```
# 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

[ ] ↳ *7 cells hidden*

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

[ ] ↳ *7 cells hidden*

Data Visualization

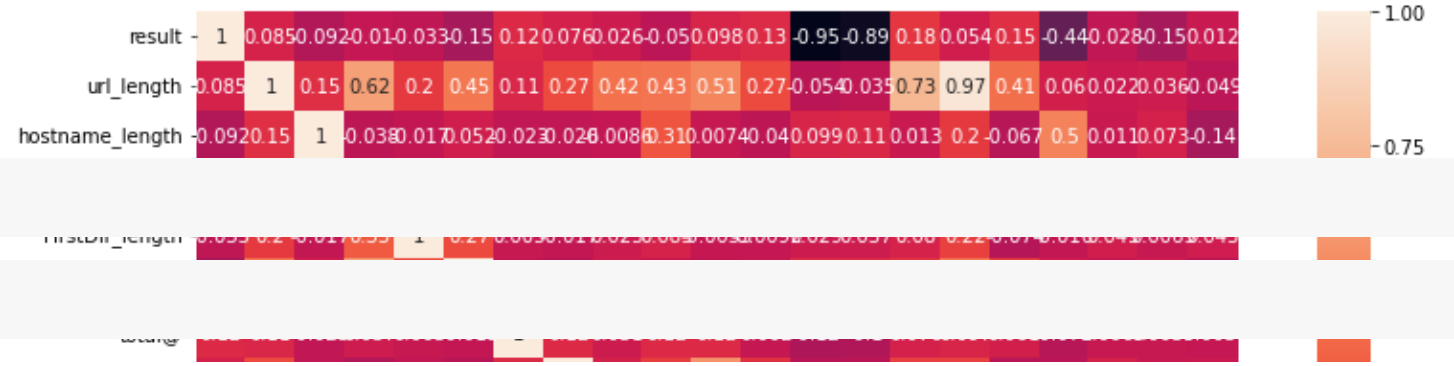
```
import seaborn as sns
import matplotlib.pyplot as plt
```

```
# #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()
```

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})
```

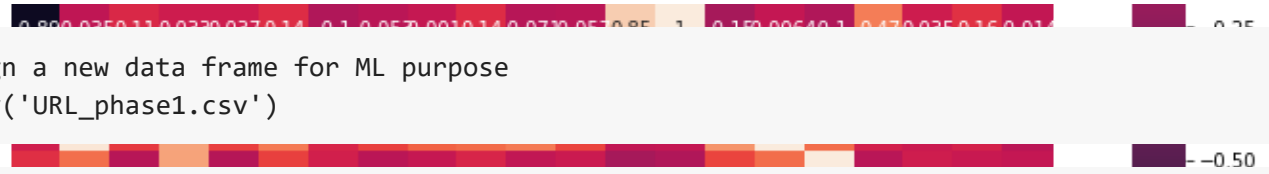
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8834aaee50>
```




```
#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	F
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

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

	0	25	17	0	0	0	0	0

```
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,)
```

Decision Tree

```
from sklearn.tree import DecisionTreeClassifier

dt_model = DecisionTreeClassifier()
```

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

```
DecisionTreeClassifier()
```

```
y_pred_train = dt_model.predict(X_train)
dt_pred = dt_model.predict(X_test)
```

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

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

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

```
Accuracy of DT on Training dataset :  0.999
Accuracy of DT on Testing dataset :  0.997
```

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

```
Confusion Matrix
[[68934   208]
 [  178 68976]]
              precision    recall  f1-score   support

  legitimate       1.00      1.00      1.00     69142
   malicious       1.00      1.00      1.00     69154

    accuracy                           1.00    138296
   macro avg       1.00      1.00      1.00    138296
weighted avg       1.00      1.00      1.00    138296
```

Random Forest

```
from sklearn.ensemble import RandomForestClassifier
```

```
rf_model = RandomForestClassifier(n_estimators=10)

# start training the model
rf_model.fit(X_train,y_train)
```

```
RandomForestClassifier(n_estimators=10)
```

```
y_pred_train = rf_model.predict(X_train)
rf_pred = rf_model.predict(X_test)
```

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

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

```
Accuracy of RF on Training dataset :  0.999
Accuracy of RF on Testing dataset :  0.998
```

Extremely Randomized Trees Classifier

```
from sklearn.ensemble import ExtraTreesClassifier
```

```
xrf_model = ExtraTreesClassifier()
xrf_model.fit(X_train,y_train)
```

```
ExtraTreesClassifier()
```

```
y_pred_train = xrf_model.predict(X_train)
xrf_pred = xrf_model.predict(X_test)
```

```
train_acc_xrf = accuracy_score(y_train,y_pred_train)
test_acc_xrf = accuracy_score(y_test,xrf_pred)
```

```
print("Accuracy of RF on Training dataset : ",round(train_acc_xrf,3))
print("Accuracy of RF on Testing dataset : ",round(test_acc_xrf,3))
```

```
Accuracy of RF on Training dataset :  0.999
Accuracy of RF on Testing dataset :  0.998
```

SVM

```
from sklearn import svm
```

Colab paid products - Cancel contracts here

0s completed at 4:08 PM