Entrée [4]:

```
#Informations titré du livre Hands-On Machine Learning for Cybersecurity
#Author : Soma Halder and Sinan Ozdemir
#Explication seulement sans l'implementation
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns

import os
print(os.listdir("."))
import joblib
```

```
['urlchecker', '.ipynb_checkpoints', 'requirements.txt', 'manage.py',
'templates', 'db.sqlite3', 'URL-Detection Model.ipynb', '.git', 'urlda
ta.csv', 'finalized_model.sav']
```

Entrée [5]:

```
#importation de la dataset
#dataset obtenu fourni par le livre que j'ai lu
urldata = pd.read_csv("urldata.csv")
```

Entrée [6]:

```
#afficher les données pour une premiere vue
urldata.tail()
```

Out[6]:

	Unnamed: 0	url	label	result
450171	450171	http://ecct-it.com/docmmmnn/aptgd/index.php	malicious	1
450172	450172	http://faboleena.com/js/infortis/jquery/plugin...	malicious	1
450173	450173	http://faboleena.com/js/infortis/jquery/plugin...	malicious	1
450174	450174	http://atualizapj.com/	malicious	1
450175	450175	http://writeassociate.com/test/Portal/inicio/I...	malicious	1

Entrée [7]:

```
#faire une petite statistique descriptive de nos données
urldata.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 450176 entries, 0 to 450175
Data columns (total 4 columns):
 #   Column      Non-Null Count   Dtype 
---  ------      --------------   ----- 
 0   Unnamed: 0  450176 non-null  int64 
 1   url         450176 non-null  object
 2   label       450176 non-null  object
 3   result      450176 non-null  int64 
dtypes: int64(2), object(2)
memory usage: 13.7+ MB
```

Entrée [8]:

```
#Recherche de valeurs manquantes
urldata.isnull().sum()
```

Out[8]:

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

Entrée [9]:

```
#import des lib importants pour le transformation des variables
from urllib.parse import urlparse
from tld import get_tld
import os.path
```

Entrée [10]:

```
#calculer la longueur des liens
urldata['url_length'] = urldata['url'].apply(lambda i: len(str(i)))
```

Entrée [11]:

```
#extraction de la longueur de l'hostname
urldata['hostname_length'] = urldata['url'].apply(lambda i: len(urlparse(i).netloc))
```

Entrée [12]:

```
#extraction de la longueur du path
urldata['path_length'] = urldata['url'].apply(lambda i: len(urlparse(i).path))
```

Entrée [13]:

```
urldata.head()
```

Out[13]:

	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

Entrée [14]:

```
def fd_length(url):
    urlpath= urlparse(url).path
    try:
        return len(urlpath.split('/')[1])
    except:
        return 0

urldata['fd_length'] = urldata['url'].apply(lambda i: fd_length(i))
```

Entrée [15]:

```
#Longueur de TLD {TLD exemple : "com", "net","fr"}
urldata['tld'] = urldata['url'].apply(lambda i: get_tld(i,fail_silently=True))
def tld_length(tld):
    try:
        return len(tld)
    except:
        return -1

urldata['tld_length'] = urldata['tld'].apply(lambda i: tld_length(i))
```

Entrée [ ]:

```
#urldata = urldata.drop("tld",1)
```

Entrée [16]:

```
#Compter les caracteres speciaux

urldata['count-'] = urldata['url'].apply(lambda i: i.count('-'))
urldata['count@'] = urldata['url'].apply(lambda i: i.count('@'))
urldata['count?'] = urldata['url'].apply(lambda i: i.count('?'))
urldata['count%'] = urldata['url'].apply(lambda i: i.count('%'))
urldata['count.'] = urldata['url'].apply(lambda i: i.count('.'))
urldata['count='] = urldata['url'].apply(lambda i: i.count('='))
urldata['count-http'] = urldata['url'].apply(lambda i : i.count('http'))
urldata['count-https'] = urldata['url'].apply(lambda i : i.count('https'))
urldata['count-www'] = urldata['url'].apply(lambda i: i.count('www'))
```

Entrée [17]:

```
#Le nombre de chiffres dans le lien
def digit_count(url):
    digits = 0
    for i in url:
        if i.isnumeric():
            digits = digits + 1
    return digits
urldata['count-digits']= urldata['url'].apply(lambda i: digit_count(i))
```

Entrée [18]:

```
#le nombre de lettre dans le lien
def letter_count(url):
    letters = 0
    for i in url:
        if i.isalpha():
            letters = letters + 1
    return letters
urldata['count-letters']= urldata['url'].apply(lambda i: letter_count(i))
```

Entrée [19]:

```
#Compter d'arborescence
def no_of_dir(url):
    urldir = urlparse(url).path
    return urldir.count('/')
urldata['count_dir'] = urldata['url'].apply(lambda i: no_of_dir(i))
```

Entrée [20]:

```
#Verifier si le lien contient un IP
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])\\.([0
        '([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[
        '(?:[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
urldata['use_of_ip'] = urldata['url'].apply(lambda i: having_ip_address(i))
```

Entrée [21]:

```
#Si le lien est un raccourceur de lien
def shortening_service(url):
    match = re.search('bit\.ly|goo\.gl|shorte\.st|go2l\.ink|x\.co|ow\.ly|t\.co|tiny
                      'yfrog\.com|migre\.me|ff\.im|tiny\.cc|url4\.eu|twit\.ac|su\.p
                      'short\.to|BudURL\.com|ping\.fm|post\.ly|Just\.as|bkite\.com|
                      'doiop\.com|short\.ie|kl\.am|wp\.me|rubyurl\.com|om\.ly|to\.l
                      'db\.tt|qr\.ae|adf\.ly|goo\.gl|bitly\.com|cur\.lv|tinyurl\.co
                      'q\.gs|is\.gd|po\.st|bc\.vc|twitthis\.com|u\.to|j\.mp|buzurl\
                      'x\.co|prettylinkpro\.com|scrnch\.me|filoops\.info|vzturl\.co
                      'tr\.im|link\.zip\.net',
                      url)
    if match:
        return -1
    else:
        return 1
urldata['short_url'] = urldata['url'].apply(lambda i: shortening_service(i))
```

Entrée [22]:

```
urldata.head()
```

Out[22]:

	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

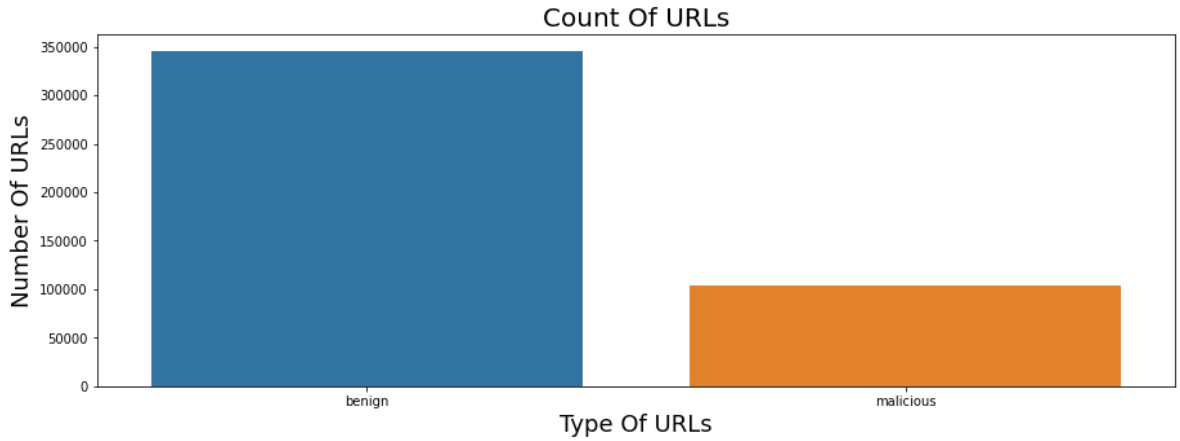
5 rows × 24 columns

Entrée [23]:

```
#Un peu de sur l'equilibrage de notre dataset
plt.figure(figsize=(15,5))
sns.countplot(x='label',data=urldata)
plt.title("Count Of URLs",fontsize=20)
plt.xlabel("Type Of URLs",fontsize=18)
plt.ylabel("Number Of URLs",fontsize=18)
```

Out[23]:

```
Text(0, 0.5, 'Number Of URLs')
```

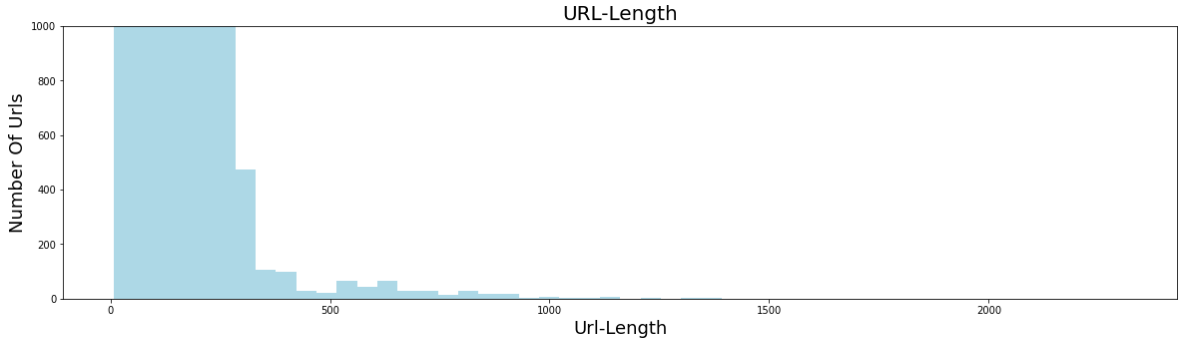


Entrée [24]:

```
#Distribution en fonction de la longueur de l'url
plt.figure(figsize=(20,5))
plt.hist(urldata['url_length'],bins=50,color='LightBlue')
plt.title("URL-Length",fontsize=20)
plt.xlabel("Url-Length",fontsize=18)
plt.ylabel("Number Of Urls",fontsize=18)
plt.ylim(0,1000)
```

Out[24]:

```
(0.0, 1000.0)
```



Entrée [25]:

```
#Import des libs pour les modeles
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split

from sklearn.metrics import confusion_matrix,classification_report,accuracy_score

from sklearn.tree import DecisionTreeClassifier

from sklearn.linear_model import LogisticRegression
```

Entrée [26]:

```
#Variables pertinentes
x = urldata[['hostname_length',
       'path_length', 'fd_length', 'tld_length', 'count-', 'count@', 'count?',
       'count%', 'count.', 'count=', 'count-http','count-https', 'count-www', 'coun
       'count-letters', 'count_dir', 'use_of_ip']]

#Variable cibles
y = urldata['result']
```

Entrée [27]:

```
#Diviser les données en données d'entrainement et de tes
x_train, x_test, y_train, y_test = train_test_split(x, y, train_size=0.3, random_st
```

Entrée [28]:

```
#Entrainer du modele arbre de decision
dt_model = DecisionTreeClassifier()
dt_model.fit(x_train,y_train)
#Evaluation du modèles
dt_predictions = dt_model.predict(x_test)
accuracy_score(y_test,dt_predictions)
```

Out[28]:

```
0.9953605564793542
```

Entrée [29]:

```
#Sauvegarde du modèles
print(confusion_matrix(y_test,dt_predictions))
filename = 'finalized_model_DT.sav'
joblib.dump(dt_model, filename)
```

```
[[241207    745]
 [   717  72455]]
```

Out[29]:

```
['finalized_model_DT.sav']
```

Entrée [30]:

```
#Entrainer de la régression logistique
log_model = LogisticRegression()
log_model.fit(x_train,y_train)
#Evalutions
log_predictions = log_model.predict(x_test)
accuracy_score(y_test,log_predictions)

filename = 'finalized_model_LR.sav'
joblib.dump(dt_model, filename)
```

```
/home/abou/anaconda3/envs/tf/lib/python3.8/site-packages/sklearn/linea
r_model/_logistic.py:762: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as show
n in:
    https://scikit-learn.org/stable/modules/preprocessing.html (http
s://scikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic
-regression (https://scikit-learn.org/stable/modules/linear_model.html
#logistic-regression)
  n_iter_i = _check_optimize_result(
```

Out[30]:

```
['finalized_model_LR.sav']
```

```
Conclusion : Le modèle obtenu à une moyenne de 0.99 qui semble etre de
l'overfitting.
J'ai pas pu faire des combinaisons de modèles afin de mieux juger du temps.
```