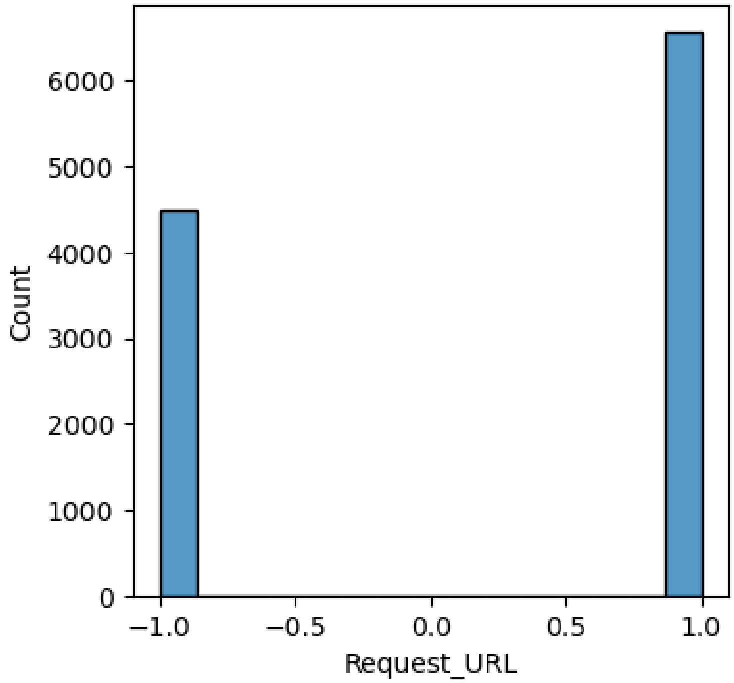

Count
8000
6000
4000
2000
0
−1.0
−0.5
0.0
0.5
1.0
Favicon



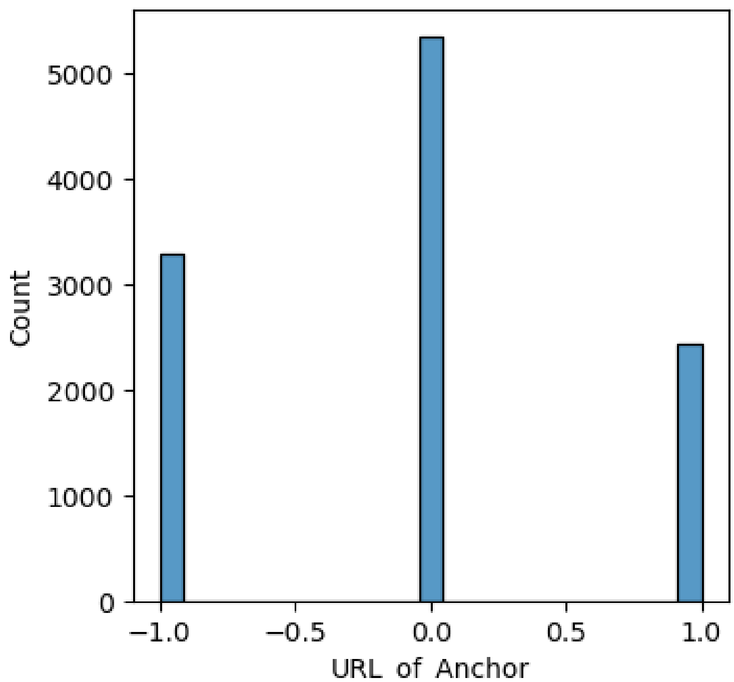
10000
8000
6000
4000
2000
0
Count
−1.0
−0.5
0.0
0.5
1.0
port



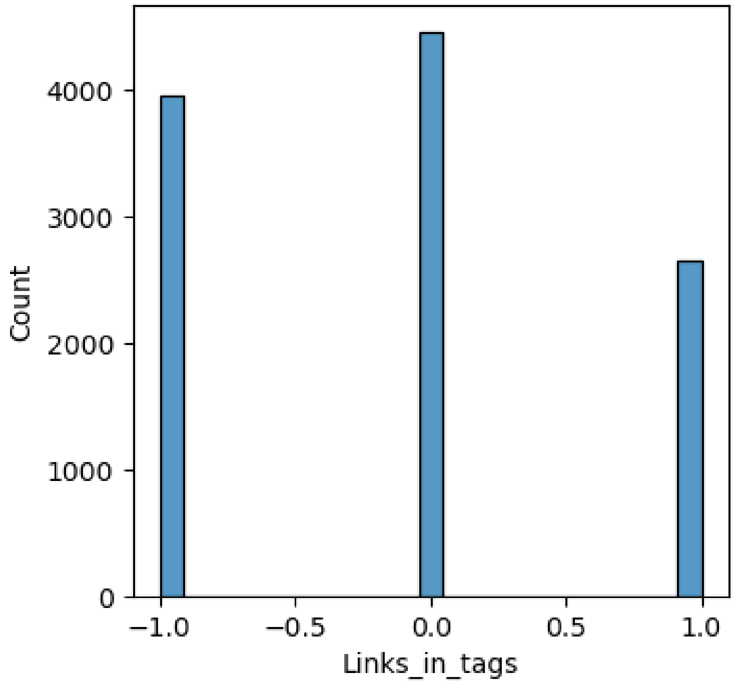
8000
6000
4000
2000
0
Count
−1.0
−0.5
0.0
0.5
1.0
HTTPS_token



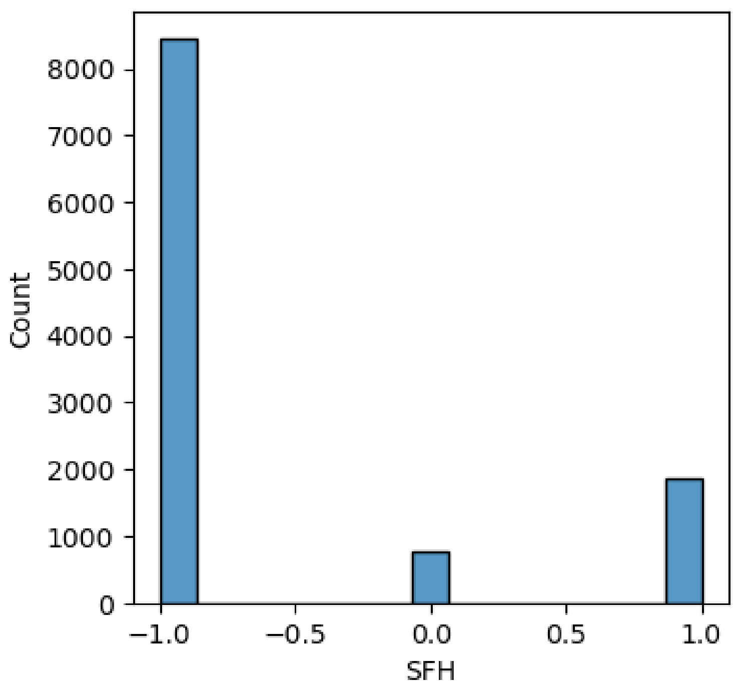
6000
5000
4000
3000
2000
1000
0
Count
−1.0
−0.5
0.0
0.5
1.0
Request_URL



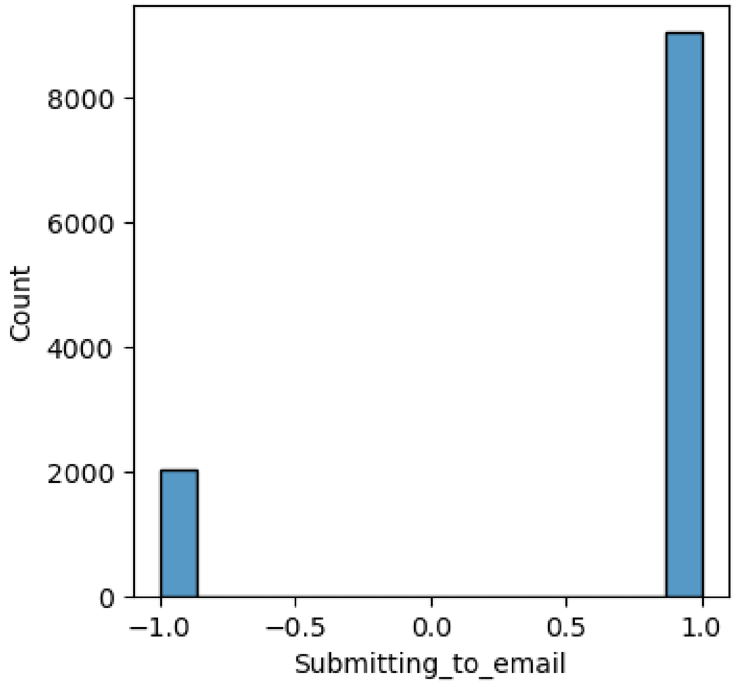
5000
4000
3000
2000
1000
0
Count
−1.0
−0.5
0.0
0.5
1.0
URL_of_Anchor



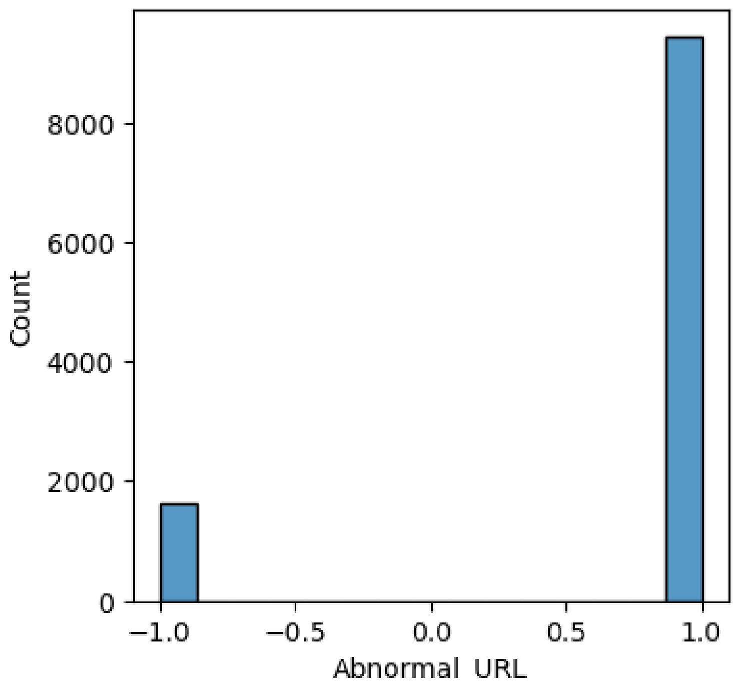
4000
3000
2000
1000
0
Count
−1.0
−0.5
0.0
0.5
1.0
Links_in_tags



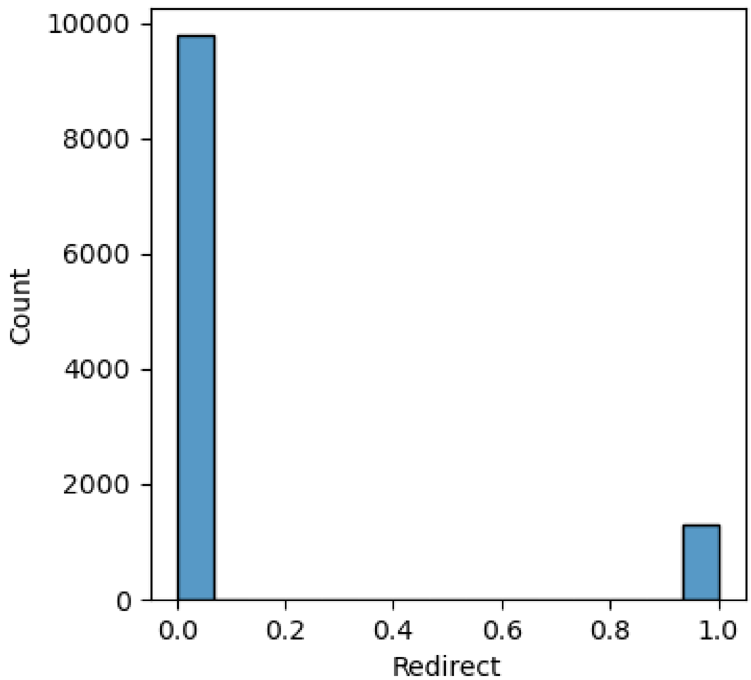
8000
7000
6000
5000
4000
3000
2000
1000
0
Count
−1.0
−0.5
0.0
0.5
1.0
SFH

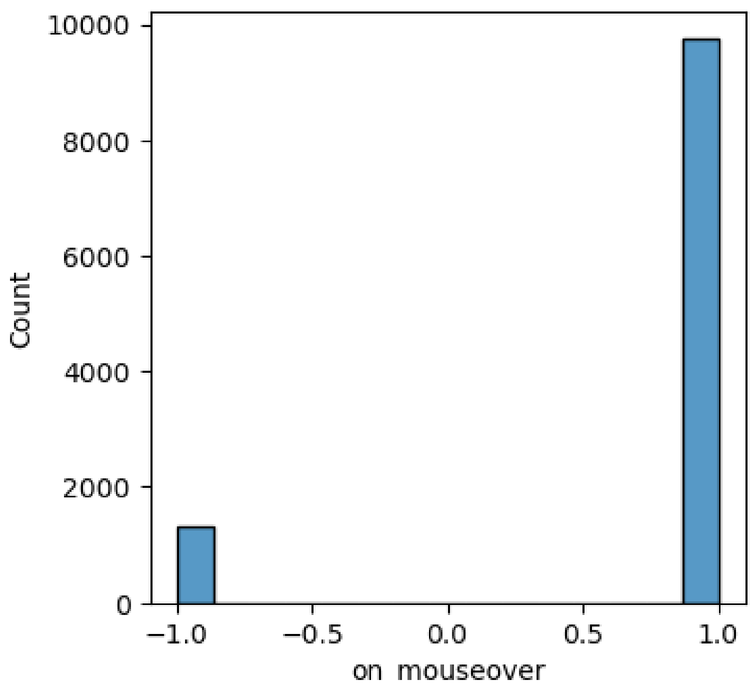


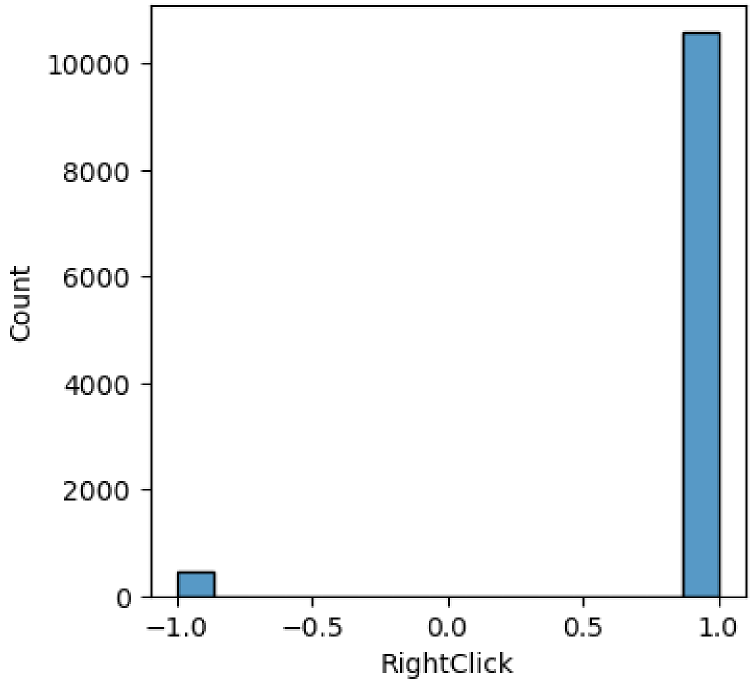
8000
6000
Count
4000
2000
0
−1.0
−0.5
0.0
0.5
1.0
Submitting_to_email

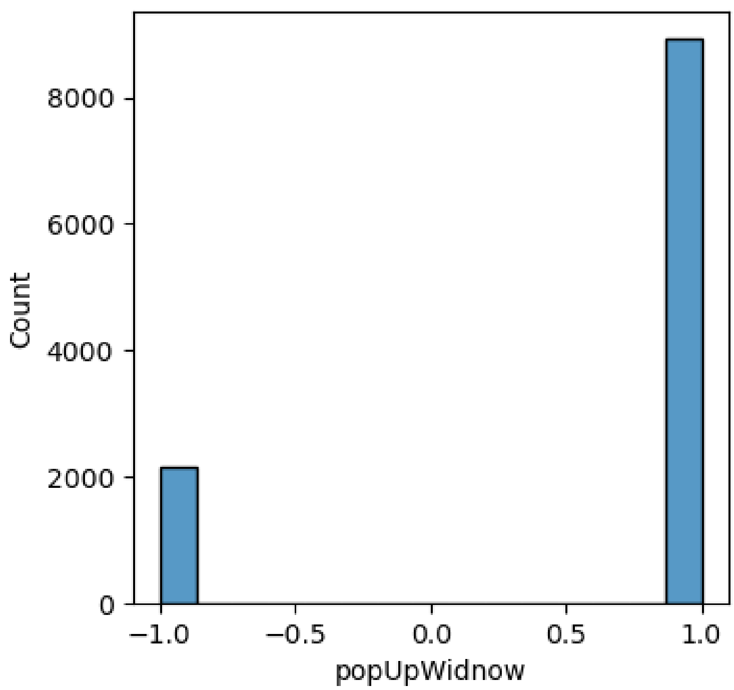


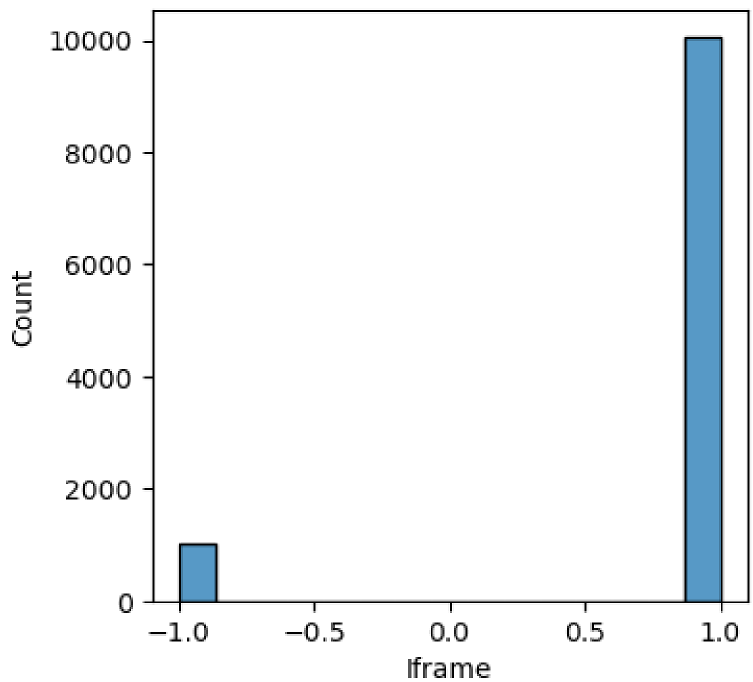
8000
6000
Count
4000
2000
0
−1.0
−0.5
0.0
0.5
1.0
Abnormal_URL

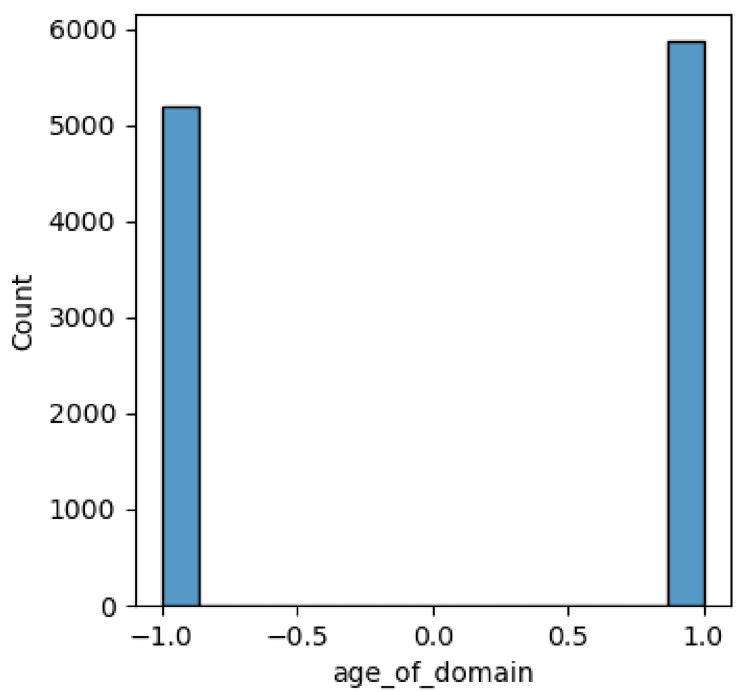

Count
Redirect
0
2000
4000
6000
8000
10000
0.0
0.2
0.4
0.6
0.8
1.0

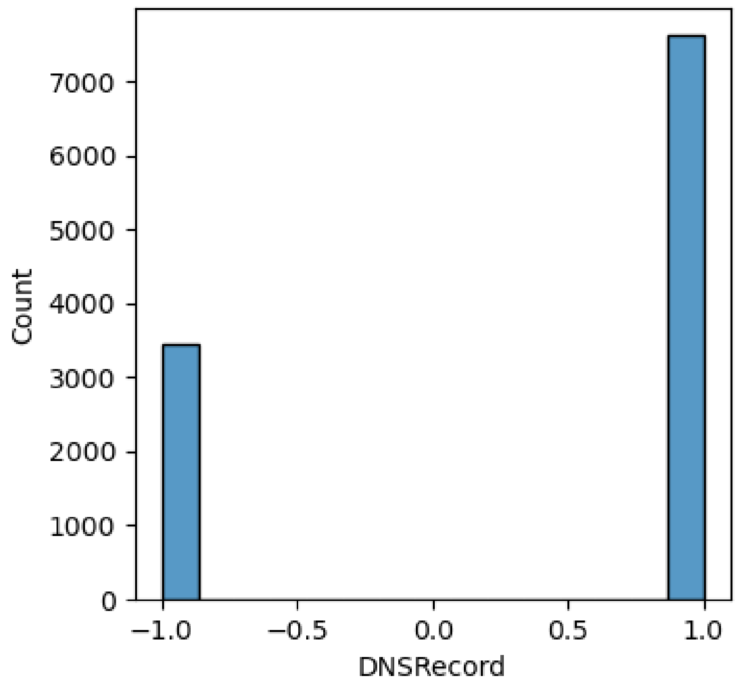

Count
on_mouseover
0
2000
4000
6000
8000
10000
−1.0
−0.5
0.0
0.5
1.0


10000
8000
6000
4000
2000
0
Count
−1.0
−0.5
0.0
0.5
1.0
RightClick

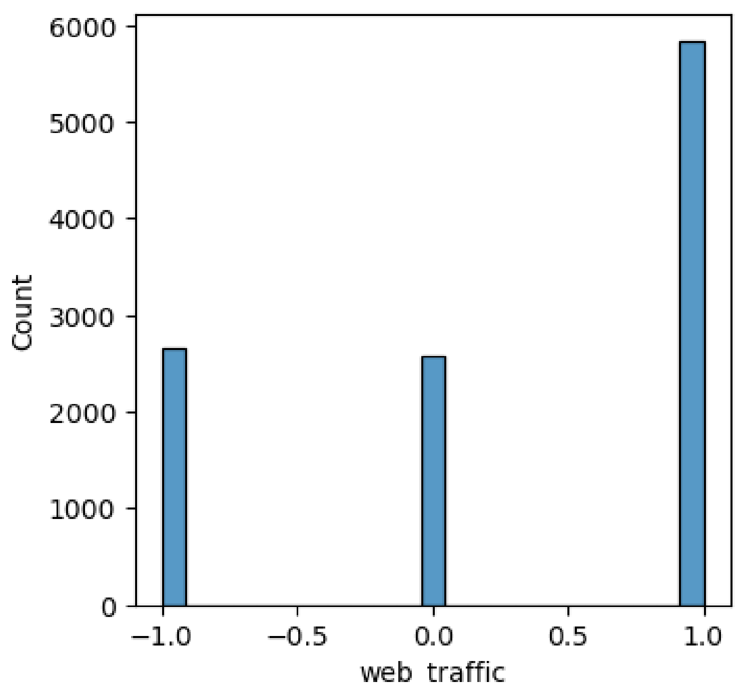

8000
6000
4000
2000
0
Count
−1.0
−0.5
0.0
0.5
1.0
popUpWidnow


10000
8000
6000
4000
2000
0
Count
−1.0
−0.5
0.0
0.5
1.0
Iframe

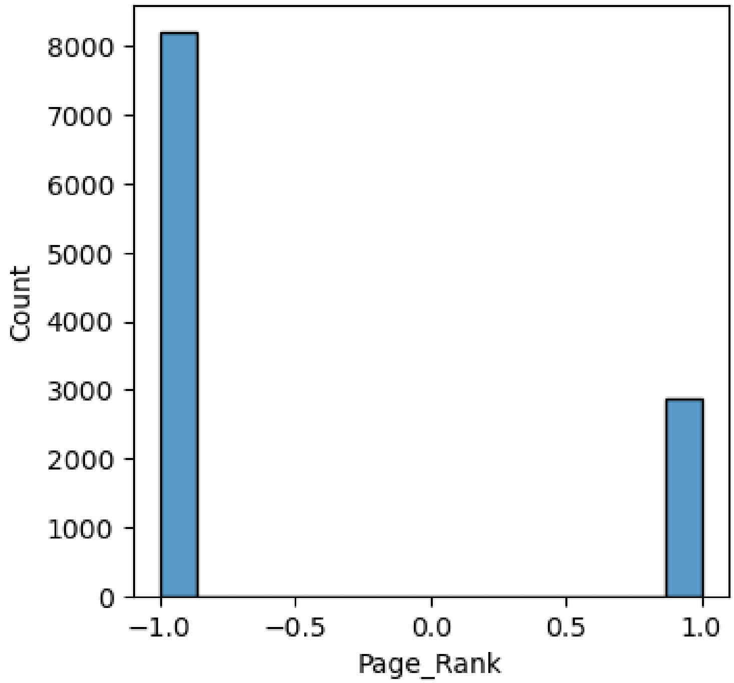

6000
5000
4000
3000
2000
1000
0
Count
−1.0
−0.5
0.0
0.5
1.0
age_of_domain



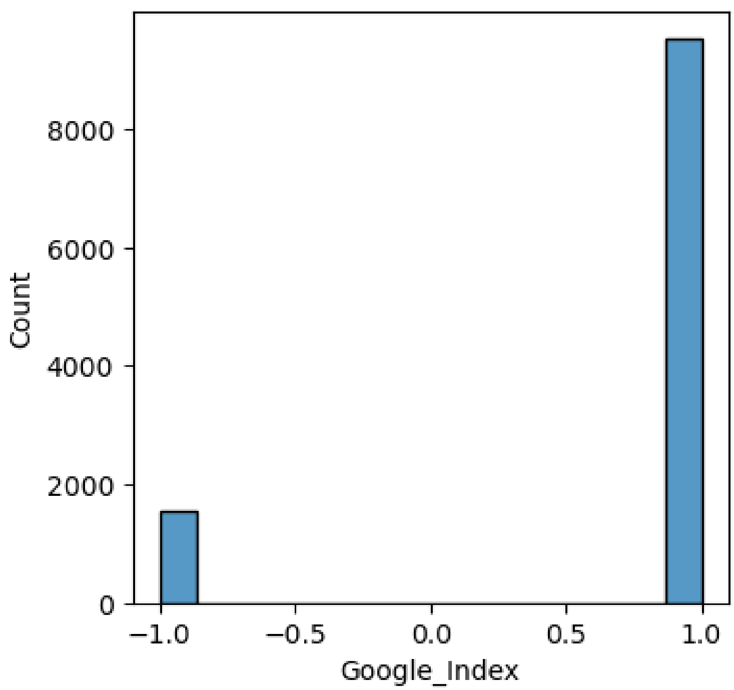
Count
DNSRecord



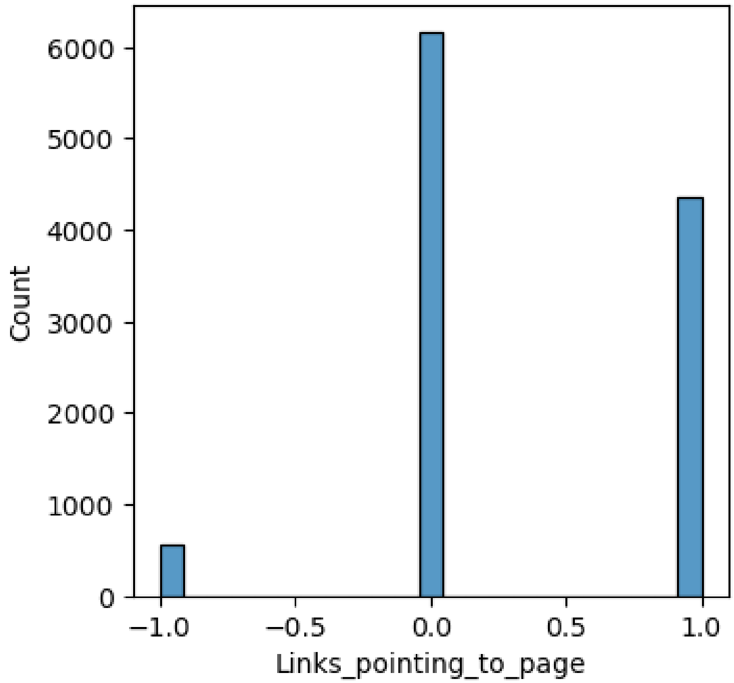
Count
web_traffic



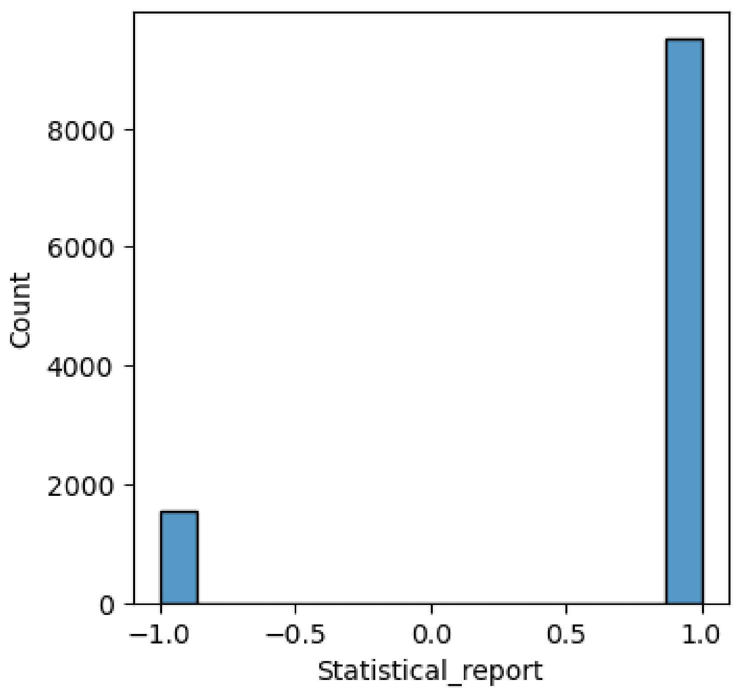
8000
7000
6000
5000
4000
3000
2000
1000
0
Count
−1.0
−0.5
0.0
0.5
1.0
Page_Rank



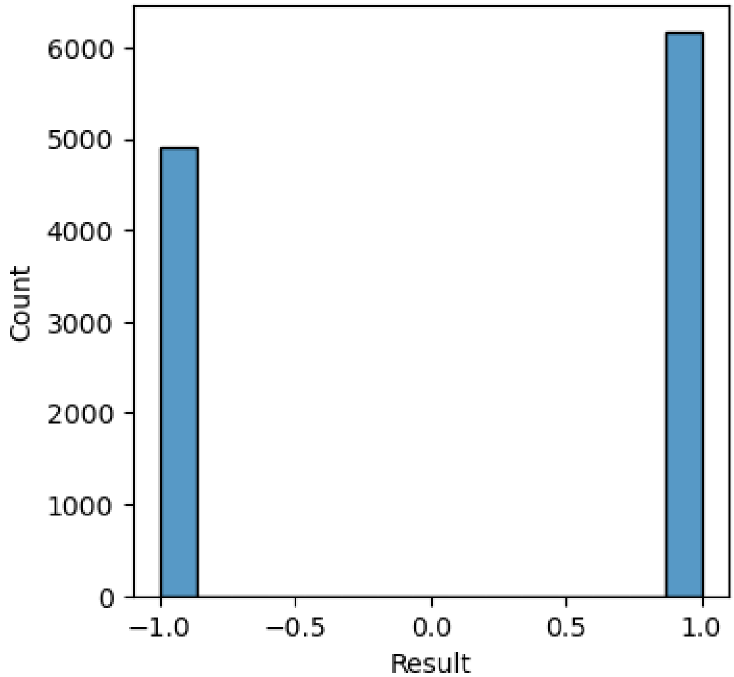
8000
6000
4000
2000
0
Count
−1.0
−0.5
0.0
0.5
1.0
Google_Index



Count
Links_pointing_to_page

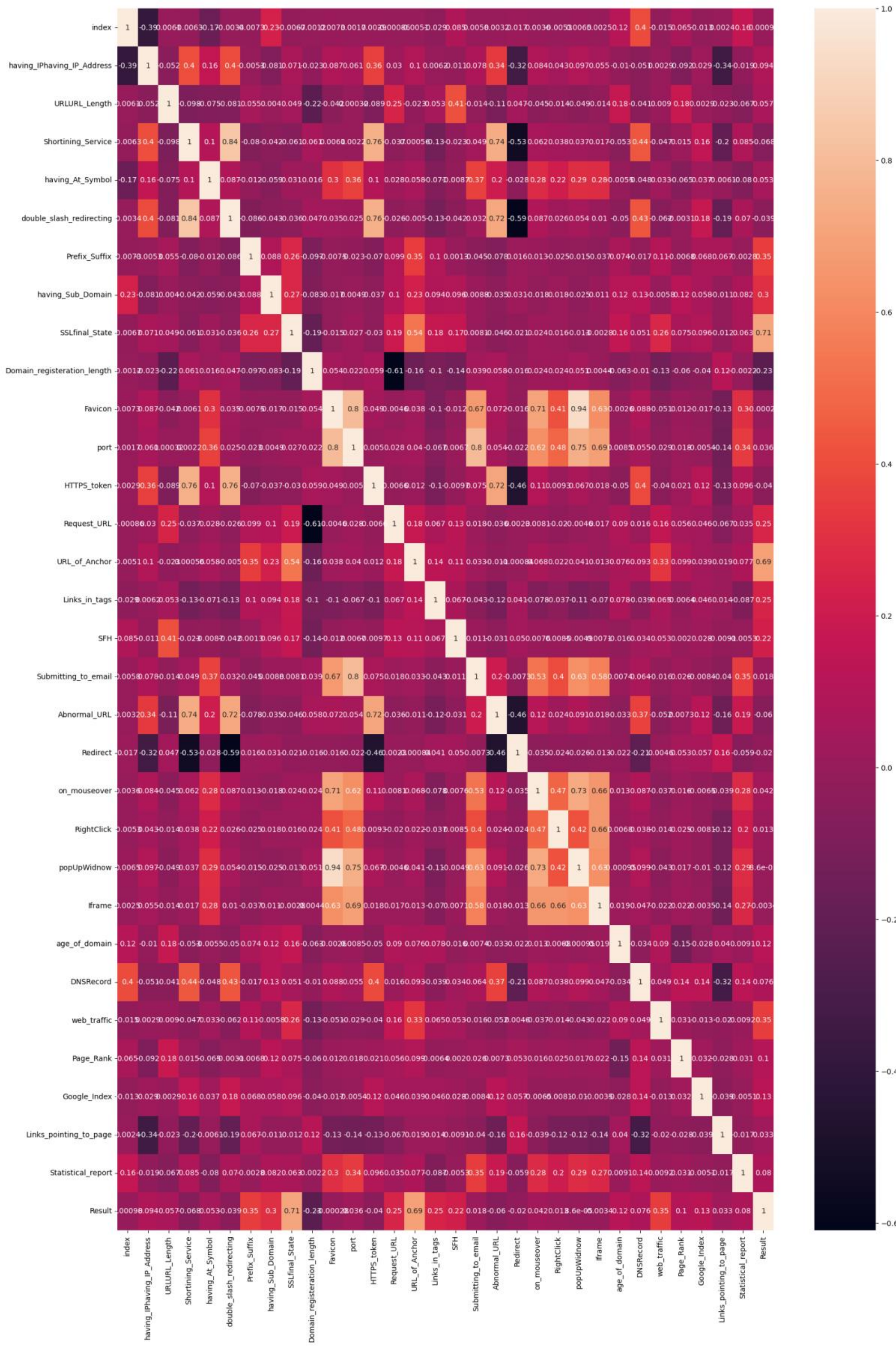


Count
Statistical_report



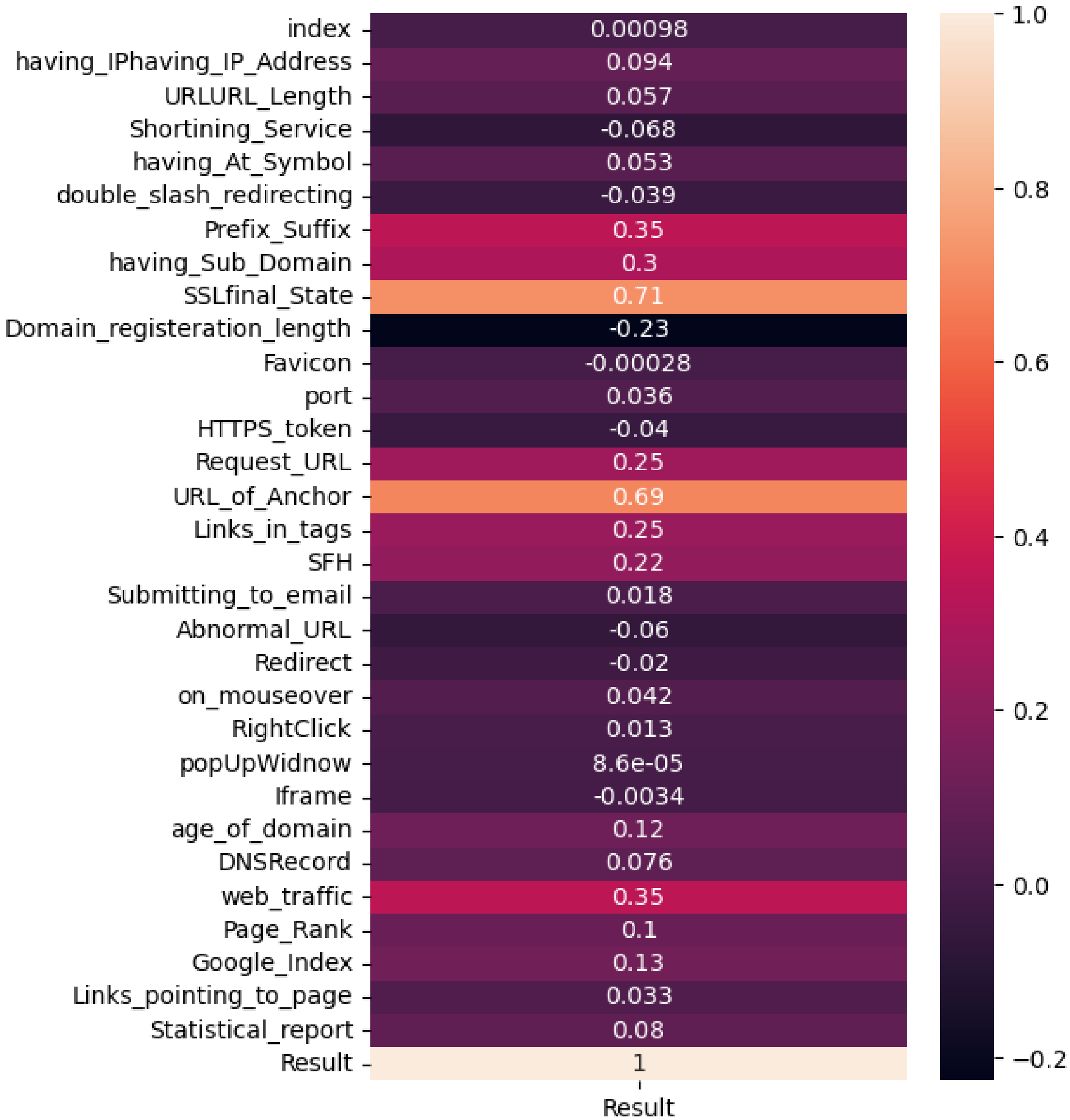
Features 'URLURL_Length', 'Prefix_Suffix',' Domain_registeration_length', 'SFH', 'Page_Rank' have more impact on value -1 and the rest features have high impact on value 1.
Features 'URL_of_Anchor', 'Links_in_tags' has more impact on value 0 and then -1 and has less impact on 1.

```
plt.figure(figsize = (20,30))
sns.heatmap(data = df.corr(), annot = True)
plt.show()
```



index
having_IPhaving_IP_Address
URLURL_Length
Shortining_Service
having_At_Symbol
double_slash_redirecting
Prefix_Suffix
having_Sub_Domain
SSLfinal_State
Domain_registeration_length
Favicon
port
HTTPS_token
Request_URL
URL_of_Anchor
Links_in_tags
SFH
Submitting_to_email
Abnormal_URL
Redirect
on_mouseover
RightClick
popUpWidnow
Iframe
age_of_domain
DNSRecord
web_traffic
Page_Rank
Google_Index
Links_pointing_to_page
Statistical_report
Result
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6

```
plt.figure(figsize = (5,8))
sns.heatmap(df.corr()[['Result']], annot = True)
```



Feature 'SSLfinal_State' has high correlation 0.71 with 'Result'.

2. Determine the number of samples present in the data, unique elements in all the features.

```
df.shape
```

```
df.shape
```

```
(11055, 32)
```

df.describe().T

```
df.describe().T
```

	count	mean	std	min	25%	50%	75%	max
index	11055.0	5528.000000	3191.447947	1.0	2764.5	5528.0	8291.5	11055.0
having_IPhaving_IP_Address	11055.0	0.313795	0.949534	-1.0	-1.0	1.0	1.0	1.0
URLURL_Length	11055.0	-0.633198	0.766095	-1.0	-1.0	-1.0	-1.0	1.0
Shortining_Service	11055.0	0.738761	0.673998	-1.0	1.0	1.0	1.0	1.0
having_At_Symbol	11055.0	0.700588	0.713598	-1.0	1.0	1.0	1.0	1.0
double_slash_redirecting	11055.0	0.741474	0.671011	-1.0	1.0	1.0	1.0	1.0
Prefix_Suffix	11055.0	-0.734962	0.678139	-1.0	-1.0	-1.0	-1.0	1.0
having_Sub_Domain	11055.0	0.063953	0.817518	-1.0	-1.0	0.0	1.0	1.0
SSLfinal_State	11055.0	0.250927	0.911892	-1.0	-1.0	1.0	1.0	1.0
Domain_registeration_length	11055.0	-0.336771	0.941629	-1.0	-1.0	-1.0	1.0	1.0
Favicon	11055.0	0.628584	0.777777	-1.0	1.0	1.0	1.0	1.0
port	11055.0	0.728268	0.685324	-1.0	1.0	1.0	1.0	1.0
HTTPS_token	11055.0	0.675079	0.737779	-1.0	1.0	1.0	1.0	1.0
Request_URL	11055.0	0.186793	0.982444	-1.0	-1.0	1.0	1.0	1.0
URL_of_Anchor	11055.0	-0.076526	0.715138	-1.0	-1.0	0.0	0.0	1.0
Links_in_tags	11055.0	-0.118137	0.763973	-1.0	-1.0	0.0	0.0	1.0

SFH	11055.0	-0.595749	0.759143	-1.0	-1.0	-1.0	-1.0	1.0
Submitting_to_email	11055.0	0.635640	0.772021	-1.0	1.0	1.0	1.0	1.0
Abnormal_URL	11055.0	0.705292	0.708949	-1.0	1.0	1.0	1.0	1.0
Redirect	11055.0	0.115694	0.319872	0.0	0.0	0.0	0.0	1.0
on_mouseover	11055.0	0.762099	0.647490	-1.0	1.0	1.0	1.0	1.0
RightClick	11055.0	0.913885	0.405991	-1.0	1.0	1.0	1.0	1.0
popUpWidnow	11055.0	0.613388	0.789818	-1.0	1.0	1.0	1.0	1.0
Iframe	11055.0	0.816915	0.576784	-1.0	1.0	1.0	1.0	1.0
age_of_domain	11055.0	0.061239	0.998168	-1.0	-1.0	1.0	1.0	1.0
DNSRecord	11055.0	0.377114	0.926209	-1.0	-1.0	1.0	1.0	1.0
web_traffic	11055.0	0.287291	0.827733	-1.0	0.0	1.0	1.0	1.0
Page_Rank	11055.0	-0.483673	0.875289	-1.0	-1.0	-1.0	1.0	1.0
Google_Index	11055.0	0.721574	0.692369	-1.0	1.0	1.0	1.0	1.0
Links_pointing_to_page	11055.0	0.344007	0.569944	-1.0	0.0	0.0	1.0	1.0
Statistical_report	11055.0	0.719584	0.694437	-1.0	1.0	1.0	1.0	1.0
Result	11055.0	0.113885	0.993539	-1.0	-1.0	1.0	1.0	1.0

```
df['Result'].unique()

df['Result'].nunique()

for i in df.columns:
 print     (i     ,     ":",     df[i].unique(),"\n     len
:",df[i].nunique())
 print ("\n")
```

```
df['Result'].unique()
```

```
array([-1,  1], dtype=int64)
```

```
df['Result'].nunique()
```

```
2
```

```
for i in df.columns:
 print (i , ":", df[i].unique(),"\n len :",df[i].nunique())
 print ("\n")
```

```
index : [    1     2     3 ... 11053 11054 11055]
 len : 11055


having_IPhaving_IP_Address : [-1  1]
 len : 2


URLURL_Length : [ 1  0 -1]
 len : 3


Shortining_Service : [ 1 -1]
 len : 2
```


```
having_IPhaving_IP_Address : [-1  1]
 len : 2


URLURL_Length : [ 1  0 -1]
 len : 3


Shortining_Service : [ 1 -1]
 len : 2
```


```
having_At_Symbol : [ 1 -1]
 len : 2


double_slash_redirecting : [-1  1]
 len : 2


Prefix_Suffix : [-1  1]
 len : 2


having_Sub_Domain : [-1  0  1]
 len : 3
```

```
SSLfinal_State : [-1  1  0]
 len : 3

Domain_registeration_length : [-1  1]
 len : 2

Favicon : [ 1 -1]
 len : 2

port : [ 1 -1]
 len : 2

HTTPS_token : [-1  1]
 len : 2

Request_URL : [ 1 -1]
 len : 2

URL_of_Anchor : [-1  0  1]
 len : 3
Links_in_tags : [ 1 -1  0]
 len : 3

SFH : [-1  1  0]
 len : 3

Submitting_to_email : [-1  1]
 len : 2

Abnormal_URL : [-1  1]
 len : 2

Redirect : [0 1]
 len : 2

on_mouseover : [ 1 -1]
 len : 2

RightClick : [ 1 -1]
 len : 2
```

```
popUpWidnow : [ 1 -1]
 len : 2


Iframe : [ 1 -1]
 len : 2


age_of_domain : [-1  1]
 len : 2


DNSRecord : [-1  1]
 len : 2


web_traffic : [-1  0  1]
 len : 3


Page_Rank : [-1  1]
 len : 2


Google_Index : [ 1 -1]
 len : 2


Links_pointing_to_page : [ 1  0 -1]
 len : 3


Statistical_report : [-1  1]
 len : 2


Result : [-1  1]
 len : 2
```

3. Check if there is any null value in any features.

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

```
index                           0
having_IPhaving_IP_Address      0
URLURL_Length                   0
Shortining_Service              0
having_At_Symbol                0
double_slash_redirecting        0
Prefix_Suffix                   0
having_Sub_Domain               0
SSLfinal_State                  0
Domain_registeration_length     0
Favicon                         0
port                            0
HTTPS_token                     0
Request_URL                     0
URL_of_Anchor                   0
Links_in_tags                   0
SFH                             0
Submitting_to_email             0
Abnormal_URL                    0
Redirect                        0
on_mouseover                    0
RightClick                      0
popUpWidnow                     0
Iframe                          0
age_of_domain                   0
DNSRecord                       0
web_traffic                    0
Page_Rank                      0
Google_Index                   0
Links_pointing_to_page         0
Statistical_report             0
Result                         0
dtype: int64
```

Correlation of features and feature selection:

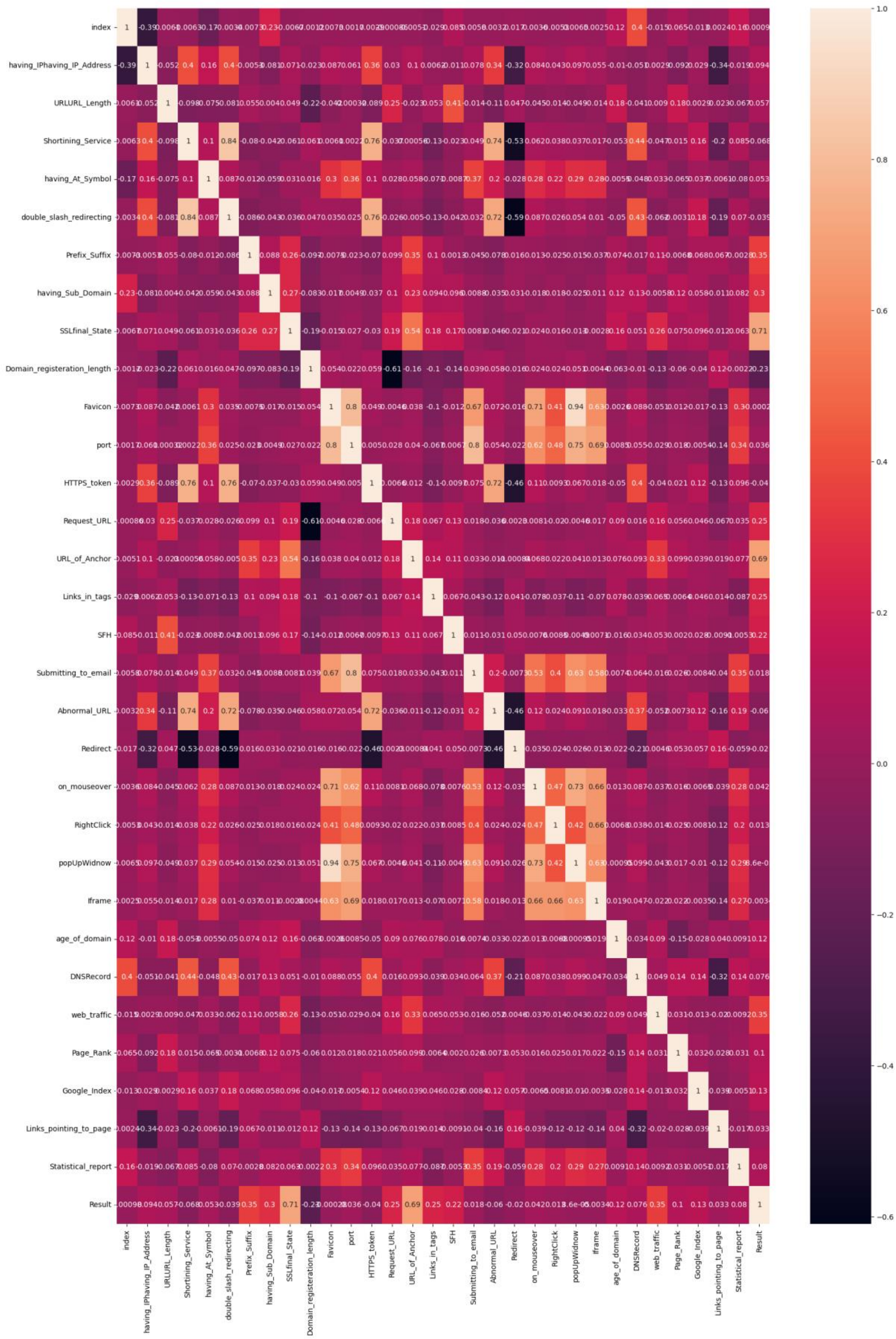
4. Next, we have to find if there are any correlated features present in the data. Remove the feature which might be correlated with some threshold.

```
df.corr()
```

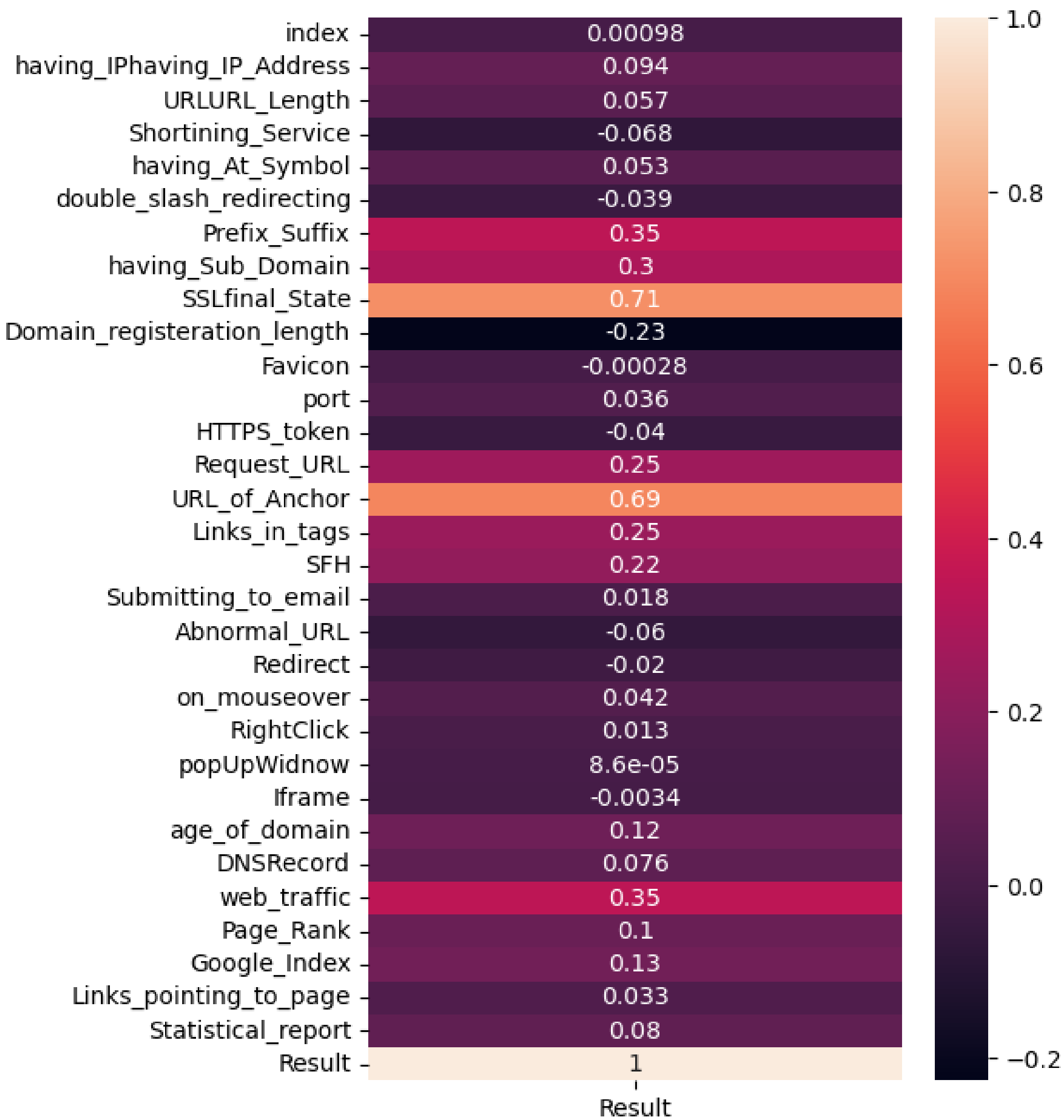
```
df.corr()
```

	index	having_IPhaving_IP_Address	URLURL_Length	Shortining_Service	having_At_Symbol	double_slash_redirecting	Prefix_Su
index	1.000000	-0.388317	0.006105	-0.006281	-0.169478	-0.003363	-0.007
having_IPhaving_IP_Address	-0.388317	1.000000	-0.052411	0.403461	0.158699	0.397389	-0.005
URLURL_Length	0.006105	-0.052411	1.000000	-0.097881	-0.075108	-0.081247	0.055
Shortining_Service	-0.006281	0.403461	-0.097881	1.000000	0.104447	0.842796	-0.080
having_At_Symbol	-0.169478	0.158699	-0.075108	0.104447	1.000000	0.086960	-0.011
double_slash_redirecting	-0.003363	0.397389	-0.081247	0.842796	0.086960	1.000000	-0.085
Prefix_Suffix	-0.007340	-0.005257	0.055247	-0.080471	-0.011726	-0.085590	1.000
having_Sub_Domain	0.234091	-0.080745	0.003997	-0.041916	-0.058976	-0.043079	0.087
SSLfinal_State	-0.006682	0.071414	0.048754	-0.061426	0.031220	-0.036200	0.261
Domain_registeration_length	-0.001180	-0.022739	-0.221892	0.060923	0.015522	0.047464	-0.096
Favicon	0.007293	0.087025	-0.042497	0.006101	0.304899	0.035100	-0.007
port	0.001656	0.060979	0.000323	0.002201	0.364891	0.025060	-0.022
HTTPS_token	0.002916	0.363534	-0.089383	0.757838	0.104561	0.760799	-0.070
Request_URL	-0.000862	0.029773	0.246348	-0.037235	0.027909	-0.026368	0.098

```
plt.figure(figsize = (20,30))
sns.heatmap(data = df.corr(), annot = True)
plt.show()
```


index
having_IPhaving_IP_Address
URLURL_Length
Shortining_Service
having_At_Symbol
double_slash_redirecting
Prefix_Suffix
having_Sub_Domain
SSLfinal_State
Domain_registeration_length
Favicon
port
HTTPS_token
Request_URL
URL_of_Anchor
Links_in_tags
SFH
Submitting_to_email
Abnormal_URL
Redirect
on_mouseover
RightClick
popUpWidnow
Iframe
age_of_domain
DNSRecord
web_traffic
Page_Rank
Google_Index
Links_pointing_to_page
Statistical_report
Result
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6

```
plt.figure(figsize = (5,8))
sns.heatmap(df.corr()[['Result']], annot = True)
```



With 0.84 'Shortining_Service' and 'double_slash_redirecting' are correlated with each other .

With 0.8 'Favicon', 'port' are correlated with each other .

Not going to drop any columns has no high correlation.

Building Classification Model

1. Finally, build a robust classification system that classifies whether the URL sample is a phishing site or not.

- Build classification models using a binary classifier to detect malicious or phishing URLs.

```
#identify x and y

features = ['index', 'having_IPhaving_IP_Address',
'URLURL_Length',
       'Shortining_Service', 'having_At_Symbol',
'double_slash_redirecting',
       'Prefix_Suffix', 'having_Sub_Domain',
'SSLfinal_State',
       'Domain_registeration_length', 'Favicon', 'port',
'HTTPS_token',
       'Request_URL', 'URL_of_Anchor', 'Links_in_tags',
'SFH',
       'Submitting_to_email', 'Abnormal_URL',
'Redirect', 'on_mouseover',
       'RightClick', 'popUpWidnow', 'Iframe',
'age_of_domain', 'DNSRecord',
       'web_traffic', 'Page_Rank', 'Google_Index',
'Links_pointing_to_page',
       'Statistical_report']
target = ['Result']



x = df[features]
y = df[target]

#splitting
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x,
y, test_size=0.33, random_state=42)

from sklearn.linear_model import LogisticRegression
```

```
logreg=LogisticRegression()

logreg.fit(X_train,y_train)
```

```
logreg.fit(X_train,y_train)
```

```
▾ LogisticRegression
LogisticRegression()
```

```
pred=logreg.predict(X_test)

logreg.score(X_train,y_train)

logreg.score(X_test,y_test)
```

```
logreg.score(X_train,y_train)
```

```
0.9261409667836888
```

```
logreg.score(X_test,y_test)
```

```
0.9199780761852563
```

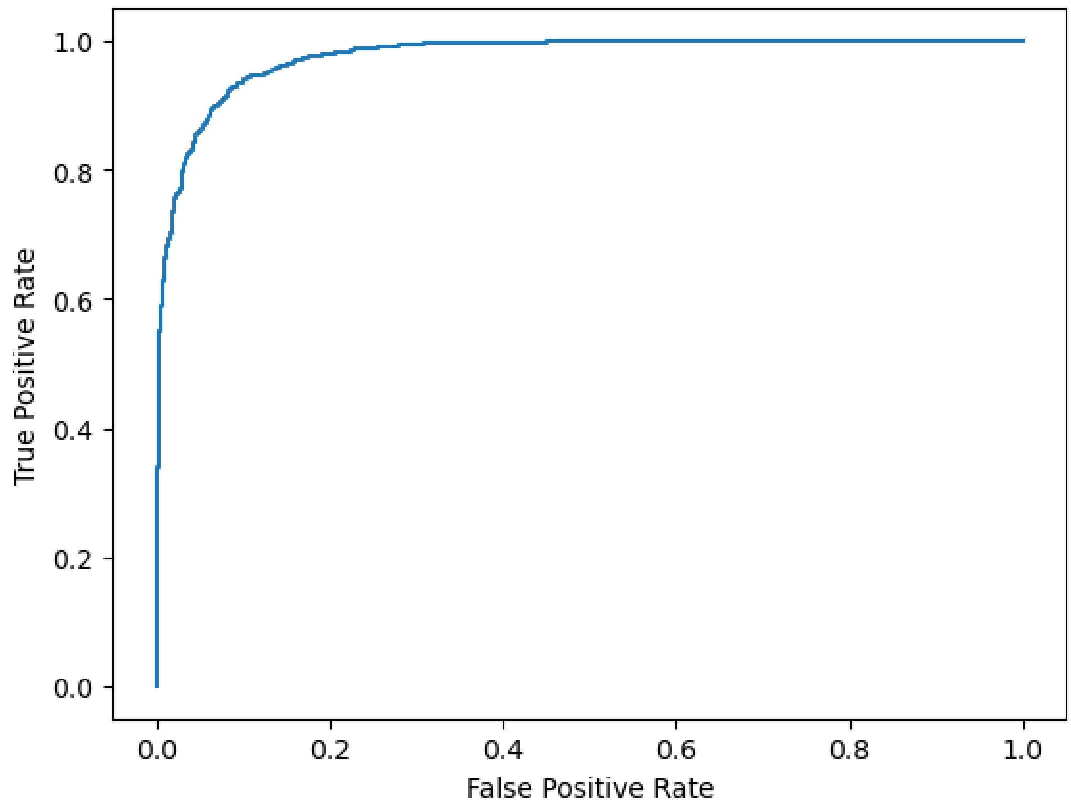
```
#testing
from sklearn.metrics import confusion_matrix, classification_report
confusion_matrix(y_test, pred)

print(classification_report(y_test, pred))
```

```
[63]: #testing
      from sklearn.metrics import confusion_matrix, classification_report
      confusion_matrix(y_test, pred)

[63]: array([[1421,  144],
             [ 148, 1936]], dtype=int64)
```

```
[64]: print(classification_report(y_test, pred))

              precision    recall  f1-score   support

          -1       0.91      0.91      0.91      1565
           1       0.93      0.93      0.93      2084

    accuracy                           0.92      3649
   macro avg       0.92      0.92      0.92      3649
weighted avg       0.92      0.92      0.92      3649
```

- Illustrate the diagnostic ability of this binary classifier by plotting the ROC curve.

```
from sklearn import metrics

y_pred_proba = logreg.predict_proba(X_test)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred_proba)

#create ROC curve
plt.plot(fpr,tpr)
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



- Validate the accuracy of data by the K-Fold cross-validation technique.

```
#define cross-validation method to use
cv = KFold(n_splits=10, random_state=1, shuffle=True)

#use k-fold CV to evaluate model
scores      =      cross_val_score(logreg,     x,     y,
scoring='neg_mean_absolute_error',
                         cv=cv, n_jobs=-1)

#view mean absolute error
mean(absolute(scores))

#view RMSE
sqrt(mean(absolute(scores)))
```

```
#view mean absolute error
mean(absolute(scores))
```

```
0.15413630301195455
```

```
# average absolute error between the model prediction and the actual observed data is 0.154
```

```
#view RMSE
sqrt(mean(absolute(scores)))
```

```
0.39260196511473877
```

```
# root mean squared error (RMSE) was 0.392
```

- The final output consists of the model, which will give maximum accuracy on the validation dataset with selected attributes.

```
#testing
from sklearn.metrics import confusion_matrix, classification_report
confusion_matrix(y_test, pred)

print(classification_report(y_test, pred))
```

```
[63]: #testing
      from sklearn.metrics import confusion_matrix, classification_report
      confusion_matrix(y_test, pred)
```

```
[63]: array([[1421,  144],
             [ 148, 1936]], dtype=int64)
```

```
[64]: print(classification_report(y_test, pred))
```

```
              precision    recall  f1-score   support

          -1       0.91      0.91      0.91      1565
           1       0.93      0.93      0.93      2084

    accuracy                           0.92      3649
   macro avg       0.92      0.92      0.92      3649
weighted avg       0.92      0.92      0.92      3649
```