Pre Processing

```
from google.colab import drive
drive.mount('/content/drive')
```

```
import pandas as pd

# df=pd.read_csv("../data/initial_urls.csv")
# df2=pd.read_csv("../data/additional_urls.csv") #reading datas
df = pd.read_csv("/content/drive/MyDrive/ML-Phishing Detection
Project/old/data/initial_urls.csv")
df2 = pd.read_csv("/content/drive/MyDrive/ML-Phishing Detection
Project/old/data/additional_urls.csv")

print(df.head())   #display first 5 rows...
print(df2.head())   #display first 5 rows...

def initial_read(df):
    print('SHAPE')
    print(df.shape) #display no of rows and columns
    print('-------')
    print('DTYPES')
    print(df.dtypes) #data types of different columns
    print('-------')
    print('NULL VALUES')
    print(df.isnull().sum()) #display sum of total null valued rows of
each columns
    print('-------')
#initial_read(df)
#initial_read(df2)
df.dropna(inplace=True) #modifies the data by removing the rows having
null value. i.e, rows without having a label of df
#print(df.isnull().sum())
df2.drop(columns = ['Unnamed: 0', 'label'], inplace=True) #dropping 2
unwanted columns of df2...unnamed and label.
#print(df2.head())
df['label']=df['label'].astype(int) #changing float dtype of label to
int
#print(df.dtypes)
df.rename(columns={"domain": "url", "label": "phishing"},inplace=True)
#changing column names to url and phishing of df
#print(df.columns)
df2.rename(columns={"result": "phishing"},inplace=True) #changing
column name result to phishing of df2
#print(df2.head())
```

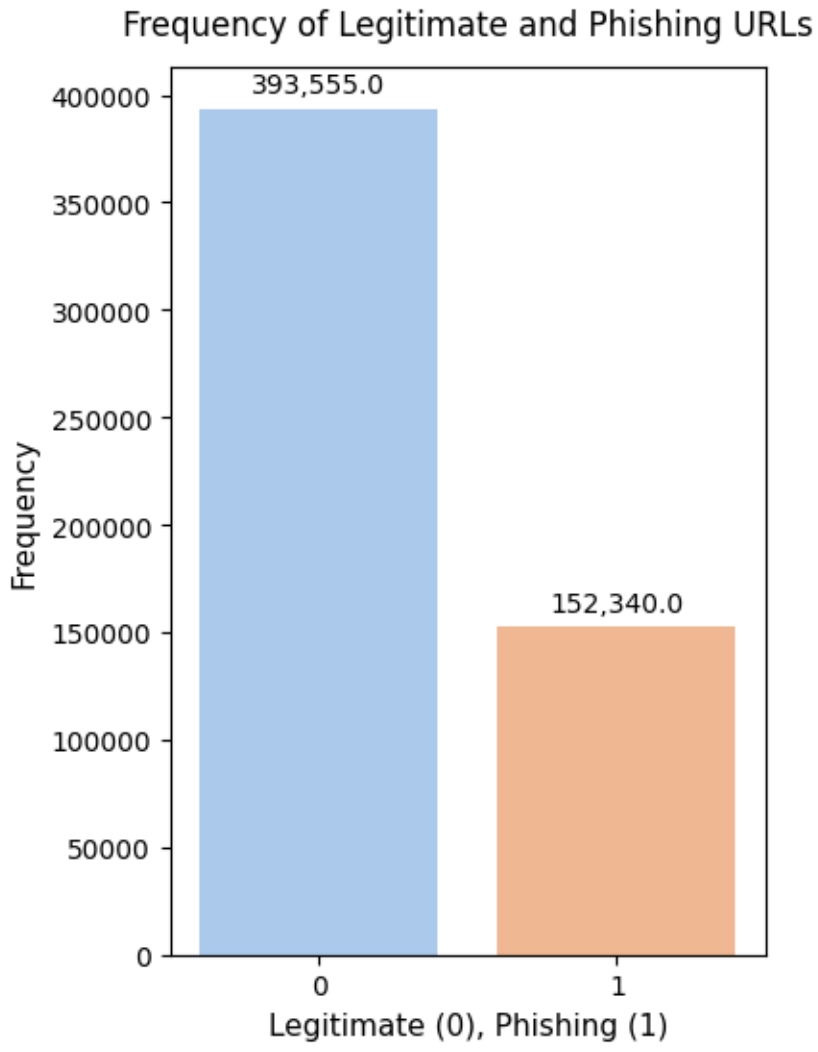
```
df['url']= 'https://' + df['url'].astype(str) #adding protocols to urls
#print(df.head())

df_final = pd.concat([df,df2]) #concatenate df and df2
#initial_read(df_final)
#print(df_final[df_final.duplicated()])
df_final .drop_duplicates(inplace=True) #drop duplicate rows..
#print(df_final[df_final.duplicated()])

# df_final.to_csv('../data/final_urls.csv', index=False) #save
final/cleaned dataset to final_urls.csv
path = '/content/drive/My Drive/ML-Phishing Detection
Project/data/final_urls.csv'
with open(path, 'w', encoding = 'utf-8-sig') as f:
  df_final.to_csv(f,index=False)

print(df.columns)
print(df2.columns)
print(f" \nnew dataset Length:     {len(df_final)}")
```

```
                                                domain  label
0  nobell.it/70ffb52d079109dca5664cce6f317373782/...     1.0
1  www.dghjdgf.com/paypal.co.uk/cycgi-bin/webscrc...     1.0
2  serviciosbys.com/paypal.cgi.bin.get-into.herf....     1.0
3  mail.printakid.com/www.online.americanexpress....     1.0
4  thewhiskeydregs.com/wp-content/themes/widescre...     1.0
   Unnamed: 0                       url   label  result
0           0     https://www.google.com  benign       0
1           1    https://www.youtube.com  benign       0
2           2   https://www.facebook.com  benign       0
3           3      https://www.baidu.com  benign       0
4           4  https://www.wikipedia.org  benign       0
Index(['url', 'phishing'], dtype='object')
Index(['url', 'phishing'], dtype='object')
```

```
new dataset Length:     545895
```

```
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings("ignore")

# df = pd.read_csv("../data/final_urls.csv")
df = pd.read_csv("/content/drive/MyDrive/ML-Phishing Detection
Project/data/final_urls.csv")


plt.figure(figsize = (4, 6))
```

```
bar = sns.countplot(x=df["phishing"],
              data=df,
              palette="pastel")
for p in bar.patches:
    bar.annotate(format(p.get_height(), ','),
                  (p.get_x() + p.get_width() / 2., p.get_height()),
                  ha = 'center', va = 'center',
                  xytext = (0, 9),
                  textcoords = 'offset points')
plt.title('Frequency of Legitimate and Phishing URLs', fontsize = 12,
pad = 12)
plt.xlabel('Legitimate (0), Phishing (1)', fontsize = 11)
plt.ylabel("Frequency",fontsize = 11)
plt.show()
```



Feature Extraction

```
import pandas as pd
import urllib
from urllib.parse import urlparse
import re
from math import log
import time
start_time = time.time()

df = pd.read_csv('/content/drive/MyDrive/ML-Phishing Detection
Project/data/final_urls.csv')
urls = [url for url in df['url']]
print( "[%s datas]" %(len(urls)))

df['protocol'],df['domain'],df['path'],df['query'],df['fragment'] =
zip(*[urllib.parse.urlsplit(x) for x in urls])

shortening_services =
r"bit\.ly|goo\.gl|shorte\.st|go2l\.ink|x\.co|ow\.ly|t\.co|tinyurl|tr\.i
m|is\.gd|cli\.gs|" \
                      r"yfrog\.com|migre\.me|ff\.im|tiny\.cc|url4\.eu|t
wit\.ac|su\.pr|twurl\.nl|snipurl\.com|" \
                      r"short\.to|BudURL\.com|ping\.fm|post\.ly|Just\.a
s|bkite\.com|snipr\.com|fic\.kr|loopt\.us|" \
                      r"doiop\.com|short\.ie|kl\.am|wp\.me|rubyurl\.com
|om\.ly|to\.ly|bit\.do|t\.co|lnkd\.in|db\.tt|" \
                      r"qr\.ae|adf\.ly|goo\.gl|bitly\.com|cur\.lv|tinyu
rl\.com|ow\.ly|bit\.ly|ity\.im|q\.gs|is\.gd|" \
                      r"po\.st|bc\.vc|twitthis\.com|u\.to|j\.mp|buzurl\
.com|cutt\.us|u\.bb|yourls\.org|x\.co|" \
                      r"prettylinkpro\.com|scrnch\.me|filoops\.info|vzt
url\.com|qr\.net|1url\.com|tweez\.me|v\.gd|" \
                      r"tr\.im|link\.zip\.net"

def getEntropy(url):
    url = url.lower()
    probs = [url.count(c) / len(url) for c in set(url)]
    entropy = -sum([p * log(p) / log(2.0) for p in probs])
    return entropy

def hasLogin(url):
    return int('login' in url.lower())

def redirection(url):
    pos = url.rfind('//')
```

```
    if pos > 6:
      if pos > 7:
        return 1
      else:
        return 0
    else:
      return 0

def lenClassify(url):
    if len(url) < 54:
      length = 0
    else:
      length = 1
    return length

def haveAtSign(url):
    if "@" in url:
      at = 1
    else:
      at = 0
    return at

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

def tinyURL(url):
    match=re.search(shortening_services,url)
    if match:
        return 1
    else:
        return 0

def isDomainIp(domain):
    domain = domain.split(':')
    pattern = r'^(?:[0-9]{1,3}\.){3}[0-9]{1,3}$|^(?:[a-f0-
9]{1,4}:){7}[a-f0-9]{1,4}$'
    match = re.match(pattern, domain[0])
    if match is not None:
      return 1
    else:
      return 0

def prefixSuffix(domain):
```

```
    if '-' in domain:
        return 1
    else:
        return 0

def get_features(df):
    df['get_Entropy'] = df['url'].map(lambda x: getEntropy(x))
    df['has_Login'] = df['url'].map(lambda x: hasLogin(x))
    df['redirection'] = df['url'].map(lambda x: redirection(x))
    df['len_Classify'] = df['url'].map(lambda x: lenClassify(x))
    df['have_At_Sign'] = df['url'].map(lambda x: haveAtSign(x))
    df['get_Depth'] = df['url'].map(lambda x: getDepth(x))
    df['tiny_URL'] = df['url'].map(lambda x: tinyURL(x))
    df['is_Domain_Ip'] = df['domain'].map(lambda x: isDomainIp(x))
    df['prefix_Suffix'] = df['domain'].map(lambda x: prefixSuffix(x))

    needed_cols = ['url', 'domain', 'path', 'query', 'fragment']
    for col in needed_cols:
        df[f'{col}_length']=df[col].str.len()
        df[f'qty_dot_{col}'] = df[[col]].applymap(lambda x:
str.count(x, '.'))
        df[f'qty_hyphen_{col}'] = df[[col]].applymap(lambda x:
str.count(x, '-'))
        df[f'qty_slash_{col}'] = df[[col]].applymap(lambda x:
str.count(x, '/'))
        df[f'qty_questionmark_{col}'] = df[[col]].applymap(lambda x:
str.count(x, '?'))
        df[f'qty_equal_{col}'] = df[[col]].applymap(lambda x:
str.count(x, '='))
        df[f'qty_at_{col}'] = df[[col]].applymap(lambda x: str.count(x,
'@'))
        df[f'qty_and_{col}'] = df[[col]].applymap(lambda x:
str.count(x, '&'))
        df[f'qty_exclamation_{col}'] = df[[col]].applymap(lambda x:
str.count(x, '!'))
        df[f'qty_space_{col}'] = df[[col]].applymap(lambda x:
str.count(x, ' '))
        df[f'qty_tilde_{col}'] = df[[col]].applymap(lambda x:
str.count(x, '~'))
        df[f'qty_comma_{col}'] = df[[col]].applymap(lambda x:
str.count(x, ','))
        df[f'qty_plus_{col}'] = df[[col]].applymap(lambda x:
str.count(x, '+'))
        df[f'qty_asterisk_{col}'] = df[[col]].applymap(lambda x:
str.count(x, '*'))
        df[f'qty_hashtag_{col}'] = df[[col]].applymap(lambda x:
str.count(x, '#'))
```

```
        df[f'qty_dollar_{col}'] = df[[col]].applymap(lambda x:
str.count(x, '$'))
        df[f'qty_percent_{col}'] = df[[col]].applymap(lambda x:
str.count(x, '%'))

get_features(df)

col_in_question = ['qty_slash_domain',
'qty_questionmark_domain','qty_equal_domain', 'qty_at_domain',
'qty_and_domain',
 'qty_exclamation_domain', 'qty_space_domain',
'qty_tilde_domain','qty_comma_domain', 'qty_plus_domain',
 'qty_asterisk_domain','qty_hashtag_domain', 'qty_dollar_domain',
'qty_percent_domain', 'qty_questionmark_path',
 'qty_hashtag_path', 'qty_hashtag_query',
'qty_at_fragment','qty_tilde_fragment', 'qty_plus_fragment']

df.drop(columns = col_in_question, inplace=True)

df.to_csv("/content/drive/MyDrive/ML-Phishing Detection
Project/data/url_features.csv",index=False)

noOfFeatures=len(df.select_dtypes(include='int').columns)

print("\n--- %s features extracted in %s seconds ---" %
(noOfFeatures,time.time() - start_time))
```

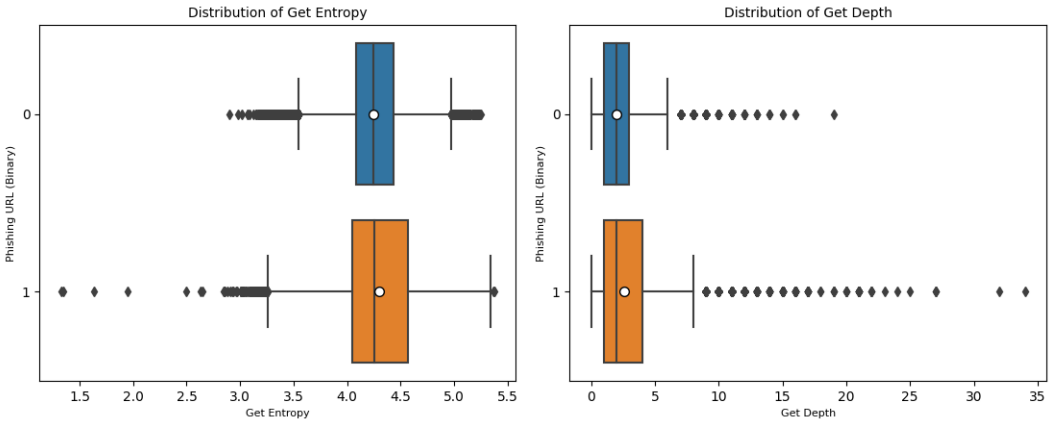
```
[545895 datas]
```

```
--- 74 features extracted in 52.08950185775757 seconds ---
```

BOXPLOT GRAPH

```
import matplotlib.pyplot as plt
import seaborn as sns
num_cols = df.select_dtypes(exclude='object')

for col in num_cols:
    column_name = col.title().replace('_', ' ')
    title = 'Distribution of ' + column_name
    sns.boxplot(x=df[col],
                y=df['phishing'],
                data=df,
                orient='h',
```

```
                showmeans=True,
                meanprops={"marker":"o",
                           "markerfacecolor":"white",
                           "markeredgecolor":"black",
                           "markersize":"7"})
    plt.xlabel(column_name, fontsize = 8)
    plt.ylabel('Phishing URL (Binary)', fontsize = 8)
    plt.title(title, fontsize = 10, pad = 6)
    plt.show();
```



Pycaret Classification

```
pip install pycaret
```

```
import pandas as pd
from pycaret.classification import *
import time
start_time = time.time()

df = pd.read_csv('/content/drive/MyDrive/ML-Phishing Detection
Project/data/url_features.csv')
data =
df.drop(columns=['url','protocol','domain','path','query','fragment'])

s = setup(data, target = 'phishing', session_id = 123)
best = compare_models()
```

```
print(f"\n--- pycaret check completed in {(time.time() -
start_time)/60} minutes ---")
```

	Description	Value
0	Session id	123
1	Target	phishing
2	Target type	Binary
3	Original data shape	(545895, 75)
4	Transformed data shape	(545895, 75)
5	Transformed train set shape	(382126, 75)
6	Transformed test set shape	(163769, 75)
7	Numeric features	74
8	Preprocess	True
9	Imputation type	simple
10	Numeric imputation	mean
11	Categorical imputation	mode
12	Fold Generator	StratifiedKFold
13	Fold Number	10
14	CPU Jobs	-1
15	Use GPU	False
16	Log Experiment	False
17	Experiment Name	clf-default-name
18	USI	3fb4

Initiated	. . . . .	12:01:16
Status	. . . . .	Fitting 10 Folds

	Description	Value
Estimator	. . . . .	Random Forest Classifier

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
lr	Logistic Regression	0.9445	0.9619	0.8166	0.9816	0.8915	0.8547	0.8612	75.2590
ridge	Ridge Classifier	0.9279	0.0000	0.7425	0.9988	0.8518	0.8055	0.8210	3.4460
svm	SVM - Linear Kernel	0.9271	0.0000	0.8320	0.9215	0.8679	0.8182	0.8261	18.7650
dt	Decision Tree Classifier	0.9204	0.8977	0.8463	0.8655	0.8558	0.8009	0.8010	7.9880
knn	K Neighbors Classifier	0.9187	0.9402	0.7856	0.9106	0.8435	0.7890	0.7929	266.3830
nb	Naive Bayes	0.7918	0.8363	0.2822	0.9089	0.4306	0.3439	0.4324	3.1910

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
lr	Logistic Regression	0.9445	0.9619	0.8166	0.9816	0.8915	0.8547	0.8612	75.2590
et	Extra Trees Classifier	0.9402	0.9677	0.8584	0.9221	0.8891	0.8483	0.8493	135.2880
rf	Random Forest Classifier	0.9400	0.9716	0.8526	0.9266	0.8880	0.8472	0.8485	115.4030
xgboost	Extreme Gradient Boosting	0.9318	0.9677	0.8028	0.9446	0.8679	0.8223	0.8273	159.0900

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
qda	Quadratic Discriminant Analysis	0.9313	0.9444	0.7752	0.9738	0.8630	0.8179	0.8276	6.1170
lda	Linear Discriminant Analysis	0.9284	0.9571	0.7442	0.9987	0.8529	0.8069	0.8221	9.3110
ridge	Ridge Classifier	0.9279	0.0000	0.7425	0.9988	0.8518	0.8055	0.8210	3.4460
svm	SVM - Linear Kernel	0.9271	0.0000	0.8320	0.9215	0.8679	0.8182	0.8261	18.7650
lightgbm	Light Gradient Boosting Machine	0.9226	0.9608	0.7715	0.9403	0.8475	0.7963	0.8033	9.2770
dt	Decision Tree Classifier	0.9204	0.8977	0.8463	0.8655	0.8558	0.8009	0.8010	7.9880
knn	K Neighbors Classifier	0.9187	0.9402	0.7856	0.9106	0.8435	0.7890	0.7929	266.3830
gbc	Gradient Boosting Classifier	0.9092	0.9379	0.7161	0.9454	0.8149	0.7563	0.7691	117.2510
ada	Ada Boost Classifier	0.9076	0.9321	0.7240	0.9294	0.8139	0.7537	0.7641	30.6170
nb	Naive Bayes	0.7918	0.8363	0.2822	0.9089	0.4306	0.3439	0.4324	3.1910
dummy	Dummy Classifier	0.7209	0.5000	0.0000	0.0000	0.0000	0.0000	0.0000	1.9710

```
--- pycaret check completed in 9747.736599206924 seconds ---
```

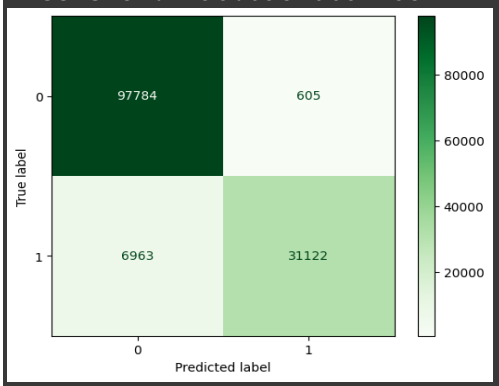
Model Implementation

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
import time
start_time = time.time()
import warnings
warnings.filterwarnings("ignore")

df = pd.read_csv("/content/drive/MyDrive/ML-Phishing Detection
Project/data/url_features.csv")

x =
df.drop(columns=['url','protocol','domain','path','query','fragment','p
hishing'])
y = df['phishing']
X_train, X_test, y_train, y_test = train_test_split(x,y,random_state =
42, stratify = y )

clf = LogisticRegression(penalty="l2",C=10,max_iter=1000,
random_state=42)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)

acc = accuracy_score(y_test, y_pred)*100
train_accuracy = clf.score(X_train, y_train)*100
test_accuracy = clf.score(X_test, y_test)*100

print(f"LogisticRegression Accuracy: {acc}")
print("Training accuracy:", train_accuracy)
print("Test accuracy:", test_accuracy)

print(f"\n--- Model Evaluation ended in {(time.time() - start_time)/60}
minutes ---")
```

```
LogisticRegression Accuracy: 94.48246552456878
Training accuracy: 94.53105727356437
Test accuracy: 94.48246552456878

--- Model Evaluation ended in 1.2195839087168376 minutes ---
```

```
from sklearn.metrics import  confusion_matrix, classification_report,
ConfusionMatrixDisplay

print('\033[1m' + '\t\t-----CLASSIFICATION REPORT-----' + '\033[0m')
print(classification_report(y_test,y_pred))
print('\033[1m' + '\t\t-----CONFUSION MATRIX-----' + '\033[0m')
print(confusion_matrix(y_test, y_pred))
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix = cm, display_labels =
clf.classes_)
disp.plot(cmap = 'Greens', values_format='')

print('\033[1m' + '\t\t-----MATRIX CALC-----' + '\033[0m')
tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
print("Accuracy: ", ((tn + tp) / (tn + fp + fn + tp))*100)
print('Misclassification Rate: ', ((fp+fn)/(tp+fp+tn+fn))*100)
print("Recall: ", (tp / (tp + fn))*100)
print("Specificity: ", (tn / (tn + fp))*100)
print("Precision: ", (tp / (tp + fp))*100)
```

```
               -----CLASSIFICATION REPORT-----
              precision    recall  f1-score   support

           0       0.93      0.99      0.96     98389
           1       0.98      0.82      0.89     38085

    accuracy                           0.94    136474
   macro avg       0.96      0.91      0.93    136474
weighted avg       0.95      0.94      0.94    136474

               -----CONFUSION MATRIX-----
[[97784   605]
 [ 6963 31122]]
               -----MATRIX CALC-----
Accuracy:  94.45462139308586
Misclassification Rate:  5.545378606914138
Recall:  81.71721150059078
Specificity:  99.38509386211874
Precision:  98.09310681753712
```



```
import numpy as np

coefficients = clf.coef_[0]

feature_importance = pd.DataFrame({'Feature': x.columns, 'Importance':
np.abs(coefficients)})
feature_importance = feature_importance.sort_values('Importance',
ascending=False)

col_widths = [27, 10]
print('{:<{}} {:>{}}\n'.format('\033[1m    Features\033[0m',
col_widths[0], '\033[1m           Importance\033[0m', col_widths[1]))

for index, row in feature_importance.iterrows():
  print('{:<{}} {:>{}}'.format(row["Feature"], col_widths[0],
row["Importance"], col_widths[1]))
```

Features	Importance
has_Login	17.50104731596883
qty_slash_url	17.081183381852934
qty_slash_query	17.07447575442481
qty_slash_path	15.593042042519762
is_Domain_Ip	13.390253867250552
fragment_length	8.262890358688013
domain_length	8.190691646606577
query_length	8.152779676465919
path_length	8.102829702824081
url_length	8.08924680042928
qty_questionmark_url	7.782155913174315
redirection	4.103847946602051
qty_questionmark_query	2.6621056912247427
qty_hashtag_url	1.7503123607980398
qty_dot_domain	1.5527849024752542
qty_space_url	1.3967219137540452
qty_at_query	1.2732887325157511
qty_equal_query	1.2387965290096226
get_Depth	1.1292673252545378
qty_space_path	1.0647616851107968
qty_equal_path	1.0638010784853535
qty_plus_url	0.9771358588903926
qty_exclamation_url	0.9299441385669271
qty_equal_fragment	0.8144508208267502
prefix_Suffix	0.7672738941547081
qty_exclamation_path	0.7656261964945975
qty_at_path	0.760984663437558
qty_equal_url	0.7548966518428326
qty_space_fragment	0.7387934507115947
qty_dot_path	0.6893527568072485
len_Classify	0.6189908642550525
get_Entropy	0.586740456967459
qty_at_url	0.5710510032327958
qty_dot_query	0.5508604180692102
have_At_Sign	0.5241119783831402

```
qty_plus_path                0.5155919642094619
qty_dot_fragment             0.4975024682541953
qty_hashtag_fragment         0.4884676449055037
qty_percent_fragment         0.4721693681148116
qty_plus_query               0.4615438946798377
qty_hyphen_query             0.40983814725188505
qty_percent_query            0.3979593911476135
qty_comma_query              0.3802826938822419
qty_and_query                0.3322483356612761
qty_hyphen_domain            0.2941092940114936
qty_tilde_query              0.29176359438556876
qty_hyphen_path              0.28695563631840615
qty_dollar_url               0.26438820981016137
qty_and_fragment             0.25891134594098597
qty_tilde_path               0.24272259917237
qty_and_path                 0.234560047166241
qty_asterisk_query           0.21841953339249173
qty_space_query              0.20668753332159234
qty_hyphen_url               0.20604667522719167
qty_hyphen_fragment          0.19663781433112265
qty_comma_path               0.19436416188649505
qty_comma_url                0.1941313945068137
qty_slash_fragment           0.19151390408523206
qty_dot_url                  0.18493074066094836
qty_asterisk_path            0.17178166383517102
qty_and_url                  0.16603644856400335
qty_dollar_fragment          0.16440943018532014
qty_exclamation_query        0.12563001068773397
qty_questionmark_fragment    0.12140466028934585
qty_dollar_path              0.1167456675467784
tiny_URL                     0.11124906947800393
qty_percent_path             0.10913155953709368
qty_asterisk_fragment        0.08942613083734235
qty_tilde_url                0.04904099521305312
qty_asterisk_url             0.04278826128011399
qty_exclamation_fragment     0.038687931384083 96
qty_percent_url              0.030244689775533563
qty_dollar_query             0.016766887922030665
qty_comma_fragment           0.00821286251044417
```

```
import pickle

with open('/content/drive/MyDrive/ML-Phishing Detection
Project/model/model.pkl', 'wb') as f:
    pickle.dump(clf, f)
```