```
# importing the necessary libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,confusion_matrix,precision_score,classification_report
from sklearn.naive_bayes import MultinomialNB
import xgboost as xgb
from lightgbm import LGBMClassifier
import itertools
```

Reading the malicious csv file from drive

```
df = pd.read_csv("//content//malicious_phish.csv") #give the path of the file in your system or drive
```

```
df.head(5)
```

	url	type
0	br-icloud.com.br	phishing
1	mp3raid.com/music/krizz_kaliko.html	benign
2	bopsecrets.org/rexroth/cr/1.htm	benign
3	http://www.garage-pirenne.be/index.php?option=...	defacement
4	http://adventure-nicaragua.net/index.php?optio...	defacement

Getting the info about the datarame, checking for the duplicates and null values, then removing them

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 651191 entries, 0 to 651190
Data columns (total 2 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   url     651191 non-null  object
 1   type    651191 non-null  object
dtypes: object(2)
memory usage: 9.9+ MB
```

```
df.duplicated().sum()
```

```
10066
```

```
df = df.drop_duplicates()
```

```
df.duplicated().sum()
```

```
0
```

```
df.head()
```

	url	type
0	br-icloud.com.br	phishing
1	mp3raid.com/music/krizz_kaliko.html	benign
2	bopsecrets.org/rexroth/cr/1.htm	benign
3	http://www.garage-pirenne.be/index.php?option=...	defacement
4	http://adventure-nicaragua.net/index.php?optio...	defacement

Total number of unique categories in the DataFrame

```
df['type'].value_counts()
```

```
benign        428080
defacement     95308
phishing       94092
malware        23645
Name: type, dtype: int64
```

Getting the urls of each category in the datafame and counverting them all into a single string separated by space in order to generate the wordcloud

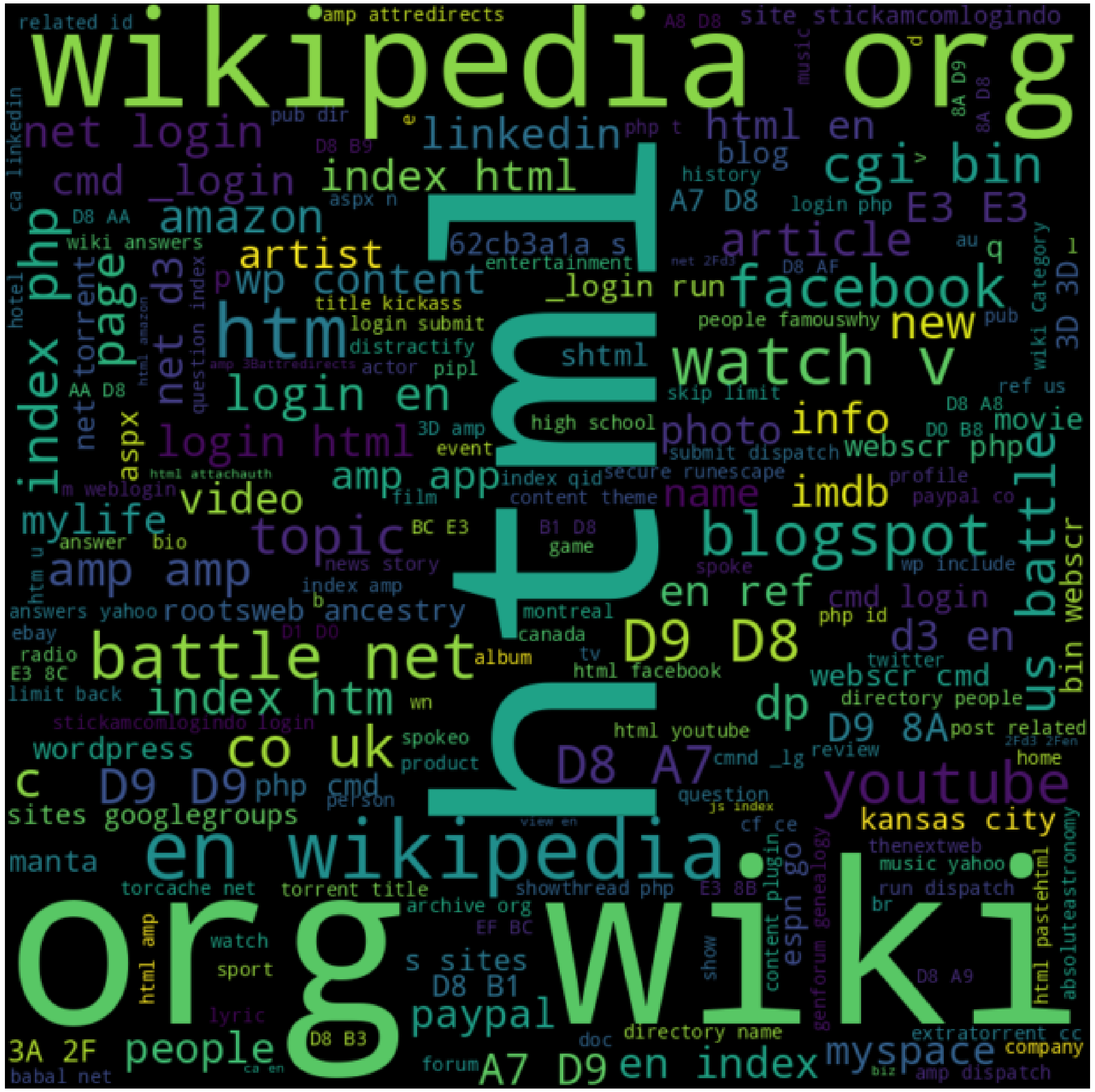
wordcloud

A word cloud is a visual representation of text data in which words are displayed in varying sizes and colors. The size of each word is typically proportional to its frequency or importance within the given text. Word clouds are a popular way to depict the most frequently occurring words in a body of text and provide a quick visual summary of the key terms

```
b = df[df['type']=="benign"]
safe = " ".join(i for i in b['url'])
```

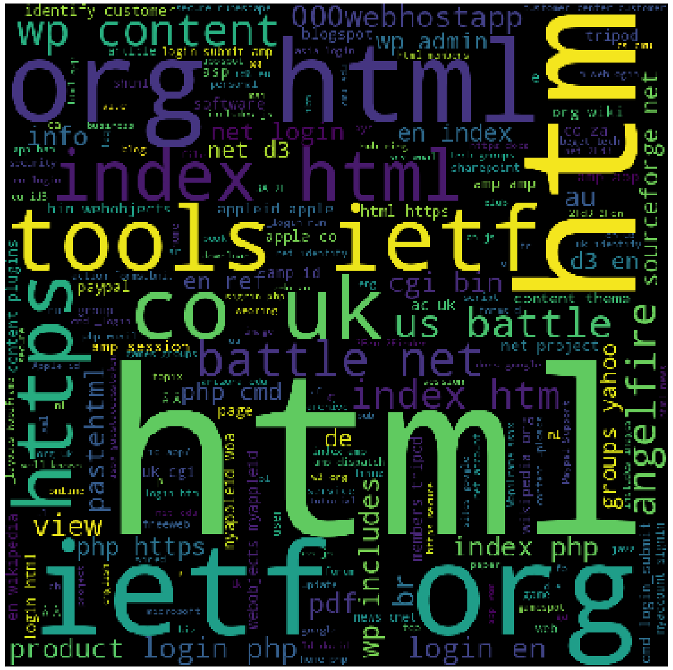
```
from wordcloud import WordCloud

wc = WordCloud(width=800,height=800,background_color= "Black").generate(safe)
plt.figure(figsize=(15,8))
plt.imshow(wc,interpolation="bilinear")
plt.axis("OFF")
plt.show()
```



```
# for phising url category
```

```
p = df[df['type']=="phishing"]
phish = " ".join(i for i in p['url'])
wc = WordCloud(width=800,height=800,background_color = "Black").generate(phish)
plt.imshow(wc,interpolation="nearest")
plt.axis('off')
plt.figure(figsize=(15,8))
plt.show()
```

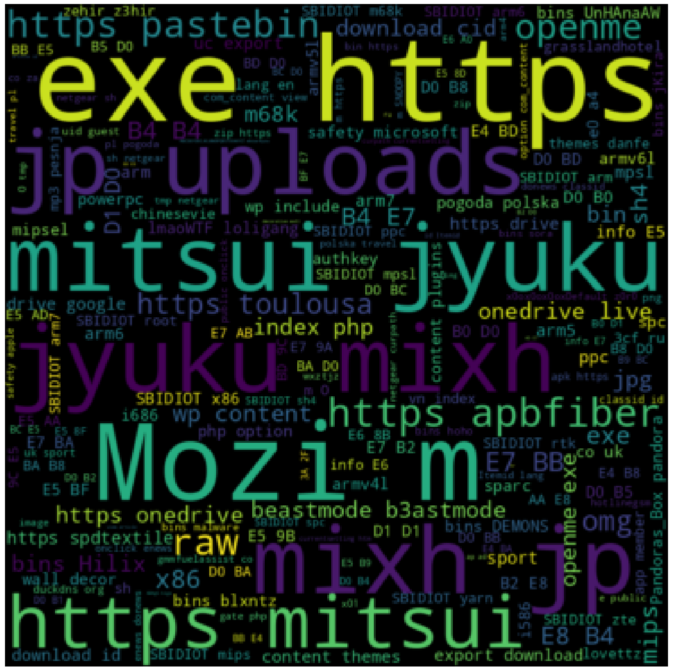


```
<Figure size 1500x800 with 0 Axes>
```

```
# for malicious url category


m = df[df['type']=="malware"]
malware = " ".join(i for i in m['url'])

wc = WordCloud(width=800,height=800,background_color = "Black").generate(malware)
plt.imshow(wc,interpolation="bilinear")
plt.axis('off')
plt.figure(figsize=(15,8))
plt.show()
```

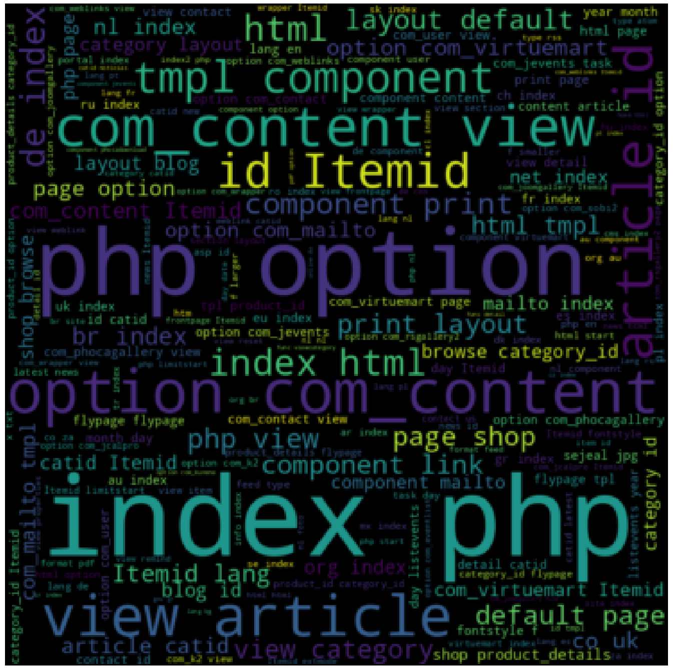


```
<Figure size 1500x800 with 0 Axes>
```

```
# for defacement url category


d = df[df['type']=="defacement"]
defacement = " ".join(i for i in d['url'])

wc = WordCloud(width=800,height=800,background_color = "Black").generate(defacement)
plt.imshow(wc,interpolation="bilinear")
plt.axis('off')
plt.figure(figsize=(15,8))
plt.show()
```



```
<Figure size 1500x800 with 0 Axes>
```

Feature Engineering

```
import re

def having_ip_address(url):
    # Define a regular expression pattern to match IPv4, IPv4 in hexadecimal, and IPv6
    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]\\d|25[0-5])\\.'
        '([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})\\/)' # IPv4 in hexadecimal
        '(?:[a-fA-F0-9]{1,4}:){7}[a-fA-F0-9]{1,4}', url)  # IPv6

    if match:
        # If the regular expression pattern is found in the URL, return 1 (indicating the presence of an IP address)
        return 1
    else:
        # If no matching pattern is found, return 0 (indicating no IP address)
        return 0



df['USE_OF_IP'] = df['url'].apply(lambda i : having_ip_address(i))
df['USE_OF_IP']
```

```
     0         0
     1         0
     2         0
     3         0
     4         0
              ..
     651186    0
     651187    0
     651188    0
     651189    0
     651190    0
     Name: USE_OF_IP, Length: 641125, dtype: int64
```

```
from  urllib.parse import urlparse
# to check whether there is a  hostname in each url
def search_for_hostname(url):
  hostname = urlparse(url).hostname
  hostname = str(hostname)

  match = re.search(hostname,url)
  if match:
    return 1
  else:
    return 0

df['Hostname'] = df['url'].apply(lambda i : search_for_hostname(i))
```

```
df['Hostname']
```

```
     0         0
     1         0
     2         0
     3         1
     4         1
              ..
     651186    0
     651187    0
     651188    0
     651189    0
     651190    0
     Name: Hostname, Length: 641125, dtype: int64
```

```
!pip install googlesearch-python
```

```
     Requirement already satisfied: googlesearch-python in /usr/local/lib/python3.10/dist-packages (1.2.3)
     Requirement already satisfied: beautifulsoup4>=4.9 in /usr/local/lib/python3.10/dist-packages (from googlesearch-python) (4.12.3)
     Requirement already satisfied: requests>=2.20 in /usr/local/lib/python3.10/dist-packages (from googlesearch-python) (2.31.0)
     Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.10/dist-packages (from beautifulsoup4>=4.9->googlesearch-python) (2.5)
     Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests>=2.20->googlesearch-python) (3
     Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests>=2.20->googlesearch-python) (3.6)
     Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests>=2.20->googlesearch-python) (2.0.7)
     Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests>=2.20->googlesearch-python) (2024.2
from googlesearch import search
# to find whether the each url is indexed in google or not, if not it is most likely to be a malicious one
def google_index(url):
    site = search(url, 5)
    return 1 if site else 0
df['google_index'] = df['url'].apply(lambda i: google_index(i))
```

```
df['google_index']
```

```
0         1
1         1
2         1
3         1
4         1
         ..
651186    1
651187    1
651188    1
651189    1
651190    1
Name: google_index, Length: 641125, dtype: int64
```

```
# counting the number of times these specific characters are occured in the url to create new columns [url analysis]

df['count-@'] = df['url'].apply(lambda x: x.count('@'))
df['count--'] = df['url'].apply(lambda x:x.count("-"))
df['count-?'] = df['url'].apply(lambda x:x.count("?"))
df['count.'] = df['url'].apply(lambda x:x.count("."))
df['count-%'] = df['url'].apply(lambda x:x.count("%"))
df['count-='] = df['url'].apply(lambda x:x.count("="))
df['count_www'] = df['url'].apply(lambda x:x.count("www"))
df['count_http'] = df['url'].apply(lambda x:x.count("http"))
df['count_https'] = df['url'].apply(lambda x:x.count("https"))
df['len_of_url'] = df['url'].apply(lambda x: len(str(x)))
df['len_of_hostname'] = df['url'].apply(lambda x: len(urlparse(x).netloc))
df['digit_count'] = df['url'].apply(lambda x: sum(1 for i in str(x) if i== i.isdigit()))
```

```
df.head()
```

	url	type	USE_OF_IP	Hostname	google_index	count-@	count--	count-?	count.	count-%	count-=	count_www	coun
0	br-icloud.com.br	phishing	0	0	1	0	1	0	2	0	0	0	
1	mp3raid.com/music/krizz_kaliko.html	benign	0	0	1	0	0	0	2	0	0	0	
2	bopsecrets.org/rexroth/cr/1.htm	benign	0	0	1	0	0	0	2	0	0	0	
3	http://www.garage-pirenne.be/index.php?option=...	defacement	0	1	1	0	1	1	3	0	4	1	
4	http://adventure-nicaragua.net/index.php?optio...	defacement	0	1	1	0	1	1	2	0	3	0	

```
#  to get the number of path segments in the given url

def count_path_embeddings(url):
    parsed = urlparse(url)
    path_segments = parsed.path.split('/')
    return len(path_segments) - 1
```

```
# test case
# a = count_path_embeddings('https://colab.research.google.com/drive/1It30516nMuvs7DKLR9zGDitak_S0U_I-#scrollTo=yCJYP9mYjpN-')
# print(a)
```

```
2
```

```
df['emb_count'] = df['url'].apply(lambda x: count_path_embeddings(x))
```

```
df['emb_count']
```

```
0         0
1         2
2         3
3         1
4         1
```

```
              ..
     651186    3
     651187    4
     651188    4
     651189    2
     651190    3
     Name: emb_count, Length: 641125, dtype: int64
```

```
df.head()
```

	url	type	USE_OF_IP	Hostname	google_index	count-@	count--	count-?	count.	count-%	count-=	count_www	coun
0	br-icloud.com.br	phishing	0	0	1	0	1	0	2	0	0	0	
1	mp3raid.com/music/krizz_kaliko.html	benign	0	0	1	0	0	0	2	0	0	0	
2	bopsecrets.org/rexroth/cr/1.htm	benign	0	0	1	0	0	0	2	0	0	0	
3	http://www.garage-pirenne.be/index.php?option=...	defacement	0	1	1	0	1	1	3	0	4	1	
4	http://adventure-nicaragua.net/index.php?optio...	defacement	0	1	1	0	1	1	2	0	3	0	

```
# to find the lenght of the first directory in the path of the given url

def fd_length_cal(url):
  parsed_url = urlparse(url).path
  length = parsed_url.split('/')
  try:
    return len(length[1])
  except Exception as e:
    return 0
df['fd_length'] = df['url'].apply(lambda x: fd_length_cal(x))
```

```
df['fd_length']
```

```
     0         0
     1         5
     2         7
```

```
3         9
4         9
         ..
651186    7
651187    8
651188    7
651189    4
651190    4
Name: fd_length, Length: 641125, dtype: int64
```

```
!pip install tld
```

```
Collecting tld
  Downloading tld-0.13-py2.py3-none-any.whl (263 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 263.8/263.8 kB 3.5 MB/s eta 0:00:00
Installing collected packages: tld
Successfully installed tld-0.13
```

```
from tld import get_tld
#  to get the top level domain in the url and its lenght (tld--> com,org,in etc...)

def get_len_tld(url):
  tld = get_tld(url,fail_silently=True)
  try:
    # print(tld)
    return len(str(tld))
    # print(tld)
  except Exception as e:
    return 0

df['Tld_lenght'] = df['url'].apply(lambda x: get_len_tld(x))
```

```
df['Tld_lenght']
```

```
0         4
1         4
2         4
3         2
4         3
         ..
651186    4
651187    4
```

```
        651188    4
        651189    4
        651190    4
        Name: Tld_lenght, Length: 641125, dtype: int64
```

```
a = get_len_tld('https://colab.research.google.com/drive/1It30516nMuvs7DKLR9zGDitak_S0U_I-#scrollTo=uth3M0ymswa5')
print(a)
```