```
# import necessary libraries
import pandas as pd
```

Data Preparation

```
# load dataset
from google.colab import files
files=files.upload()
```

Choose Files urldata.csv

- **urldata.csv**(text/csv) - 35424746 bytes, last modified: 10/24/2019 - 100% done

```
Saving urldata.csv to urldata.csv
```

```
df=pd.read_csv('urldata.csv')
```

```
df.head()
```

	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

```
#remove unwanted column
df=df.drop('Unnamed: 0',axis=1)
```

```
df.head()
```

	url	label	result
0	https://www.google.com	benign	0
1	https://www.youtube.com	benign	0
2	https://www.facebook.com	benign	0
3	https://www.baidu.com	benign	0
4	https://www.wikipedia.org	benign	0

```
# check shape of dataset
df.shape
```

```
(450176, 3)
```

```
#check for Null values
```

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

```
url       0
label     0
result    0
dtype: int64
```

```
#check datatype
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 450176 entries, 0 to 450175
Data columns (total 3 columns):
 #   Column  Non-Null Count   Dtype 
---  ------  --------------   ----- 
 0   url     450176 non-null  object
 1   label   450176 non-null  object
 2   result  450176 non-null  int64 
dtypes: int64(1), object(2)
memory usage: 10.3+ MB
```

```
# check unique values of each column
print(df['label'].unique())
print(df['result'].unique())
```

```
['benign' 'malicious']
[0 1]
```

```
print(df['label'].value_counts())
print(df['result'].value_counts())
```

```
benign       345738
malicious    104438
Name: label, dtype: int64
0    345738
1    104438
Name: result, dtype: int64
```

```
# drop column label, as it's need is satisfied by result column i.e benign=0 and malicious=1
#df=df.drop('label', axis=1)
```

```
df.head()
```

	url	label	result
0	https://www.google.com	benign	0
1	https://www.youtube.com	benign	0
2	https://www.facebook.com	benign	0
3	https://www.baidu.com	benign	0
4	https://www.wikipedia.org	benign	0

Feature Creation based on some characteristics

Length of XYZ

```
# import libraries
from urllib.parse import urlparse
```

```
df_link = df
```

```
#1. get length of URL
df_link['url_length'] = df_link['url'].apply(lambda x: len(x))
```

```
#2. get length of Hostname
df_link['hostname_length'] = df_link['url'].apply(lambda x: len(str(urlparse(x).hostname)))
```

```
#3. get the length of the path from URL
df_link['path_length'] = df_link['url'].apply(lambda x: len(urlparse(x).path))
```

```
#4. get the length of first directory
def fd(URL):
    a=urlparse(URL).path
    if str(a[0:2]) == '//':
        return len(a.split('/')[2])
    else:
        try:
            return len(a.split('/')[1])
        except:
            return 0

df_link['FirstDir_length'] = df_link['url'].apply(lambda v: fd(v))
```

```
df_link.head()
```

	url	label	result	url_length	hostname_length	path_length	FirstDir_le
0	https://www.google.com	benign	0	22	14	0	
1	https://www.youtube.com	benign	0	23	15	0	
2	https://www.facebook.com	benign	0	24	16	0	
3	https://www.baidu.com	benign	0	21	13	0	
4	https://www.wikipedia.org	benign	0	25	17	0	

Counts of XYZ

```
#1. total -
df_link['total-'] = df_link['url'].apply(lambda i: i.count('-'))
```

```
#2. total @
df_link['total@'] = df_link['url'].apply(lambda i: i.count('@'))
```

```
#3. total ?
df_link['total?'] = df_link['url'].apply(lambda i: i.count('?'))
```

```
#4. total %
df_link['total%'] = df_link['url'].apply(lambda i: i.count('%'))
```

```
#5. total .
df_link['total.'] = df_link['url'].apply(lambda i: i.count('.'))
```

```
#6. total =
df_link['total='] = df_link['url'].apply(lambda i: i.count('='))
```

```
#7. total http
df_link['totalhttp'] = df_link['url'].apply(lambda i: i.count('http'))
```

```
#8. total https
df_link['totalhttps'] = df_link['url'].apply(lambda i: i.count('https'))
```

```
#9. total www
df_link['totalwww'] = df_link['url'].apply(lambda i: i.count('www'))
```

```
#10 count total digits in the URL
def totaldigits(url):
    digits = 0
    for i in url:
        if i.isnumeric():
            digits = digits + 1
    return digits
df_link['totaldigits']= df_link['url'].apply(lambda i: totaldigits(i))
```

```
#11 count total letters in the URL
def totalletters(url):
    letters = 0
    for i in url:
        if i.isalpha():
            letters = letters + 1
    return letters
df_link['totalletters']= df_link['url'].apply(lambda i: totalletters(i))
```

```
#12 count total directories in the URL
def totaldirs(url):
    urldir = urlparse(url).path
    return urldir.count('/')
df_link['totaldirs'] = df_link['url'].apply(lambda i: totaldirs(i))
```

```
#13 count of dots in netloc
df_link['totalnetlocdots'] = df_link['url'].apply(lambda i: urlparse(i).netloc.count('.'))
```

```
from google.colab import files
files = files.upload()
```

Choose Files tld.txt

- **tld.txt**(text/plain) - 11372 bytes, last modified: 11/16/2022 - 100% done

```
Saving tld.txt to tld.txt
```

```
# Internet Assigned Numbers Authority (IANA) approved list of TLD's
TLD_List = []
f=open("tld.txt","r")
for i in f:
    TLD_List.append(i.rstrip('\n').lower())
print(TLD_List)
```

```
['aaa', 'aarp', 'abarth', 'abb', 'abbott', 'abbvie', 'abc', 'able', 'abogado', 'abudhabi', 'ac
```

```
#14 Count of TLD's in the netloc



def count_TLD(c):
    count=0

    for i in c:
        if i in TLD_List:
            count+=1
    return count

df_link['totalTLD'] = df_link['url'].apply(lambda i: count_TLD(urlparse(i).netloc.split('.')))
```

Classification Features

```
#1 Use of IP or not in domain

import re
def having_ip_address(url):
    match = re.search(
        '(([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\.([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\.([01]?\\d\\d?|2[0-4]
        '([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\/)|'  # IPv4
        '((0x[0-9a-fA-F]{1,2})\\.(0x[0-9a-fA-F]{1,2})\\.(0x[0-9a-fA-F]{1,2})\\.(0x[0-9a-fA-F]{1,2})\
        '(?:[a-fA-F0-9]{1,4}:){7}[a-fA-F0-9]{1,4}', url)  # Ipv6
    if match:
        # print match.group()
        return -1
    else:
        # print 'No matching pattern found'
        return 1
df_link['use_of_ip'] = df_link['url'].apply(lambda i: having_ip_address(i))
```

```
#2 use of url shortening service
def shortening_service(url):
    match = re.search('bit\.ly|goo\.gl|shorte\.st|go2l\.ink|x\.co|ow\.ly|t\.co|tinyurl|tr\.im|is\.gd
                      'yfrog\.com|migre\.me|ff\.im|tiny\.cc|url4\.eu|twit\.ac|su\.pr|twurl\.nl|snipu
                      'short\.to|BudURL\.com|ping\.fm|post\.ly|Just\.as|bkite\.com|snipr\.com|fic\.k
                      'doiop\.com|short\.ie|kl\.am|wp\.me|rubyurl\.com|om\.ly|to\.ly|bit\.do|t\.co|l
                      'db\.tt|qr\.ae|adf\.ly|goo\.gl|bitly\.com|cur\.lv|tinyurl\.com|ow\.ly|bit\.ly|
                      'q\.gs|is\.gd|po\.st|bc\.vc|twitthis\.com|u\.to|j\.mp|buzurl\.com|cutt\.us|u\.
                      'x\.co|prettylinkpro\.com|scrnch\.me|filoops\.info|vzturl\.com|qr\.net|1url\.c
                      'tr\.im|link\.zip\.net',
                      url)
    if match:
        return -1
    else:
        return 1
df_link['short_url'] = df_link['url'].apply(lambda i: shortening_service(i))
```

```
df_link.head()
```

ength	FirstDir_length	total-	total@	total?	...	totalhttp	totalhttps	totalwww	totaldigit
0	0	0	0	0	...	1	1	1	
0	0	0	0	0	...	1	1	1	
0	0	0	0	0	...	1	1	1	
0	0	0	0	0	...	1	1	1	
0	0	0	0	0	...	1	1	1	

```
df_link.shape
```

```
(450176, 23)
```

```
df_link.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 450176 entries, 0 to 450175
Data columns (total 23 columns):
 #   Column           Non-Null Count   Dtype 
---  ------           --------------   ----- 
 0   url              450176 non-null  object
 1   label            450176 non-null  object
 2   result           450176 non-null  int64 
 3   url_length       450176 non-null  int64 
 4   hostname_length  450176 non-null  int64 
 5   path_length      450176 non-null  int64 
 6   FirstDir_length  450176 non-null  int64 
 7   total-           450176 non-null  int64 
 8   total@           450176 non-null  int64 
 9   total?           450176 non-null  int64 
 10  total%           450176 non-null  int64 
 11  total.           450176 non-null  int64 
 12  total=           450176 non-null  int64 
 13  totalhttp        450176 non-null  int64 
 14  totalhttps       450176 non-null  int64 
 15  totalwww         450176 non-null  int64 
 16  totaldigits      450176 non-null  int64 
 17  totalletters     450176 non-null  int64 
 18  totaldirs        450176 non-null  int64 
 19  totalnetlocdots  450176 non-null  int64
 20  totalTLD         450176 non-null  int64
 21  use_of_ip        450176 non-null  int64
 22  short_url        450176 non-null  int64
dtypes: int64(21), object(2)
memory usage: 79.0+ MB
```

```
df_link['label'].value_counts()
```

```
benign       345738
malicious    104438
Name: label, dtype: int64
```

```
df_link.to_csv("URL_phase1.csv")
```

Data Visualization

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

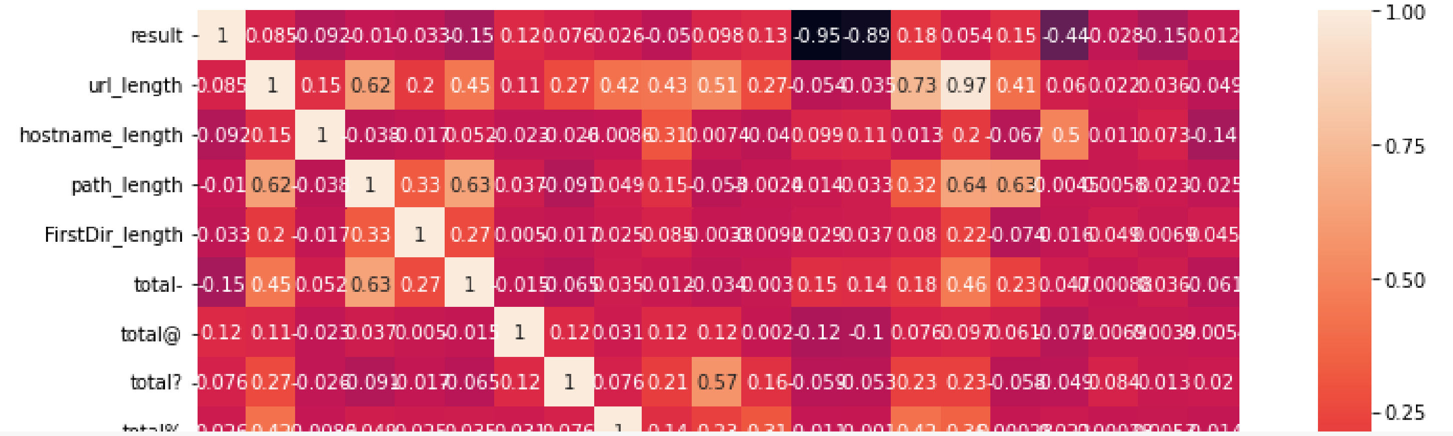
```
#1 Create a distribution plot for result
k=df[df_link['result'] == 0].count()
print(k)

plt.figure(figsize=(5,3))
sns.countplot(data=df_link, x=df_link['result'])

plt.xlabel('Types of URL')
plt.ylabel('Number of URL')
plt.title('Count of URL')
plt.text(0,350000,'abc', ha='center')
plt.text(1, )
plt.show()
```

```
url                 345738
label               345738
result              345738
url_length          345738
hostname_length     345738
path_length         345738
FirstDir_length     345738
total-              345738
total@              345738
total?              345738
total%              345738
total.              345738
total=              345738
totalhttp           345738
totalhttps          345738
totalwww            345738
totaldigits         345738
totalletters        345738
totaldirs           345738
totalnetlocdots     345738
totalTLD            345738
use_of_ip           345738
short_url           345738
dtype: int64
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-47-fb628385a499> in <module>
     10 plt.title('Count of URL')
```

EDA (Exploratory Data Analysis)

```
#1 Correlation Heatmap
corrmat = df_link.corr()
f, ax = plt.subplots(figsize=(15,10))
sns.heatmap(corrmat, square=True, annot = True, annot_kws={'size':10})
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe6a9edc850>
```



```
#2
```

Model Training

data split into test and train

```
# first lets asign a new data frame for ML purpose
df_ml=pd.read_csv('URL_phase1.csv')
```

```
df_ml.head()
```

	Unnamed: 0	url	label	result	url_length	hostname_length	path_length
0	0	https://www.google.com	benign	0	22	14	0
1	1	https://www.youtube.com	benign	0	23	15	0
2	2	https://www.facebook.com	benign	0	24	16	0
3	3	https://www.baidu.com	benign	0	21	13	0
4	4	https://www.wikipedia.org	benign	0	25	17	0

5 rows × 24 columns

```
# remove unwanted columns
df_ml.drop(['Unnamed: 0','url','label'], axis=1, inplace=True)
```

```
df_ml.head()
```

	result	url_length	hostname_length	path_length	FirstDir_length	total-	total@	total?
0	0	22	14	0	0	0	0	0
1	0	23	15	0	0	0	0	0
2	0	24	16	0	0	0	0	0
3	0	21	13	0	0	0	0	0
4	0	25	17	0	0	0	0	0

5 rows × 21 columns

```
# excluding target column from rest of dataset
X = df_ml.loc[:, df_ml.columns != 'result']
```

```
X.head()
```

	url_length	hostname_length	path_length	FirstDir_length	total-	total@	total?	total%
0	22	14	0	0	0	0	0	0
1	23	15	0	0	0	0	0	0
2	24	16	0	0	0	0	0	0
3	21	13	0	0	0	0	0	0
4	25	17	0	0	0	0	0	0

```
y=df_ml['result']
```

```
y.head()
```

```
0    0
1    0
2    0
3    0
4    0
Name: result, dtype: int64
```

```
from imblearn.over_sampling import SMOTE
oversample = SMOTE()
```

```
x_sample, y_sample = SMOTE().fit_resample(X, y.values.ravel())
```

```
print(X.shape)
print(y.shape)
print(x_sample.shape)
print(y_sample.shape)
```

```
(450176, 20)
(450176,)
```

```
(691476, 20)
(691476,)
```

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(x_sample, y_sample, test_size = 0.2)
```

```
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(553180, 20)
(553180,)
(138296, 20)
(138296,)
```

KNN

```
from sklearn.neighbors import KNeighborsClassifier
```

```
knn_model = KNeighborsClassifier(n_neighbors=5)
```

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

```
KNeighborsClassifier()
```

```
y_pred_train = knn_model.predict(X_train)
knn_pred = knn_model.predict(X_test)
```

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

```
train_acc = accuracy_score(y_train,y_pred_train)
test_acc = accuracy_score(y_test,knn_pred)
```

```
print("Accuracy of KNN on Training dataset : ",round(train_acc,3))
print("Accuracy of KNN on Testing dataset : ",round(test_acc,3))
```

```
Accuracy of KNN on Training dataset :  0.987
Accuracy of KNN on Testing dataset :  0.979
```

```
from sklearn.metrics import confusion_matrix, classification_report
print('Confusion Matrix')
print(confusion_matrix(knn_pred,y_test))
print(classification_report(knn_pred,y_test,target_names=["legit","malicious"]))
```

```
Confusion Matrix
[[67640  1143]
 [ 1747 67766]]
              precision    recall  f1-score   support
```

```
       legit       0.97      0.98      0.98     68783
   malicious       0.98      0.97      0.98     69513

    accuracy                           0.98    138296
   macro avg       0.98      0.98      0.98    138296
weighted avg       0.98      0.98      0.98    138296
```

```
#### KNN with hyper-parameter tuning
```

```
knn_model_hyper = KNeighborsClassifier(n_neighbors=10,weights='distance',algorithm='ball_tree',leaf_
```

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

```
KNeighborsClassifier(algorithm='ball_tree', leaf_size=50, metric='manhattan',
                     n_jobs=-1, n_neighbors=10, p=1, weights='distance')
```

```
y_pred_train_hyper = knn_model_hyper.predict(X_train)
knn_pred_hyper = knn_model_hyper.predict(X_test)
```

```
train_acc_hyper = accuracy_score(y_train,y_pred_train_hyper)
test_acc_hyper = accuracy_score(y_test,knn_pred_hyper)
```

```
print("Accuracy of KNN on Training dataset : ",round(train_acc_hyper,3))
print("Accuracy of KNN on Testing dataset : ",round(test_acc_hyper,3))
```

```
Accuracy of KNN on Training dataset :  0.999
Accuracy of KNN on Testing dataset :  0.989
```

```
print('Confusion Matrix')
print(confusion_matrix(knn_pred_hyper,y_test))
print(classification_report(knn_pred_hyper,y_test,target_names=["legit","malicious"]))
```

```
Confusion Matrix
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-34398ddd1c12> in <module>
      1 print('Confusion Matrix')
----> 2 print(confusion_matrix(knn_pred_hyper,y_test))
      3 print(classification_report(knn_pred_hyper,y_test,target_names=["legit","malicious"]))

NameError: name 'confusion_matrix' is not defined
```

SEARCH STACK OVERFLOW

Colab paid products - Cancel contracts here

❗ 1s completed at 11:49 AM ● ✕