Malicious URLs Analytics and Analysis:

Report using Python and technologies with Machine Learning

By Giacomo Di Paolo

In this report I want to show my work about discovering malicious Urls with an AI implemented in Python environment, using different kind of algorithms for the final results, with Pandas library. The work will be explained in four detailed parts to show every aspect of my work. I will include also the scripts used in Python to get the work done.

Gathering the Data – Part 1

<part1>

First step of the work was to obtain URLs to create a first part of the datasets that will be two in total. Both of datasets will use the same file to collect Benign domains, using the "Alexa Top1MilionSites",a collection with a massive number of benign domain(a white list). Different situation for malicious domains: in first dataset I used data gathered from https://threatfeeds.io/, sites that has a lot of blacklists collection which have URLs to extract. About second dataset, I collected informations with the package(for python) DGAlgorythm that takes data from service as threatfeeds.io and generates random URLs(the reason why doing that will be explained later on). Resuming, The dataset1 and dataset2 are created in this way:

- Dataset1: In the dataset1 I mixed data from Alexa CSV file and different collection files from the website above, in specific:
    a. `http://data.phishtank.com`
    b. `https://openphish.com/`
    c. `https://urlhaus.abuse.ch`

- Dataset2: a mix with Alexa domains and generated domains with the package announced above(DGA).

Both datasets are splitted in two files: Test and Training.

The output files will be 4: for each dataset there is a file for training and a file for test. Training and test are useful for the fourth part of the project.

Now, below, I put my scripts created in Python to collect data and generate the “pre” Datasets files. The scripts contains comments that explain how the code works.

GatherData.py

```
'''

modules for gather data from txt or csv file and with dgagenerator package.
curlJsonFile takes url from a json array and put values in set.
curlXgetfiles takes txt file and extract url calling the module
    extractUrlOrIPs.
Then the getMaliciousSetDS1() function that merge sets into one (avoiding
    duplicates) and returning on preDatasets.py

below the import for the curl functions
'''
import requests
import json
import urllib
import extractURLorIPS

'''
adding the imports to call the modules needed to make dgagen() works.
'''
from datetime import date
from datetime import timedelta
from dgaGenerator.Necurs import Necurs
from dgaGenerator.Cryptolocker import Cryptolocker
from dgaGenerator.Ranbyus import Ranbyus
from dgaGenerator.ZeusBot import ZeusBot
from dgaGenerator.Torpig import Torpig
from dgaGenerator.Symmi import Symmi
# Compute domains for the current day/period:

def dgaGen():
    mergeDGA = set()
    # Compute domains for a given date:
    #x=Symmi.domainsFor(date(2020, 3, 31))
    start_date = date(2020, 4, 1)
    end_date = date(2020, 4, 17)
    delta = timedelta(days=1)
    dgaList = [Necurs,Cryptolocker,Ranbyus,ZeusBot,Torpig,Symmi]
    while start_date <= end_date:
        for i in dgaList:
            mergeDGA = mergeDGA.union(set(i.domainsFor(start_date)))
        start_date += delta
    return mergeDGA

#see the first comment
def curlJsonFile():
    setFromJson = set()
    response = requests.get('http://data.phishtank.com/data/b88098119724429c33d9525fae289ff3c6b77ab2284a53ed70213d6d3bcb5a41f/online-valid.json').json()
    for elem in response:
        setFromJson.add(elem["url"])
    return setFromJson

#see the first comment
def curlXGetFiles(url):
    file = str(urllib.request.urlopen(url).read().decode())
    return extractURLorIPS.extractURLsOrIPs(file)

#see the comment
def getMaliciousSetDS1():
    mergeUrlFiles = set()
    urls = ['https://openphish.com/feed.txt','https://urlhaus.abuse.ch/downloads/text/']
    mergeUrlFiles = mergeUrlFiles.union(curlJsonFile())
    for url in urls:
        mergeUrlFiles = mergeUrlFiles.union(curlXGetFiles(url))
    return mergeUrlFiles

'''

function that extract urls from top-1m.csv file and put into a set.
The reason why I'm putting in a set is to put everytime random lines to write on output files.
'''
def BenignInSet():
    toText = set()
    with open('benign/top-1m.csv', "r") as infile:
        for row in infile:
            toText.add(row.split(",")[1])
    return toText
```

From this code above, I just want to clarify the function extractURLorIPS.py . This is just an external function that finds all URLs in a file with regular expression. If no url in file then it takes just IPs. In my case I chose files that contains only URL in this format:

http://something.here/and/

something.there

to consider that an URL that contains IP without a domain name it’s taken; for example:

http://173.65.89.11/path/to/threat

extractURLorIPS.py

```
'''
function for regex on txt file and extract url or Ip(ips just in case I do not find any url)
'''
import re
def extractURLsOrIPs(fileContent):
    if (re.search('http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+~]|[!*\(\),]|(?:%[0-9a-fA-F][0-9a-fA-F]))+', fileContent) is not None):
        urls = set(re.findall('http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+~]|[!*\(\),]|(?:%[0-9a-fA-F][0-9a-fA-F]))+', fileContent))
        return urls
    else:
        ips = set(re.findall('(?:[\d]{1,3})\.(?:[\d]{1,3})\.(?:[\d]{1,3})\.(?:[\d]{1,3})', fileContent))
        return ips
```

This two scripts are called by another script that put all the line in the output files. The format of this preDatasets will be like this:

URL,Class

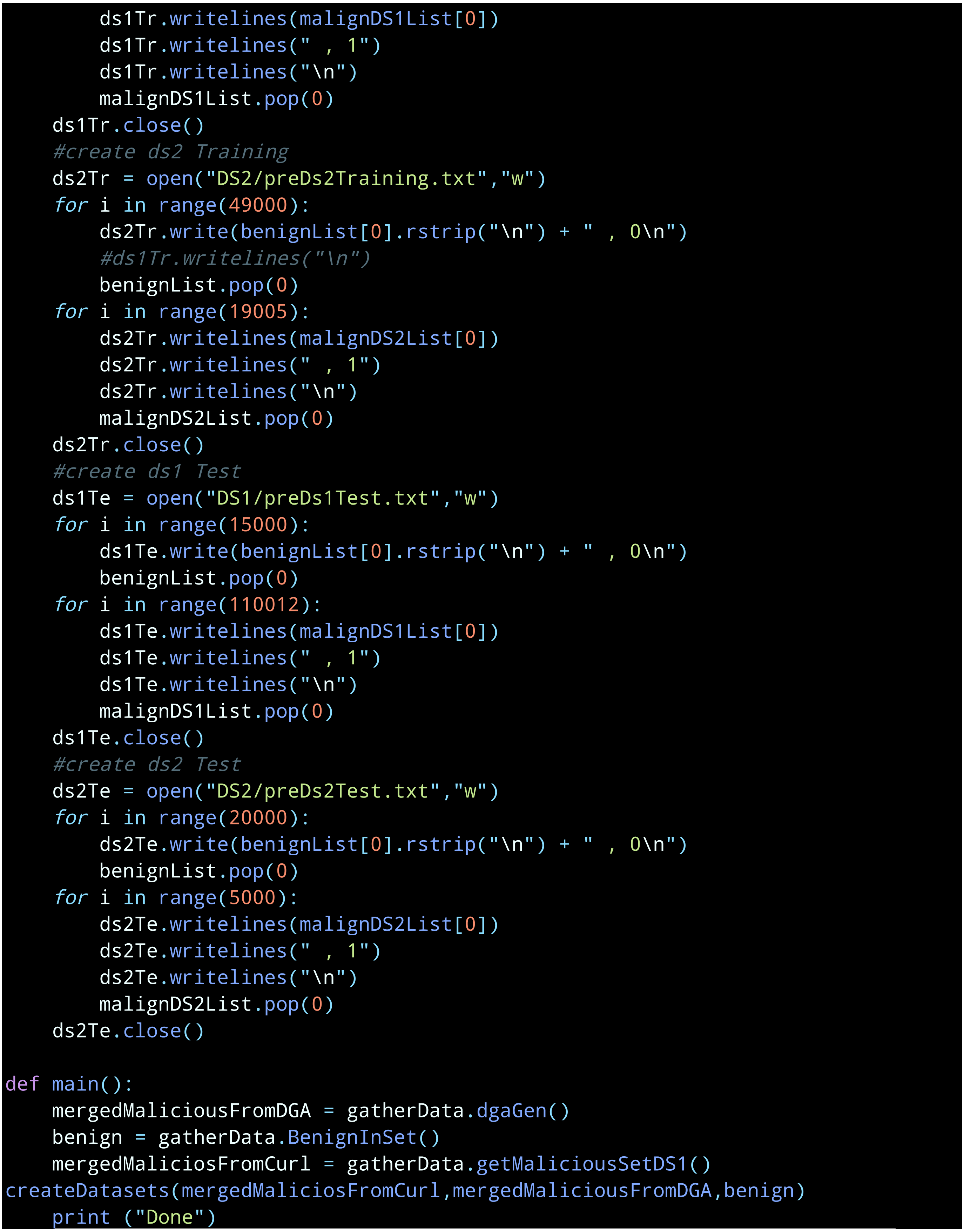
google.com,0 (benign)

http://isudfisubfis.net/!398yer/filens.exe, 1 (malign)

Down below, in the for loops, I have setted defined ranges that compose the numbers of records requested in each file.

preDatasets.py

```python
import glob
import re
import os
import test
import gatherData

'''
########################################################################################
##########################GDP CYBERINTELLIGENCE ASSIGNMENT#############################
############################started on 7th April 2020##################################
########################################################################################

this file need to start creating all the datasets from data gathered from web.
The alexa file contains all benign sites, malign sites are gathered on multiple txt file. top1mExtract.py

maliciousfromcurl.py and dgaGen(script DGAGenerator) produce sets with union function(see code) to remove duplicates.
As suggested in the paper you gave, ds1 is with file requested from url, ds2 is with dga files!

Created also a function that execute regex to be sure on having only urls or ips(if we don't have urls on txt or csv files).
Useful in first commit, but not too much now(just kept).
#urls = re.findall('http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|[!*\(\),]|(?:%[0-9a-fA-F][0-9a-fA-F]))+', page)
#ips = re.findall('(?:[\d]{1,3})\.(?:[\d]{1,3})\.(?:[\d]{1,3})\.(?:[\d]{1,3})', page)

In last part I create datasets as requested in pdf file, with the same numbers. I convert sets in lists(sets don't have index) and everytime
I add a textline in a file I delete the element from list, avoiding the risk of duplicates in different files!
'''
def createDatasets(malignDS1,malignDS2,benign):
    benignList = list(benign)
    malignDS1List = list(malignDS1)
    malignDS2List = list(malignDS2)
    #create ds1 Training
    ds1Tr = open("DS1/preDs1Training.txt","w")
    for i in range(97506):
        ds1Tr.write(benignList[0].rstrip("\n") + " , 0\n")
        #ds1Tr.writelines("\n")
        benignList.pop(0)
    for i in range(77506):
```



```
        ds1Tr.writelines(malignDS1List[0])
        ds1Tr.writelines(" , 1")
        ds1Tr.writelines("\n")
        malignDS1List.pop(0)
    ds1Tr.close()
    #create ds2 Training
    ds2Tr = open("DS2/preDs2Training.txt","w")
    for i in range(49000):
        ds2Tr.write(benignList[0].rstrip("\n") + " , 0\n")
        #ds1Tr.writelines("\n")
        benignList.pop(0)
    for i in range(19005):
        ds2Tr.writelines(malignDS2List[0])
        ds2Tr.writelines(" , 1")
        ds2Tr.writelines("\n")
        malignDS2List.pop(0)
    ds2Tr.close()
    #create ds1 Test
    ds1Te = open("DS1/preDs1Test.txt","w")
    for i in range(15000):
        ds1Te.write(benignList[0].rstrip("\n") + " , 0\n")
        benignList.pop(0)
    for i in range(110012):
        ds1Te.writelines(malignDS1List[0])
        ds1Te.writelines(" , 1")
        ds1Te.writelines("\n")
        malignDS1List.pop(0)
    ds1Te.close()
    #create ds2 Test
    ds2Te = open("DS2/preDs2Test.txt","w")
    for i in range(20000):
        ds2Te.write(benignList[0].rstrip("\n") + " , 0\n")
        benignList.pop(0)
    for i in range(5000):
        ds2Te.writelines(malignDS2List[0])
        ds2Te.writelines(" , 1")
        ds2Te.writelines("\n")
        malignDS2List.pop(0)
    ds2Te.close()

def main():
    mergedMaliciousFromDGA = gatherData.dgaGen()
    benign = gatherData.BenignInSet()
    mergedMaliciosFromCurl = gatherData.getMaliciousSetDS1()
createDatasets(mergedMaliciosFromCurl,mergedMaliciousFromDGA,benign)
    print ("Done")
```

</part1>

Creating modules to fill Datasets – Part 2

<part2>

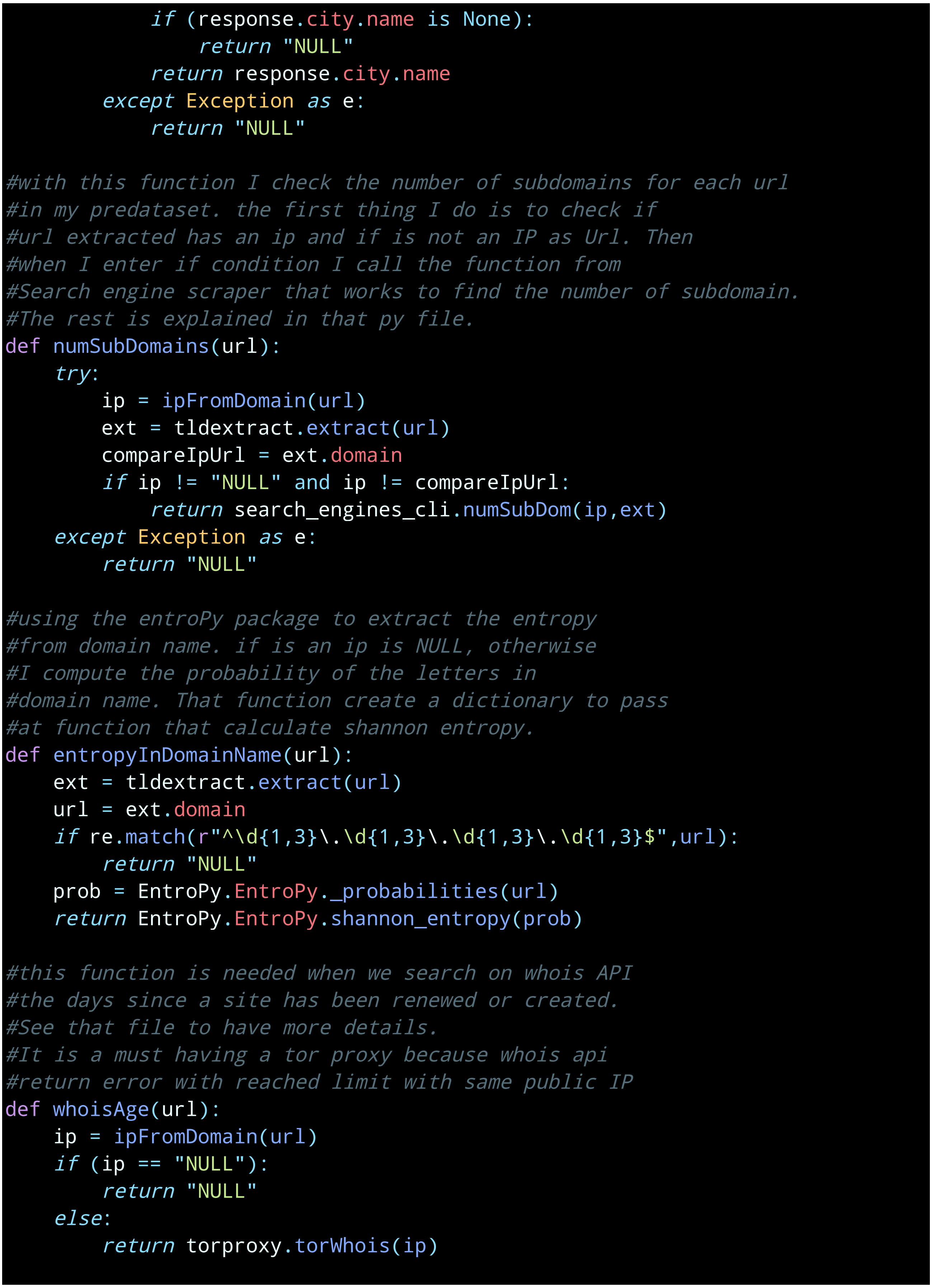
In this second part I am going to show the modules script: in few words, functions that will complete the datasets with field requested:

Domain,Entropy,Online,IP,WhoisAge,Country,City,TLD,ASN,Organisation,#SubDomains,Class

As in part1, I will expose my code commented to explain how it’s working to fill the dataset construction.

completeModules.py

```python
from urllib.parse import urlparse
import tldextract
import urllib.request
import socket
from tld import get_tld
import geoip2.database
import EntroPy.EntroPy
from collections import Counter
import requests
from requests import ConnectionError
from fake_useragent import UserAgent
import torproxy
import re
from search_engine_scraper import bing_search,yahoo_search
from SES import search_engines_cli

# this function extract the domain from URL. I added just an option
#to have always an URL with http:// format to compute as best as
#the library can (some url are just domain like google.com and I parse anyway
def extractFDQN(url):
    if not (url.startswith('//') or url.startswith('http://') or url.startswith('https://')):
        url = '//' + url
    urlTest = urlparse(url)
    return '{uri.netloc}'.format(uri=urlTest)

#check if domain is online or not. I recall extractFDQN() function and
#i do a request to see if I get 200 code(if I have a redirect I consider it
#and after response 301 check again if the redirect is online.
def isOnline(url):
    url = "http://" + extractFDQN(url)
    try:
        request = requests.get(url, allow_redirects = True, timeout = 1)
        return "Y"
    except Exception as e:
        return "N"

#this just extract the IP from domains, if available.
#There is a regex that check if the domain is an ip with port,
#and I remove the port and save the IP.
def ipFromDomain(url):
    url = extractFDQN(url)
    try:
        if re.match(r"^\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}.\d{1,5}$", url):
            ext = tldextract.extract(url)
            url = ext.domain
            s = socket.gethostbyname(url)
            return socket.gethostbyname(url)
        s = socket.gethostbyname(url)
        return socket.gethostbyname(url)
    except Exception as e:
        return "NULL"

#simple code that extract TLD. if url isn't with http:// format,
#the library needs that
def extractTld(url):
    if not (url.startswith('//') or url.startswith('http://') or
     url.startswith('https://')):
        url = 'http://' + url
    try:
        return get_tld(url)
    except Exception as e:
        return "NULL"

#for the next 4 functions I used the Geolite2 Databases to fill my dataset.
      There are
#3 db to use: 1 for ASN and Organisation, 1 for City, 1 for Countries
#the functions are all similar. I extract ip from the domain and check
#if the ip is in database
def extractASN(url):
    response = ipFromDomain(url)
    if (response == "NULL"):
        return "NULL"
    else:
        try:
            with geoip2.database.Reader('dbs/geoip2/GeoLite2-ASN.mmdb') as
     reader:
                response = reader.asn(response)
                if (response.autonomous_system_number is None):
                    return "NULL"
                return response.autonomous_system_number
        except Exception as e:
            return "NULL"

def extractASOrganization(url):
    response = ipFromDomain(url)
    if (response == "NULL"):
        return "NULL"
    else:
        try:
            with geoip2.database.Reader('dbs/geoip2/GeoLite2-ASN.mmdb') as
     reader:
                response = reader.asn(response)
                if (response.autonomous_system_organization is None):
                    return "NULL"
                return response.autonomous_system_organization
        except Exception as e:
            return "NULL"


def extractCountry(url):
    response = ipFromDomain(url)
    if (response == "NULL"):
        return "NULL"
    else:
        try:
            reader = geoip2.database.Reader('dbs/geoip2/GeoLite2-
     Country.mmdb')
            response = reader.country(response)
            if (response.country.iso_code is None):
                return "NULL"
            return response.country.iso_code
        except Exception as e:
            return "NULL"


def extractCity(url):
    response = ipFromDomain(url)
    if (response == "NULL"):
        return "NULL"
    else:
        try:
            reader = geoip2.database.Reader('dbs/geoip2/GeoLite2-City.mmdb')
            response = reader.city(response)
```

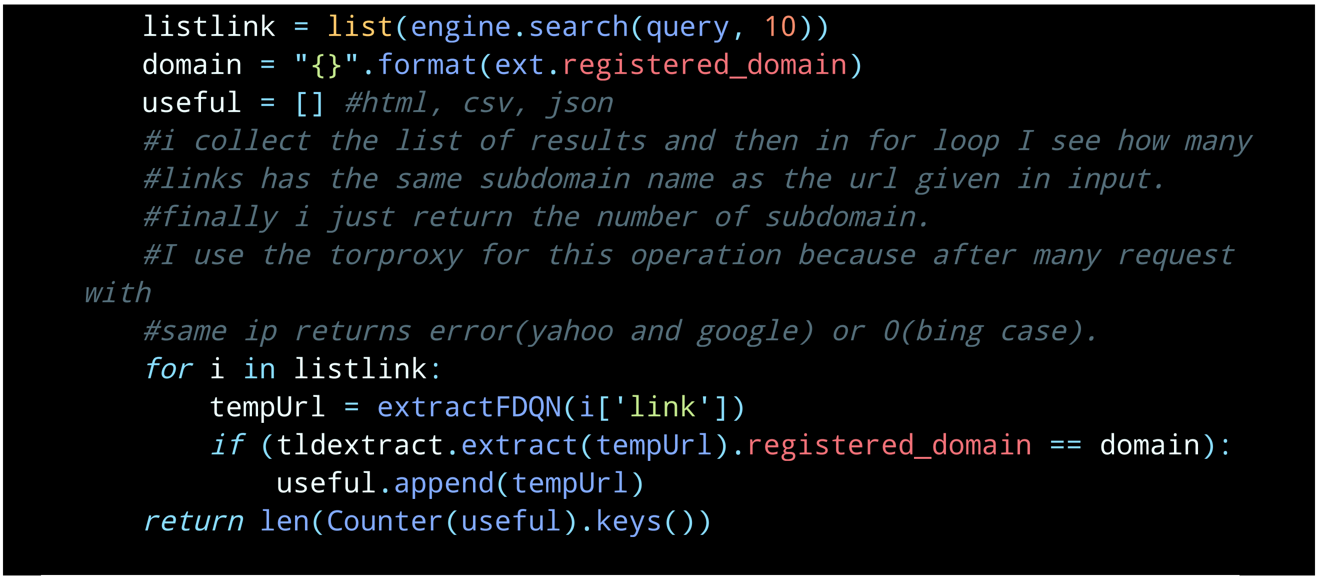


```python
            if (response.city.name is None):
                return "NULL"
            return response.city.name
        except Exception as e:
            return "NULL"

#with this function I check the number of subdomains for each url
#in my predataset. the first thing I do is to check if
#url extracted has an ip and if is not an IP as Url. Then
#when I enter if condition I call the function from
#Search engine scraper that works to find the number of subdomain.
#The rest is explained in that py file.
def numSubDomains(url):
    try:
        ip = ipFromDomain(url)
        ext = tldextract.extract(url)
        compareIpUrl = ext.domain
        if ip != "NULL" and ip != compareIpUrl:
            return search_engines_cli.numSubDom(ip,ext)
    except Exception as e:
        return "NULL"

#using the entroPy package to extract the entropy
#from domain name. if is an ip is NULL, otherwise
#I compute the probability of the letters in
#domain name. That function create a dictionary to pass
#at function that calculate shannon entropy.
def entropyInDomainName(url):
    ext = tldextract.extract(url)
    url = ext.domain
    if re.match(r"^\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}$",url):
        return "NULL"
    prob = EntroPy.EntroPy._probabilities(url)
    return EntroPy.EntroPy.shannon_entropy(prob)

#this function is needed when we search on whois API
#the days since a site has been renewed or created.
#See that file to have more details.
#It is a must having a tor proxy because whois api
#return error with reached limit with same public IP
def whoisAge(url):
    ip = ipFromDomain(url)
    if (ip == "NULL"):
        return "NULL"
    else:
        return torproxy.torWhois(ip)
```

Now the two different scripts needed for **numSubDomain(url)** and **whoisAge(url):**

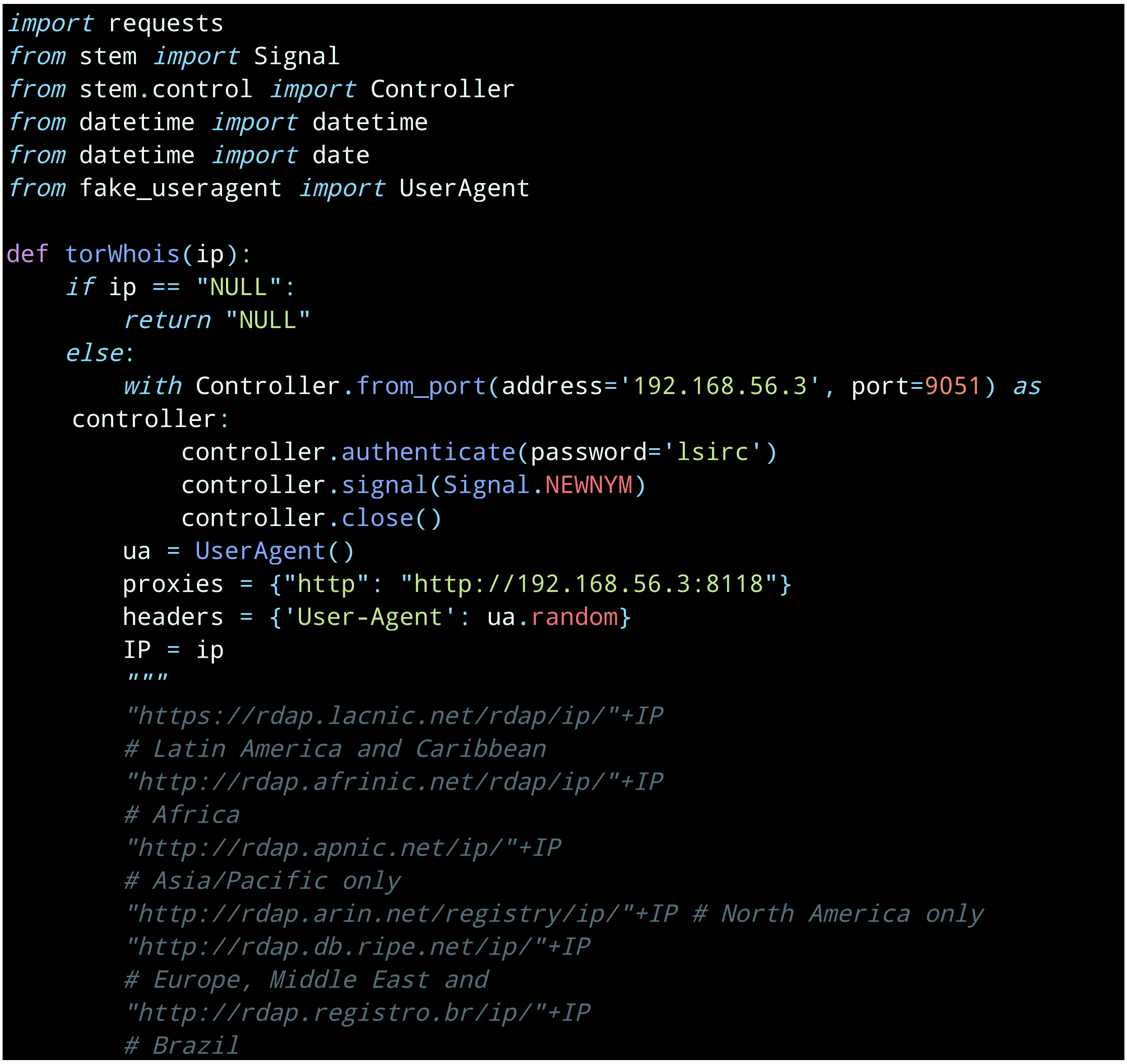
search_engine_cli.py (inside SES folder)

```
import argparse
import csv
from stem import Signal
from stem.control import Controller
import re
import tldextract
from modulesCompleteDS import extractFDQN
from collections import Counter
try:
    from search_engines.engines import search_engines_dict
    from search_engines.multiple_search_engines import MultipleSearchEngines,
    AllSearchEngines
    from search_engines import config
except ImportError as e:
    msg = '"{}"\nPlease install `search_engines` to resolve this error.'
    raise ImportError(msg.format(str(e)))

# with help of a tor proxy I just use bing dorks to see with an ip
#as query ip:"172.34.55.97" (example) how many numbers of subdomain has.
#I bring with me ip and ext value from completeModules.py with numSubDomain()
#function.
def numSubDom(ip,ext):
    with Controller.from_port(address='192.168.56.3', port=9051) as
    controller:
        controller.authenticate(password='lsirc')
        controller.signal(Signal.NEWNYM)
        controller.close()

    proxy="http://192.168.56.3:8118"
    timeout=config.TIMEOUT + (10 * bool(proxy))
    engines=['bing']  # ['all','google','bing']
    #IP="193.136.58.218"
    if not engines:
        print('Please choose a search engine: ' + ',
    '.join(search_engines_dict))
    else:
        if 'all' in engines:
            engine = AllSearchEngines(proxy, timeout)
        elif len(engines) > 1:
            engine = MultipleSearchEngines(engines, proxy, timeout)
        else:
            engine = search_engines_dict[engines[0]](proxy, timeout)
        filter = None  #[url, title, text, host]', default=None
        if filter:
            engine.set_search_operator(filter)
        query="ip:\"" + ip + "\""
        pages=10
```



```
        listlink = list(engine.search(query, 10))
        domain = "{}".format(ext.registered_domain)
        useful = [] #html, csv, json
        #i collect the list of results and then in for loop I see how many
        #links has the same subdomain name as the url given in input.
        #finally i just return the number of subdomain.
        #I use the torproxy for this operation because after many request
    with
        #same ip returns error(yahoo and google) or 0(bing case).
        for i in listlink:
            tempUrl = extractFDQN(i['link'])
            if (tldextract.extract(tempUrl).registered_domain == domain):
                useful.append(tempUrl)
        return len(Counter(useful).keys())
```

torproxy.py



```
import requests
from stem import Signal
from stem.control import Controller
from datetime import datetime
from datetime import date
from fake_useragent import UserAgent

def torWhois(ip):
    if ip == "NULL":
        return "NULL"
    else:
        with Controller.from_port(address='192.168.56.3', port=9051) as
    controller:
            controller.authenticate(password='lsirc')
            controller.signal(Signal.NEWNYM)
            controller.close()
        ua = UserAgent()
        proxies = {"http": "http://192.168.56.3:8118"}
        headers = {'User-Agent': ua.random}
        IP = ip
        """
        "https://rdap.lacnic.net/rdap/ip/"+IP
        # Latin America and Caribbean
        "http://rdap.afrinic.net/rdap/ip/"+IP
        # Africa
        "http://rdap.apnic.net/ip/"+IP
        # Asia/Pacific only
        "http://rdap.arin.net/registry/ip/"+IP # North America only
        "http://rdap.db.ripe.net/ip/"+IP
        # Europe, Middle East and
        "http://rdap.registro.br/ip/"+IP
        # Brazil
        """
        #in this case this function is called by whoisAge in
    completeModules.py
        #using torproxy.py I check how many years ago(expressed in days) the
    url
        # was renewed or created. I access the json file and then I search
    on
        # field events that contain the needed information.
        # I take always the first position because in all cases, if exist,
        # the first place is for renewals, if don't, is a created value.
        # Then I do calculate to return the value expressed in days.
        #the URL List variable is needed to verify the ip in different
    databases
        #because not everyone contains information I need.
        now = datetime.now().date()
        URL = ["http://rdap.db.ripe.net/ip/" +
    IP,"http://rdap.registro.br/ip/" +
    IP,"http://rdap.arin.net/registry/ip/" +
    IP,"http://rdap.apnic.net/ip/"+IP, "https://rdap.lacnic.net/rdap/ip/" +
    IP]
        try:
            for url in URL:
                r = requests.get(url, proxies=proxies, headers=headers)
                if r.status_code != 404:
                    r = r.json()
                    if r['events'][0]:
                        age =
    datetime.strptime(r['events'][0]['eventDate'][0:10], "%Y-%m-%d").date()
                        days = now - age
                        return days.days
        except Exception as e:
            return "NULL"
```

The second part is now concluded. In the next short step I'll explain how to create the finals datasets.

</part2>

The Datasets – Part 3

<part3>

In this short part I'll just put the code that shows how I am going to create the files we need to work in Pandas library with Python. I have to admit that my way, below, it's not efficient in terms of time because there are a huge amount of records between the datasets(almost 400.000 records in total). The right way is to use threads to split the work and going more faster. Anyway, from my experience, the problem of time it is not only about brute power, but also about speed response in network: if the network is very slow, the combination between the modules(showed in part2) will make very complex to complete the files in a short term.

finalDSet.py

```
import modulesCompleteDS
from modulesCompleteDS import extractFDQN
from modulesCompleteDS import ipFromDomain
from modulesCompleteDS import entropyInDomainName
from modulesCompleteDS import extractTld
from modulesCompleteDS import extractCountry
from modulesCompleteDS import extractCity
from modulesCompleteDS import extractASOrganization
from modulesCompleteDS import extractASN
from modulesCompleteDS import numSubDomains
from modulesCompleteDS import isOnline
from modulesCompleteDS import whoisAge
import csv

#run this as python script without any function inside.
#this is going to simply show how i get the data to fill fields
#inside the csv file. I just take the url from preDS.txt files and
#recall every function in completeModules.py for the corrispective
#field.

finalDS2 = open('DS2/finalds2.csv' , 'a')
with finalDS2:
    writer = csv.writer(finalDS2)
    firstRow =
       ["Domain","Entropy","Online","IP","WhoisAge","Country","City","TLD","ASN",
       "Organisation","#SubDomains","Class"]
    writer.writerow(firstRow)
    preDS2Test = open("DS2/preDs2Test.txt","r")
    for line in preDS2Test:
        fields = line.split(" , ")
```

```
         newRow =
        [extractFDQN(fields[0]),entropyInDomainName(fields[0]),isOnline(fields[0])
        ,ipFromDomain(fields[0]),whoisAge(fields[0]),

        extractCountry(fields[0]),extractCity(fields[0]),extractTld(fields[0]),ext
        ractASN(fields[0]),
             extractASOrganization(fields[0]),
        numSubDomains(fields[0]),fields[1].strip("\n")]
         writer.writerow(newRow)
    preDS2Test.close()
    preDS2Training = open("DS2/preDs2Training.txt","r")
    for line in preDS2Training:
         fields = line.split(" , ")
         newRow =
        [extractFDQN(fields[0]),entropyInDomainName(fields[0]),isOnline(fields[0])
        ,ipFromDomain(fields[0]),whoisAge(fields[0]),

        extractCountry(fields[0]),extractCity(fields[0]),extractTld(fields[0]),ext
        ractASN(fields[0]),
             extractASOrganization(fields[0]),
        numSubDomains(fields[0]),fields[1].strip("\n")]
         writer.writerow(newRow)
    preDS2Training.close()
finalDS2.close()

finalDS1 = open('DS1/finalds1.csv' , 'a')
with finalDS1:
    writer = csv.writer(finalDS1)
    firstRow =
        ["Domain","Entropy","Online","IP","WhoisAge","Country","City","TLD","ASN",
        "Organisation","#SubDomains","Class"]
    writer.writerow(firstRow)
    preDS1Test = open("DS1/preDs1Test.txt","r")
    for line in preDS1Test:
         fields = line.split(" , ")
         newRow =
        [extractFDQN(fields[0]),entropyInDomainName(fields[0]),isOnline(fields[0])
        ,ipFromDomain(fields[0]),whoisAge(fields[0]),

        extractCountry(fields[0]),extractCity(fields[0]),extractTld(fields[0]),ext
        ractASN(fields[0]),
             extractASOrganization(fields[0]), numSubDomains(fields[0]),
        fields[1].strip("\n")]
         writer.writerow(newRow)
    preDS1Test.close()
    preDS1Training = open("DS1/preDs1Training.txt","r")
    for line in preDS1Training:
         fields = line.split(" , ")
         newRow =
        [extractFDQN(fields[0]),entropyInDomainName(fields[0]),isOnline(fields[0])
        ,ipFromDomain(fields[0]),whoisAge(fields[0]),
        extractCountry(fields[0]),extractCity(fields[0]),extractTld(fields[0]),extractASN(fields[0]),
             extractASOrganization(fields[0]), numSubDomains(fields[0]),
        fields[1].strip("\n")]
         writer.writerow(newRow)
    preDS1Training.close()
finalDS1.close()
```

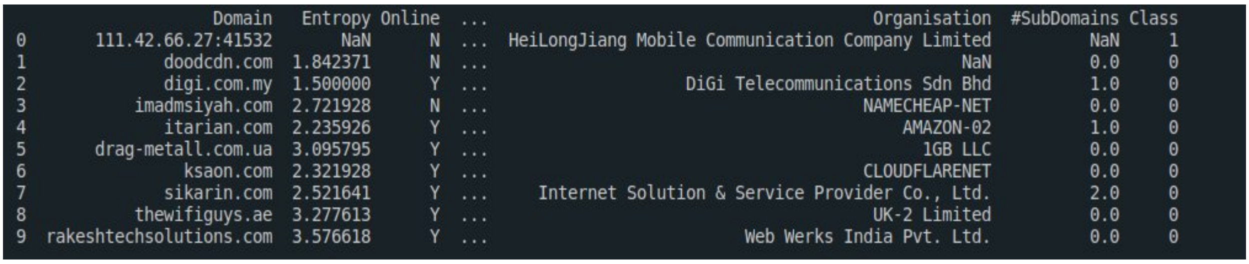
</part3>

Exploratory & ML Analysis – Part 4

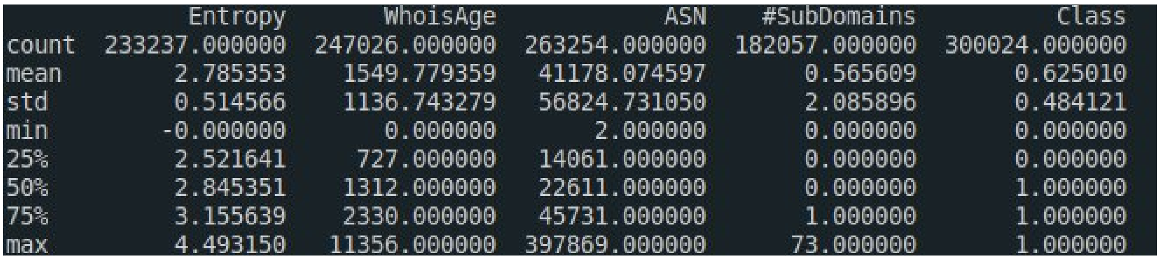
<part4>

In this part I want to show the analysis about the gathered and filled datasets. First I do the exploratory analysis to see how the data is distributed.

Dataset1



```
                 Domain   Entropy Online  ...                                   Organisation  #SubDomains Class
0       111.42.66.27:41532       NaN      N  ...  HeiLongJiang Mobile Communication Company Limited          NaN     1
1              doodcdn.com  1.842371      N  ...                                            NaN          0.0     0
2             digi.com.my  1.500000      Y  ...                   DiGi Telecommunications Sdn Bhd          1.0     0
3           imadmsiyah.com  2.721928      N  ...                                    NAMECHEAP-NET          0.0     0
4              itarian.com  2.235926      Y  ...                                        AMAZON-02          1.0     0
5        drag-metall.com.ua  3.095795      Y  ...                                          1GB LLC          0.0     0
6                ksaon.com  2.321928      Y  ...                                    CLOUDFLARENET          0.0     0
7              sikarin.com  2.521641      Y  ...      Internet Solution & Service Provider Co., Ltd.          2.0     0
8           thewifiguys.ae  3.277613      Y  ...                                     UK-2 Limited          0.0     0
9  rakeshtechsolutions.com  3.576618      Y  ...                          Web Werks India Pvt. Ltd.          0.0     0
```



```
             Entropy      WhoisAge            ASN    #SubDomains          Class
count  233237.000000  247026.000000  263254.000000  182057.000000  300024.000000
mean        2.785353    1549.779359   41178.074597       0.565609       0.625010
std         0.514566    1136.743279   56824.731050       2.085896       0.484121
min        -0.000000       0.000000       2.000000       0.000000       0.000000
25%         2.521641     727.000000   14061.000000       0.000000       0.000000
50%         2.845351    1312.000000   22611.000000       0.000000       1.000000
75%         3.155639    2330.000000   45731.000000       1.000000       1.000000
max         4.493150   11356.000000  397869.000000      73.000000       1.000000
```

We can see that we have missing values, in different columns and I choose how to fill the data :

- Entropy: for this column , this missing values are correlated to the domain as IP; entropy is calculated on domain name. I assume that value as the max of the column.
- IP: just putting 'None' value because the domain it's offline or probably the domain it's not used yet.
- Country: the IP value is NULL and if we have IP it's not in GeoLite database and return NULL(None value).
- City: Same as Country.
- WhoisAge: expressed in days. In this case if IP is NULL it returns NULL; if IP is ok then if Whois Database has records return the age expressed in days. It Could happens that the IP is not in

database and it return NULL values. If we have no values I used the approch to fill values with quantile function taking the 10% value of all values.

- ASN: same as Country and City.
- Organisation: same as ASN.
- #SubDomains: if we have NULL value we insert 0.

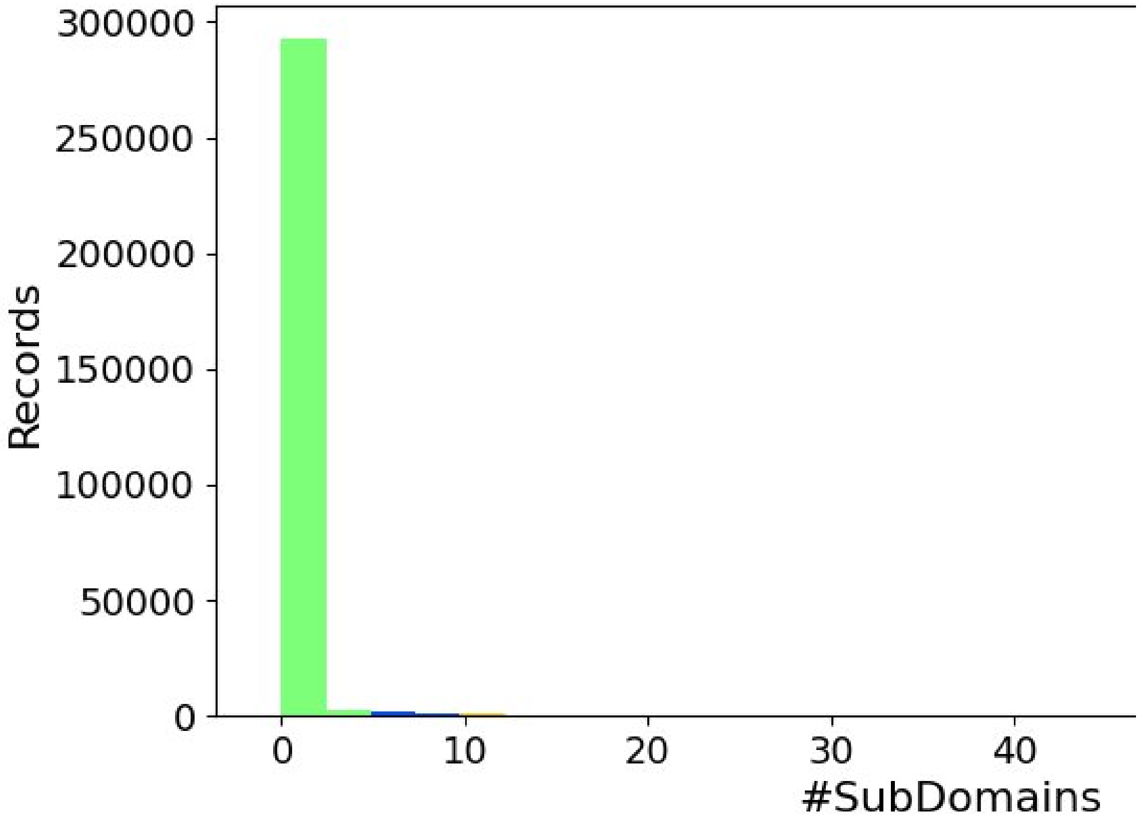
Some information about gathered data.

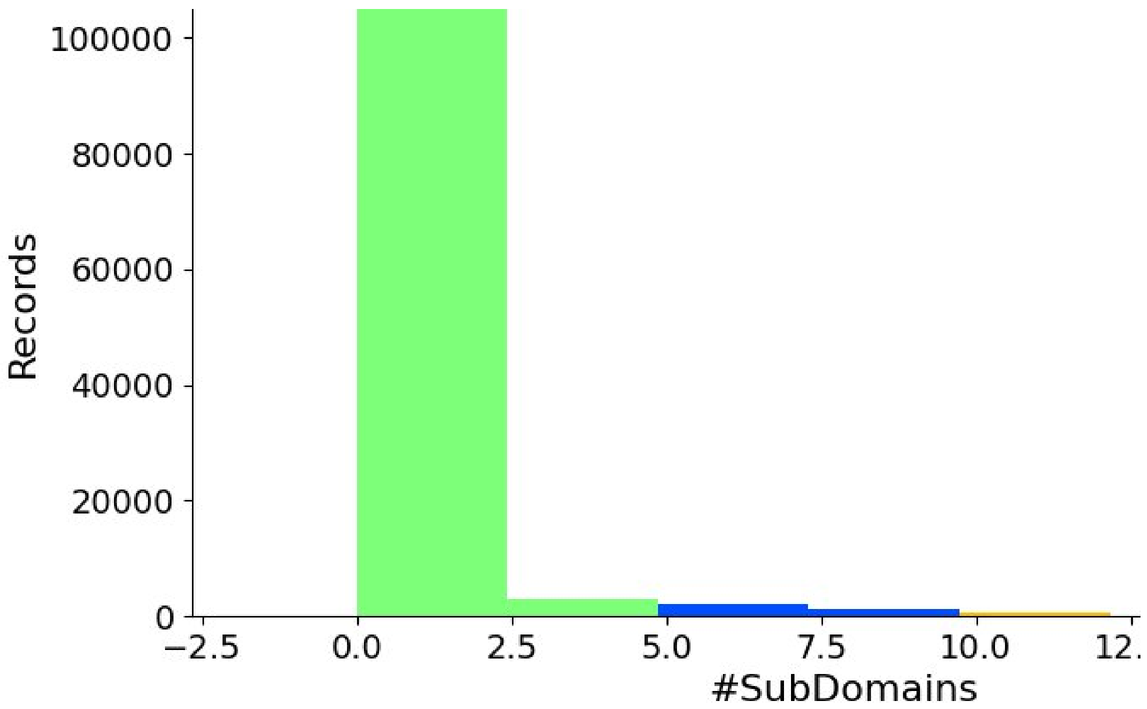
US is the country with the most Malicious and Benign domain. This is probably because the gathered data.

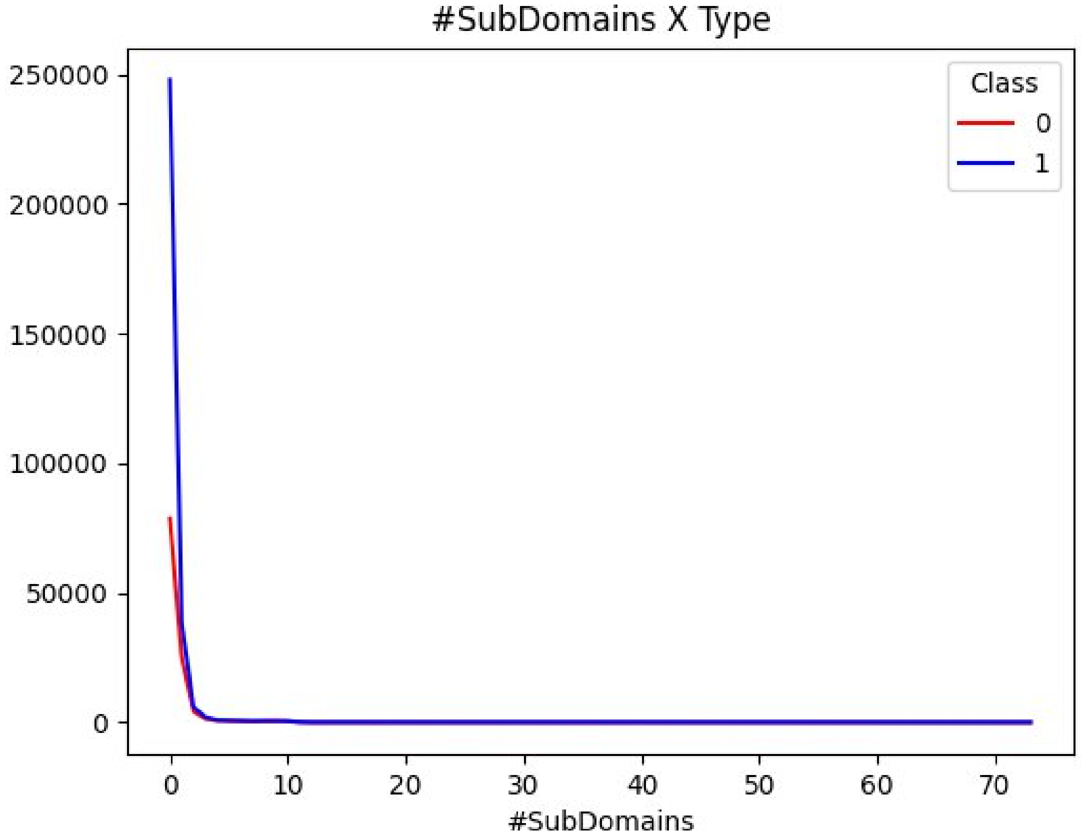
The Organisation with most benign sites is CloudFlareNet; DigitalOcean-ASN is the organisation that hosts the most number of malicious domain.

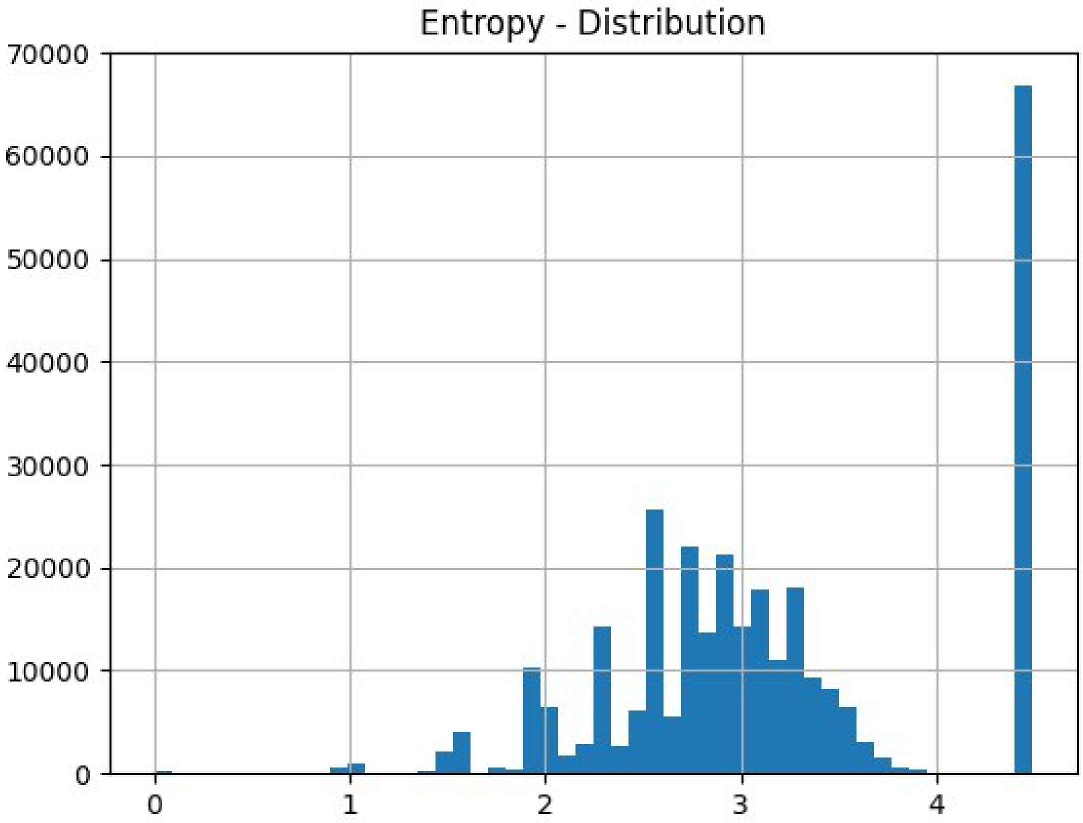
TLD for both is .com.

Now see some images about distributions:

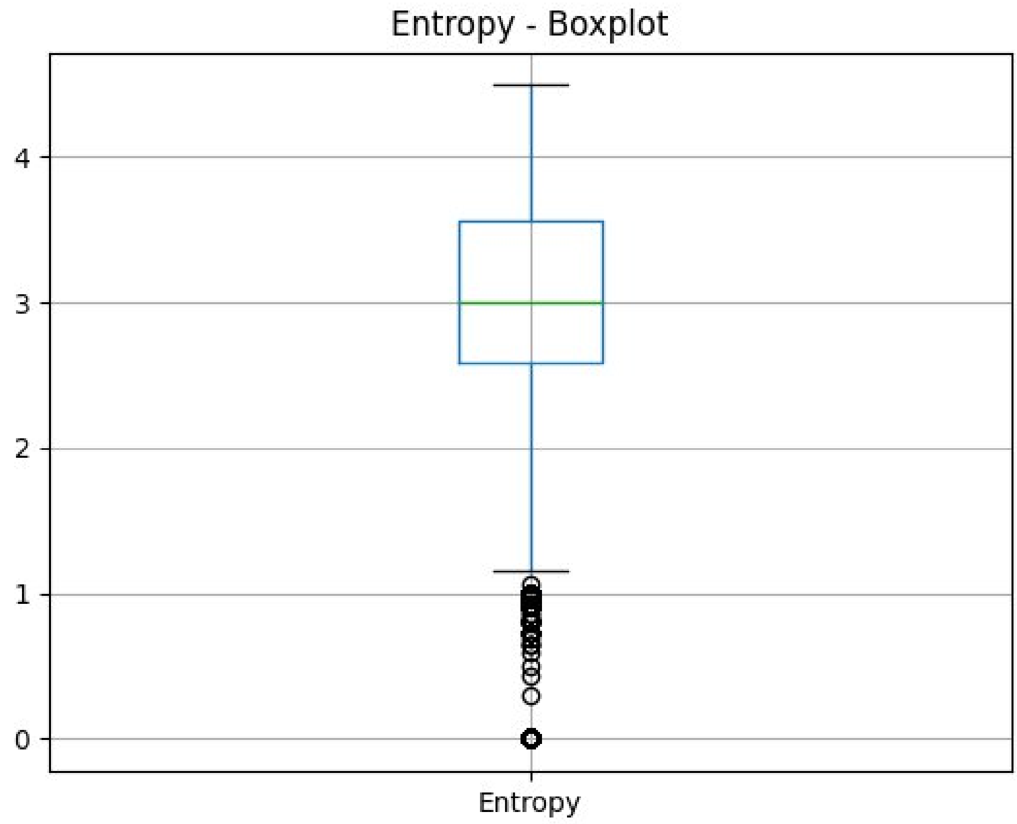



Records
100000
80000
60000
40000
20000
0
−2.5
0.0
2.5
5.0
7.5
10.0
12.
#SubDomains


#SubDomains X Type
250000
200000
150000
100000
50000
0
0
10
20
30
40
50
60
70
#SubDomains
Class
0
1

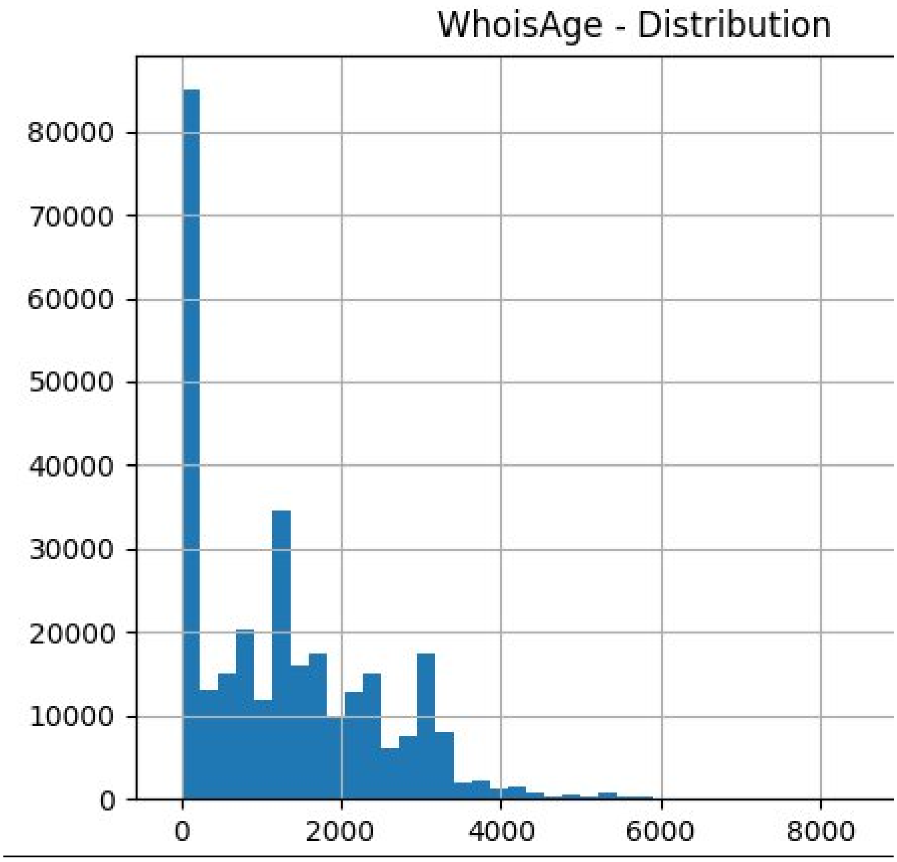


Entropy - Distribution

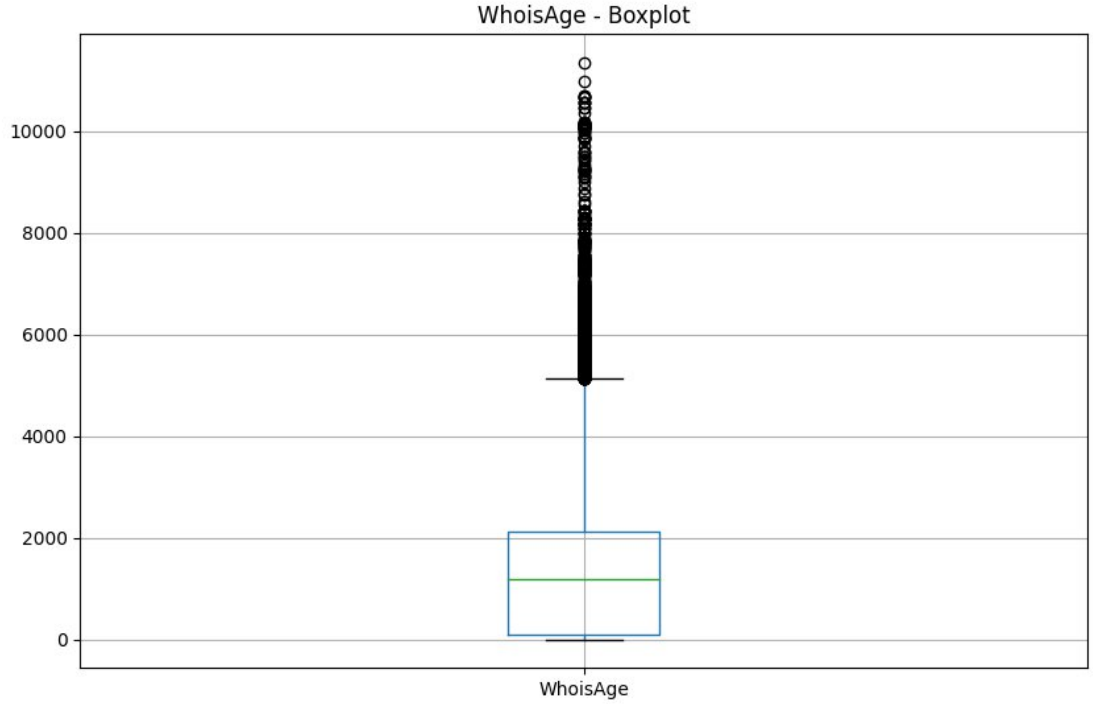


Entropy - Boxplot

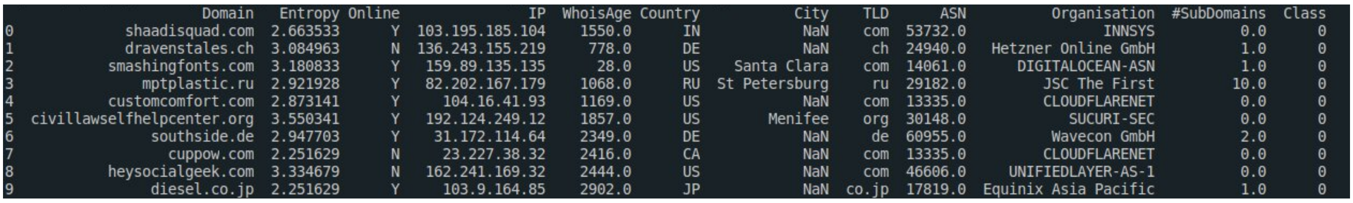
Fun fuct for Entropy. The highest value of the column is retained by a domain considered Benign.



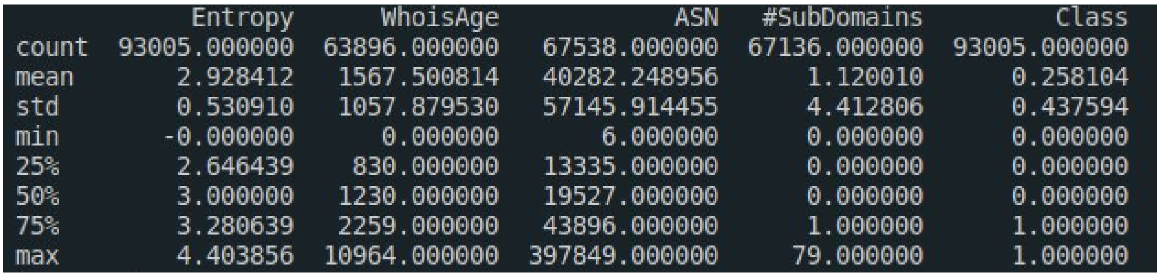
Highest concentration is between 0 and 200 days.



Dataset2



```
                       Domain   Entropy Online               IP  WhoisAge Country           City    TLD      ASN           Organisation  #SubDomains  Class
0             shaadisquad.com  2.663533      Y  103.195.185.104    1550.0      IN            NaN    com  53732.0                 INNSYS          0.0      0
1             dravenstales.ch  3.084963      N  136.243.155.219     778.0      DE            NaN     ch  24940.0    Hetzner Online GmbH          1.0      0
2           smashingfonts.com  3.180833      Y   159.89.135.135      28.0      US    Santa Clara    com  14061.0     DIGITALOCEAN-ASN          1.0      0
3             mptplastic.ru  2.921928      Y   82.202.167.179    1068.0      RU  St Petersburg     ru  29182.0          JSC The First         10.0      0
4           customcomfort.com  2.873141      Y     104.16.41.93    1169.0      US            NaN    com  13335.0          CLOUDFLARENET          0.0      0
5  civillawselfhelpcenter.org  3.550341      Y   192.124.249.12    1857.0      US        Menifee    org  30148.0              SUCURI-SEC          0.0      0
6                southside.de  2.947703      Y    31.172.114.64    2349.0      DE            NaN     de  60955.0            Wavecon GmbH          2.0      0
7                  cuppow.com  2.251629      N     23.227.38.32    2416.0      CA            NaN    com  13335.0          CLOUDFLARENET          0.0      0
8           heysocialgeek.com  3.334679      N   162.241.169.32    2444.0      US            NaN    com  46606.0       UNIFIEDLAYER-AS-1          0.0      0
9                diesel.co.jp  2.251629      Y     103.9.164.85    2902.0      JP            NaN  co.jp  17819.0  Equinix Asia Pacific          1.0      0
```



```
            Entropy      WhoisAge            ASN   #SubDomains         Class
count  93005.000000  63896.000000   67538.000000  67136.000000  93005.000000
mean       2.928412   1567.500814   40282.248956      1.120010      0.258104
std        0.530910   1057.879530   57145.914455      4.412806      0.437594
min       -0.000000      0.000000       6.000000      0.000000      0.000000
25%        2.646439    830.000000   13335.000000      0.000000      0.000000
50%        3.000000   1230.000000   19527.000000      0.000000      0.000000
75%        3.280639   2259.000000   43896.000000      1.000000      1.000000
max        4.403856  10964.000000  397849.000000     79.000000      1.000000
```

Here the situation is a bit different because we have a massive fields of NULL values because of the generated domains. I filled the missing data at the same way except for WhoisAge because of small amount of records and because we have generated domains (supposed to be like not used domain to be created for malicious purpose in a future moment).

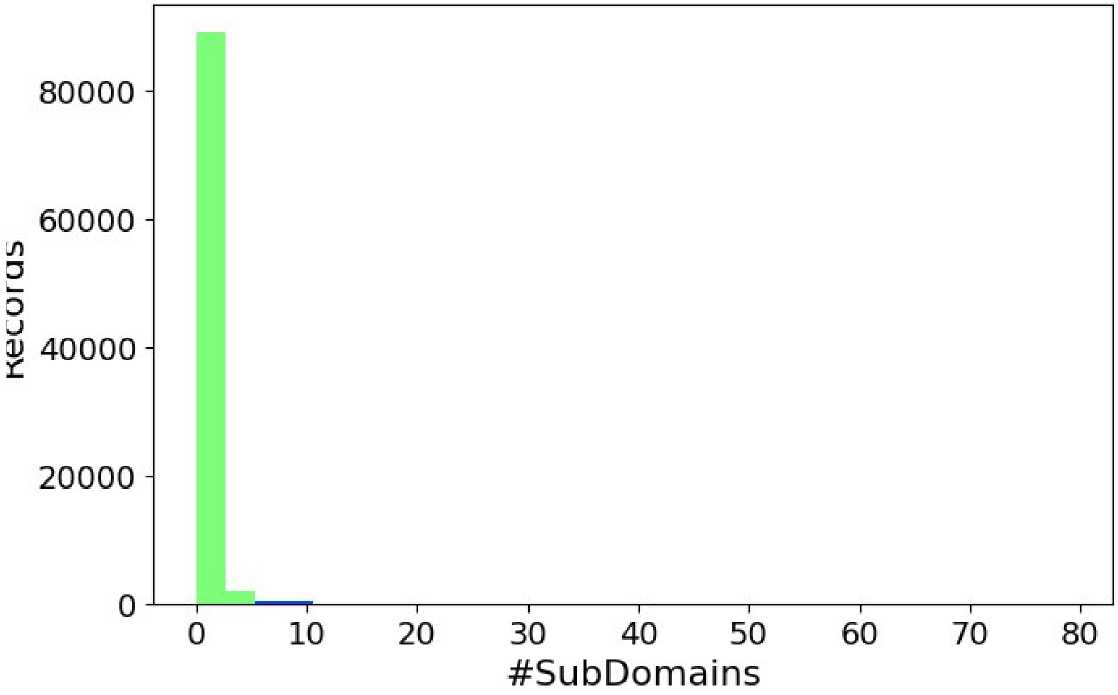
Some stats about gathered data in here.

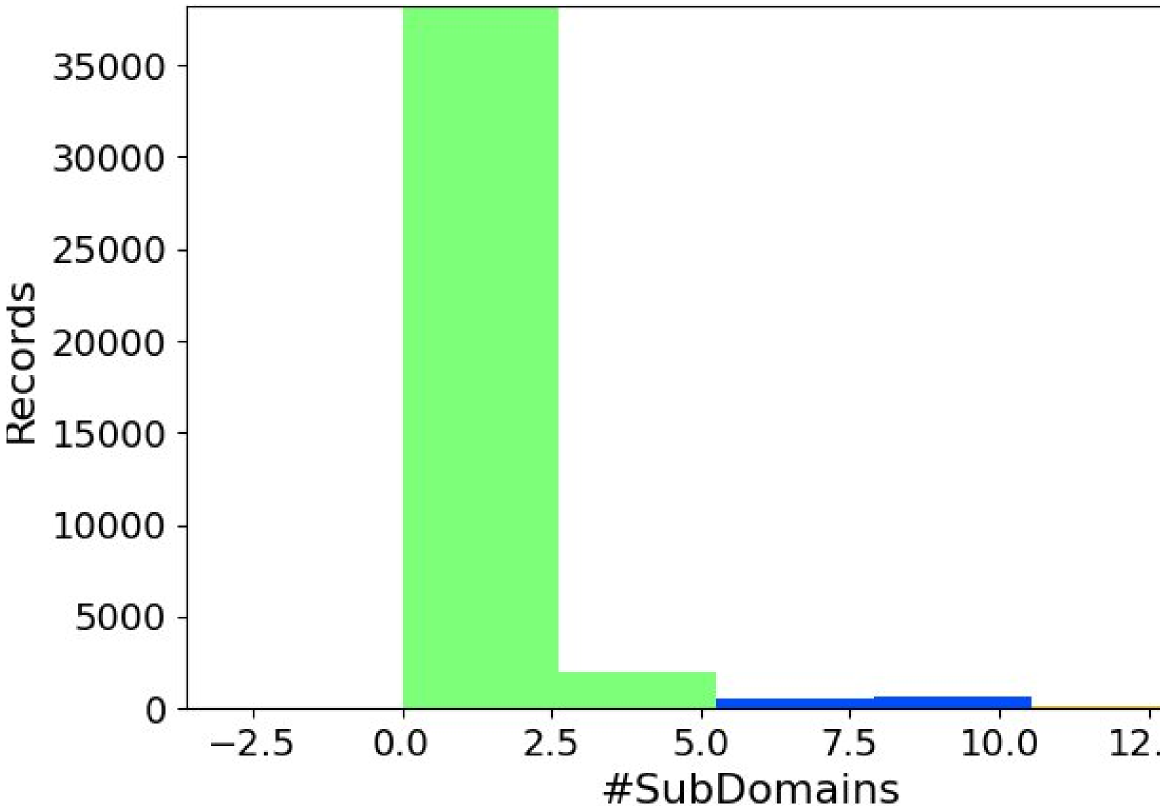
US is the country with the most Malicious and Benign domain. This is probably because the gathered data.

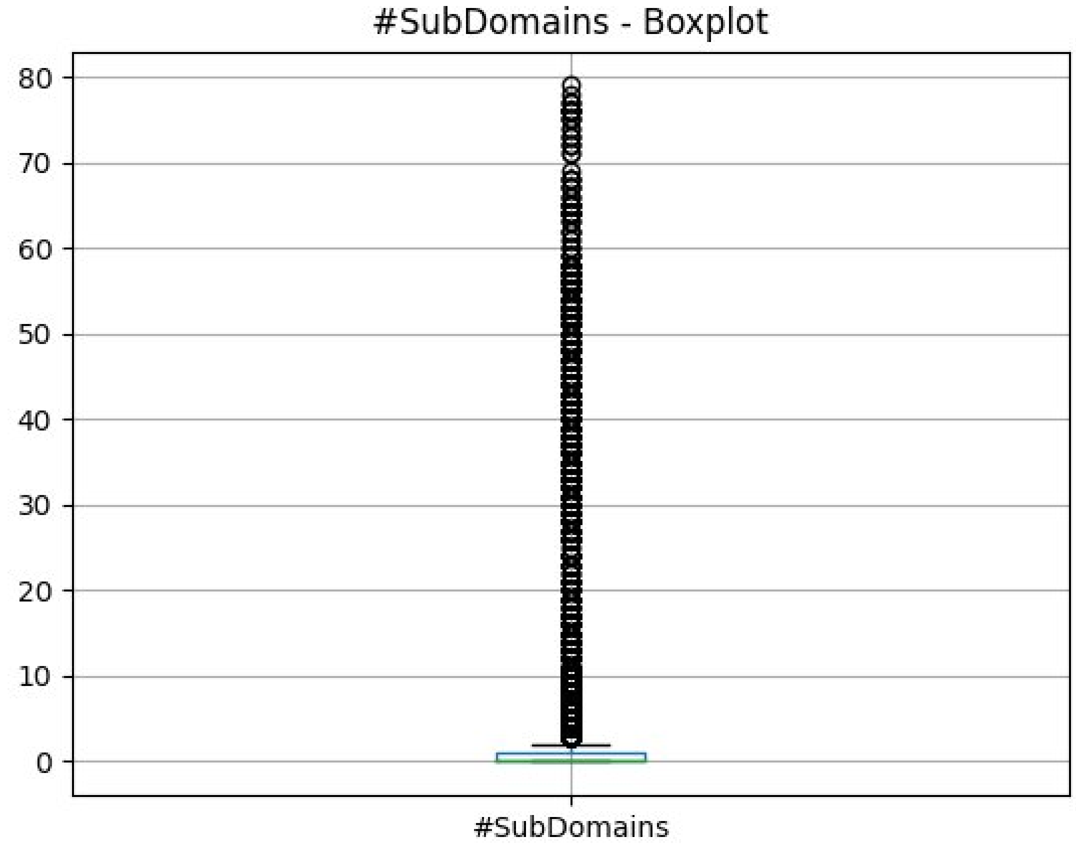
The Organisation with most benign sites is CloudFlareNet; HURRICANE is the organisation that hosts the most number of malicious domain.

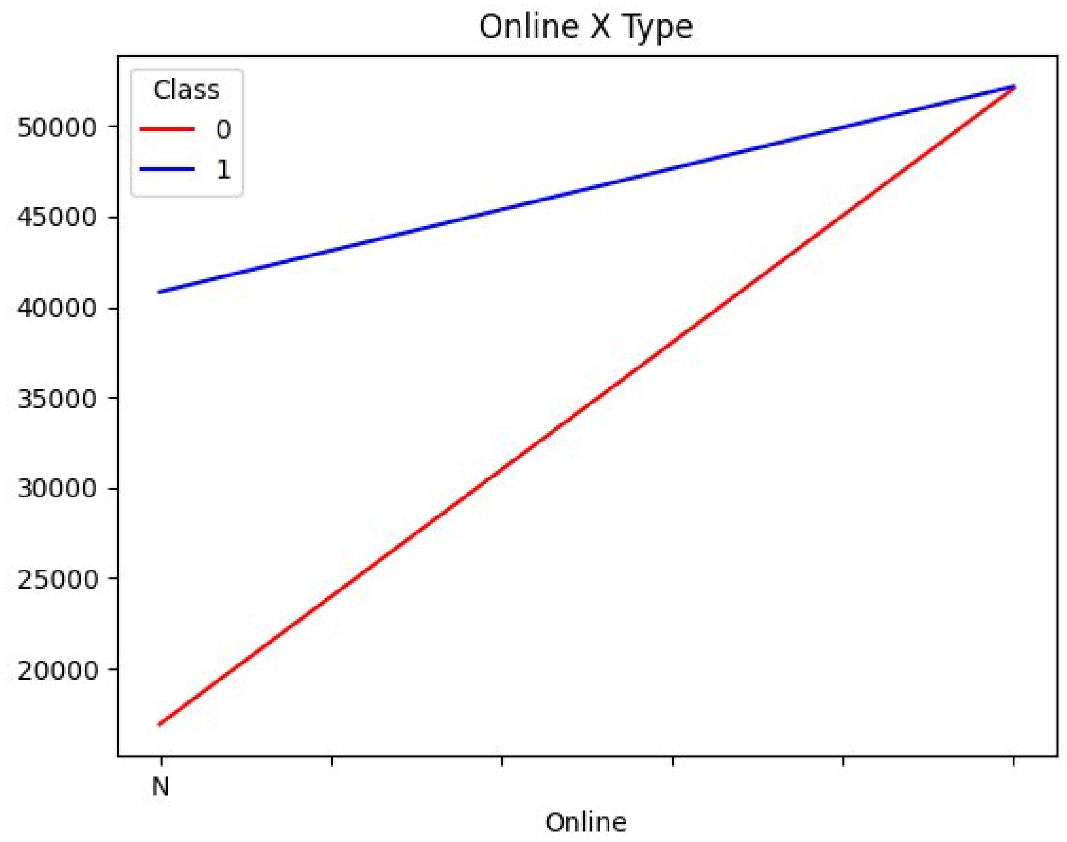
TLD for both is .com.

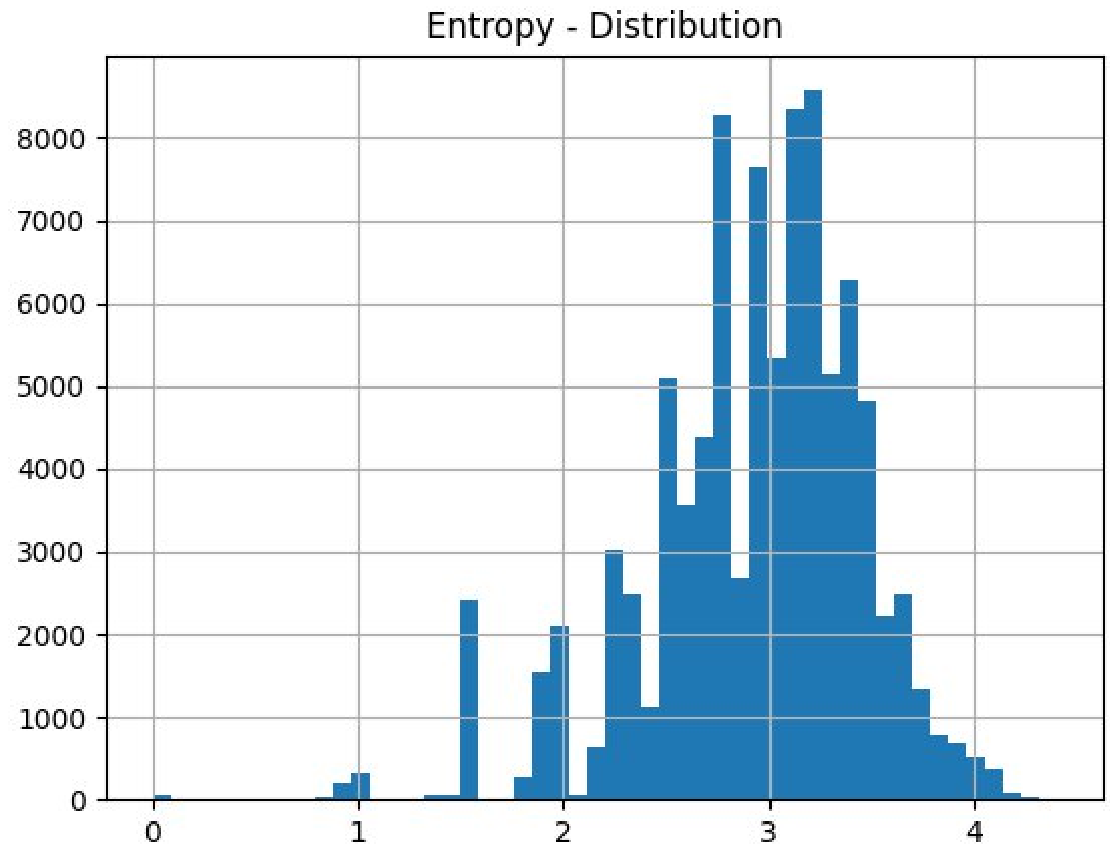
Distributions:

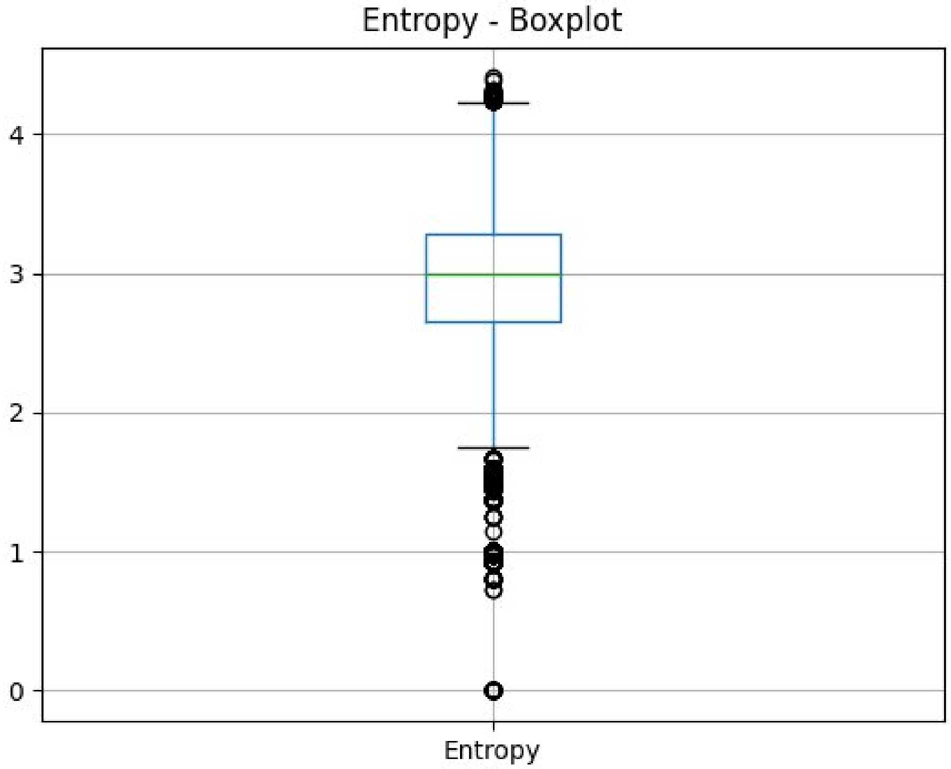


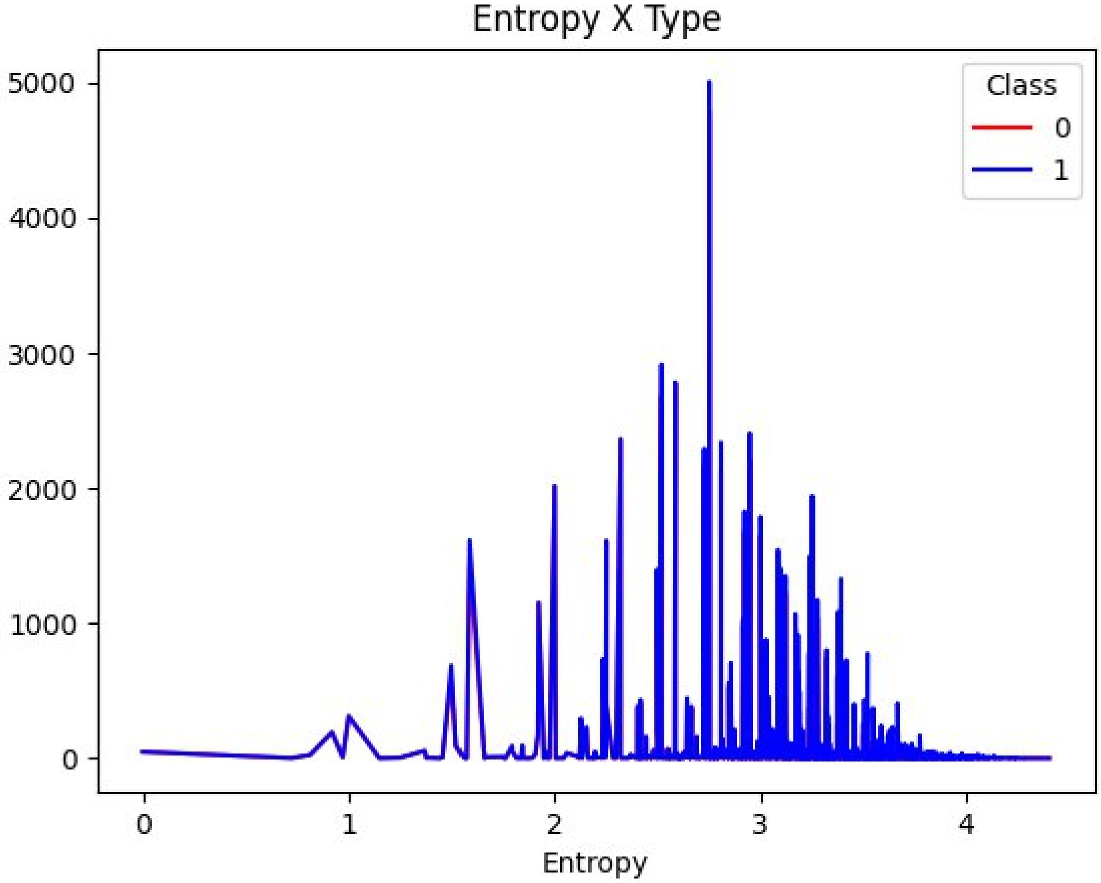

Records
35000
30000
25000
20000
15000
10000
5000
0
−2.5
0.0
2.5
5.0
7.5
10.0
12.
#SubDomains

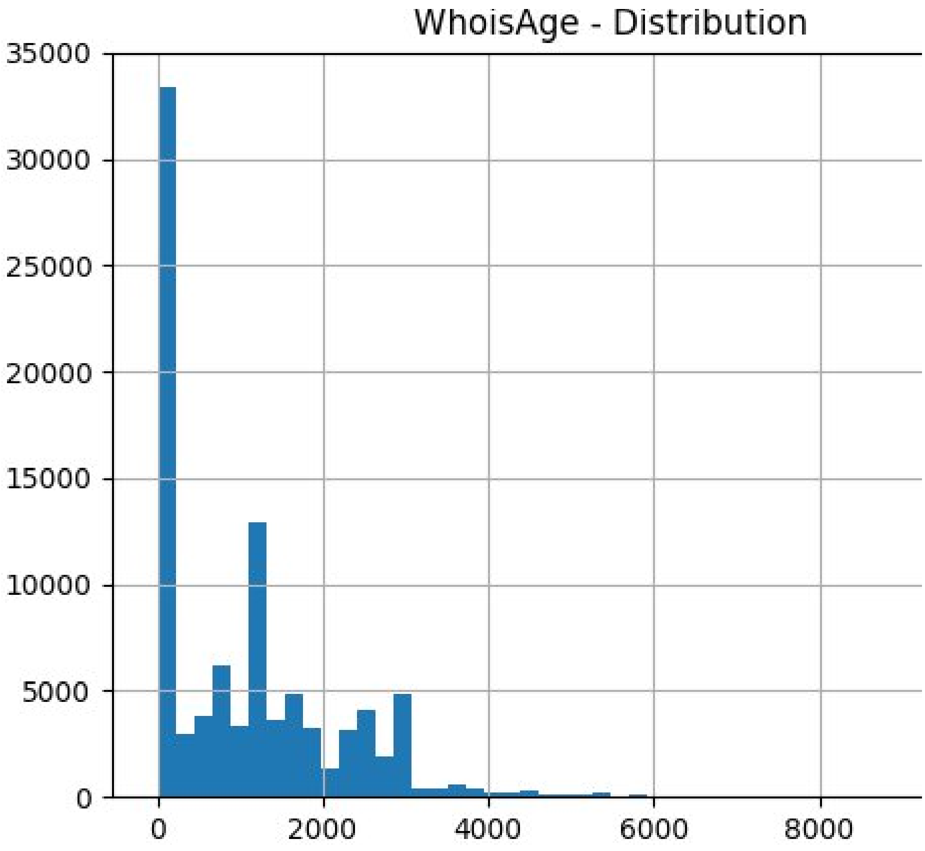

#SubDomains - Boxplot
80
70
60
50
40
30
20
10
0
#SubDomains

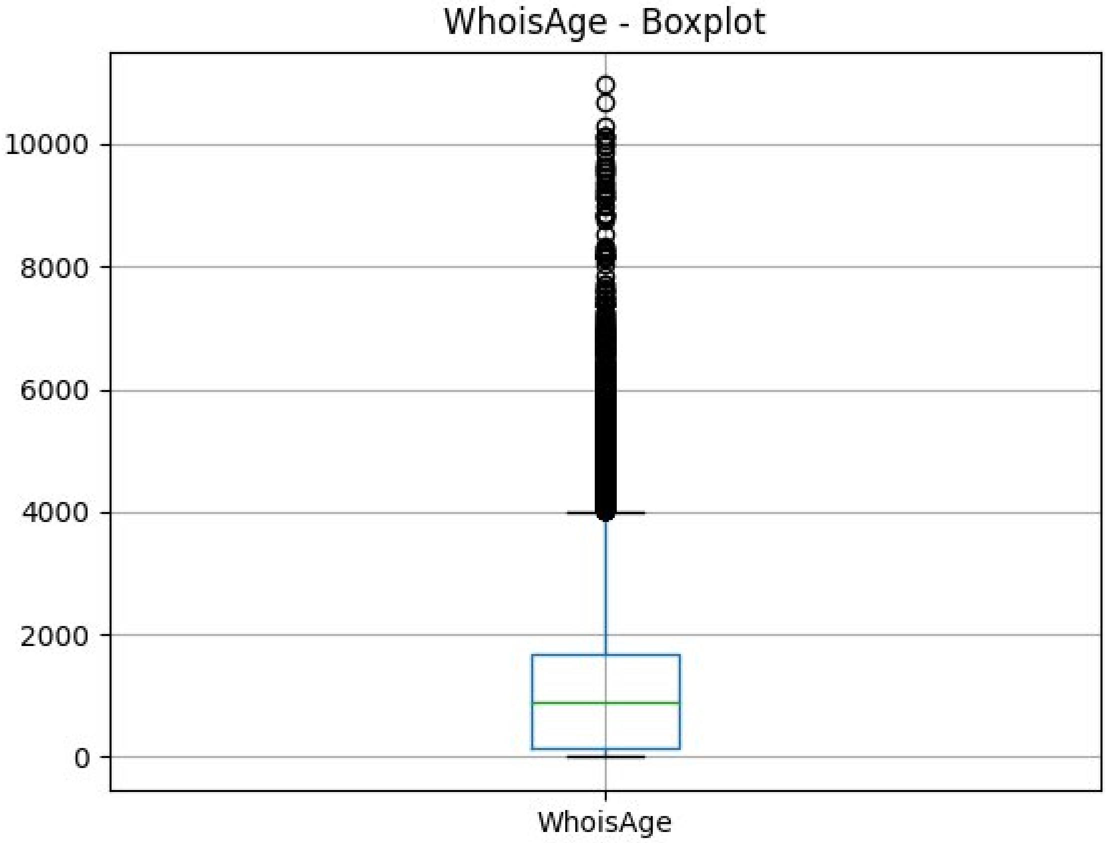

Online X Type
Class
0
1
50000
45000
40000
35000
30000
25000
20000
N
Online


Entropy - Distribution
8000
7000
6000
5000
4000
3000
2000
1000
0
0
1
2
3
4


Entropy - Boxplot
4
3
2
1
0
Entropy


Entropy X Type
5000
4000
3000
2000
1000
0
0
1
2
3
4
Entropy
Class
0
1


WhoisAge - Distribution
35000
30000
25000
20000
15000
10000
5000
0
0
2000
4000
6000
8000


WhoisAge - Boxplot
10000
8000
6000
4000
2000
0
WhoisAge

Machine Learning: Malicious Domain Prediction for DS1 & DS2

In the Machine Learning I wanted to train an AI to predict malicious and benign domains part and see the different Accuracy results. I splitted both datasets in test and training: the Ratio proposed in the assignment for Dataset 1 is 60% for training and 40% for test(about 175.000 for training and 125.000 for test); for Dataset 2 is 75% for training and 25% for test.

Two things to report:

Than I passed to apply a Logarithmic transformation :

1. I do the Logarithmic transformation for :
   a. ‘WhoisAge’: I transform only this value because is the column with highest values.
2. Fit the categorical data:

```
['Domain','Online','IP','Country','City','TLD','Organisation','ASN']
```

I used these models of prediction with their proper settings :

- Decision Tree Classifier
- Logistic Regression
- Linear SVC
- Gaussian Naive Bayes
- Random Forest Classifier (in this case I used the best 5 variables to avoid the overfitting problem)

In the case of Dataset2, because I have a lot of generated domain name I used less variables for prediction. Is it possible to see everything in the source code I paste here in report with proper comments. The two scripts below include also the part for analysis description.

dataAnalysisP1.py

```
import pandas as pd
import numpy as np
import matplotlib as plt
from matplotlib import pyplot as plt
import seaborn as sns
import re
import sklearn
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier, export_graphviz
#from sklearn.svm import SVC -- This is a problem to use with big
datasets
from sklearn.svm import LinearSVC
from sklearn.model_selection import KFold
from sklearn.naive_bayes import GaussianNB
from sklearn import metrics
from time import time

df1 = pd.read_csv("DS1/finalds1.csv")

print(df1.head(10))
print(df1.describe())

'''
#theCountry where the most number of malicious and benign are from
created Datasets
print
(df1[df1['Class']==0].groupby(['Country'])['Class'].value_counts().idxmax
())
print
(df1[df1['Class']==1].groupby(['Country'])['Class'].value_counts().idxmax
())
#theOrganisation where the most number of malicious and benign are from
created Datasets
#ASN not needed because it's representing the textual organisation
print
(df1[df1['Class']==0].groupby(['Organisation'])['Class'].value_counts().i
dxmax())
print
(df1[df1['Class']==1].groupby(['Organisation'])['Class'].value_counts().i
dxmax())
#most frequent TLD for corrispective classes
print
(df1[df1['Class']==0].groupby(['TLD'])['Class'].value_counts().idxmax())
print
(df1[df1['Class']==1].groupby(['TLD'])['Class'].value_counts().idxmax())
'''
```

```
missingList = df1.columns[df1.isna().any()].tolist()
for missing in missingList:
    if (missing == "Entropy"):
        df1['Entropy'] = df1['Entropy'].replace('?', 
np.nan).astype(float)
        df1['Entropy'] = df1['Entropy'].fillna(df1['Entropy'].max())
    elif (missing == 'IP'):
        df1['IP'].fillna('None', inplace = True)
    elif (missing == 'Country'):
        df1['Country'].fillna('None', inplace = True)
    elif (missing == 'City'):
        df1['City'].fillna('None', inplace = True)
    elif (missing == 'WhoisAge'):
        df1['WhoisAge'] = 
df1['WhoisAge'].fillna(df1['WhoisAge'].quantile(0.10))
    elif (missing == 'TLD'):
        df1['TLD'].fillna('None', inplace = True)
    elif (missing == 'ASN'):
        df1['ASN'].fillna('None', inplace = True)
        df1['ASN'] = df1['ASN'].astype(str)
    elif (missing == 'Organisation'):
        df1['Organisation'].fillna('None', inplace = True)
    elif (missing == '#SubDomains'):
        df1['#SubDomains'].fillna(0, inplace = True)

#print(df1.describe())
#print(df1.head(10))

#see how data is distributed
#every type of histogram and boxplot would be saved as png. If
#don't want to save remove all lines with plt.savefig().

#split columns in two list: numerical values and categorical value
#just for analysis purpose
numerical = ['Entropy','WhoisAge','#SubDomains']
categorical = ['Online','Country','TLD','Organisation']
df1_plot = df1[numerical + categorical]

'''
#histogram to see numbers of subDomains with colors
N, bins, patches = plt.hist(df1_plot['#SubDomains'], 30)
cmap = plt.get_cmap('jet')
low = cmap(0.5)
medium =cmap(0.2)
high = cmap(0.7)
for i in range(0,2):
    patches[i].set_facecolor(low)
for i in range(2,4):
    patches[i].set_facecolor(medium)
for i in range(4,20):
    patches[i].set_facecolor(high)
```

```
plt.xlabel("#SubDomains", fontsize=16)
plt.ylabel("Records", fontsize=16)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)

plt.show()


df1_plot[numerical].hist(bins=10, figsize=(15, 6), layout=(2, 2));
plt.show()

for column in categorical:
    ct = pd.crosstab(df1[column], df1['Class'])
    ct.plot(kind='line', stacked=True, color=['red','blue'], grid=False)
    plt.title(column + " X Type")
    #plt.savefig('img/' + column + 'Crosstab.png')
    plt.show()

for column in numerical:
    df1[column].hist(bins = 50)
    plt.title(column + " - Distribution")
    #plt.savefig('img/' + column + 'Histogram.png')
    plt.show(block = True)
    df1.boxplot(column = column)
    plt.title(column + " - Boxplot")
    #plt.savefig('img/' + column + 'Boxplot.png')
    plt.show(block=True)
    plt.interactive(False)
    ct = pd.crosstab(df1[column], df1['Class'])
    ct.plot(kind='line', stacked=True, color=['red','blue'], grid=False)
    plt.title(column + " X Type")
    #plt.savefig('img/' + column + 'Crosstab.png')
    plt.show()
'''

#Transform Var WhoisAge applying logarithmic transformation

df1['WhoisAge'] = np.log(df1['WhoisAge']+1)#+1 to avoid 0 values.

#fit categorical data
varListFit =
['Domain','Online','IP','Country','City','TLD','Organisation','ASN']
le = LabelEncoder()
for i in varListFit:
    df1[i] = le.fit_transform(df1[i])

#split Datasets for training and test


train_set1 = df1.sample(frac=0.6, random_state=0)
test_set1 = df1.drop(train_set1.index)
```

```
#now passing to use prediction models giving the needed variables
#except for random forest predict model that will take every column


#in the classification model we prepare a table that contains all the
#data about various classification models
# initialize list of lists
resTable = []
name_model_for_ftable = ''
# Create the pandas DataFrame

#process of classification with ML
def classification_model(model, data, data_train, data_test, predictors,
outcome):
    #fit the model
    model.fit(data_train[predictors], data_train[outcome])
    model.fit(data[predictors], data[outcome])

    t1=time()
    #Make predictions on training set:
    predictions = model.predict(data_test[predictors])

    #print accuracy and duration

    accuracy = metrics.accuracy_score(predictions, data_test[outcome])
    # Perform k-fold cross-validation with 5 folds
    kf=KFold(n_splits=2)
    pred=[]
    for train, test in kf.split(data):
        # Filter training data
        train_predictors =(data_train[predictors])
        # The target we're using to train the algorithm.
        train_target = data_train[outcome]
        # Training the algorithm using the predictors and target.
        model.fit(train_predictors, train_target)
        # Record error from each cross-validation run
        pred.append(model.score(data_test[predictors],
data_test[outcome]))
    f_report = metrics.f1_score(data_test[outcome], predictions, average
= 'macro')
    dur = round(time()-t1, 3)
    tempA = "%s" % "{0:.3%}".format(accuracy)
    tempB = str(dur) + 's'
    print("Accuracy : %s" % "{0:.3%}".format(accuracy) + ' || ' + "Cross-
Validation Score : %s" % "{0:.3%}".format(np.mean(pred)) + ' || ' + "F-
Score: " + str(f_report) + ' || ' + "Duration: " + tempB)
    #fit the model again so that it can be referred outside the function:
    model.fit(data_train[predictors], data_train[outcome])
    model.fit(data[predictors], data[outcome])
```

```
#now we use various model of prediction to see performances and results
#each model has is own setup after several tests to get the better
values.
model = [DecisionTreeClassifier(max_depth=15),
LogisticRegression(max_iter=105),#max iteration: default is 100, with
minimum 105 value I avoid the reach limit.

LinearSVC(dual=False),GaussianNB(),RandomForestClassifier(n_estimators=25
, min_samples_split=25, max_depth=9, max_features=1)]

predictor_var =
['Online','WhoisAge','#SubDomains','Organisation','Entropy','TLD']

'''
#Checking the best variables for random forest to avoid overfitting
predictor_varRF =
['Entropy','WhoisAge','Organisation','Online','Country','TLD','City','IP'
,'#SubDomains']
model1 = RandomForestClassifier()
outcome = 'Class'
model1.fit(train_set1[predictor_varRF], train_set1[outcome])
featimp = pd.Series(model1.feature_importances_,
index=predictor_varRF).sort_values(ascending=False)
print (featimp)
'''

for i in model:
    outcome_var = 'Class'
    name_model_for_ftable = type(i).__name__
    if (name_model_for_ftable == 'RandomForestClassifier'):
        predictor_var =
['Entropy','WhoisAge','Organisation','IP','TLD']#these are the best five
for RandomForestClassifier
    classification_model(i, df1, train_set1, test_set1,  predictor_var,
outcome_var)
```

dataAnalysisP2.py

```
import pandas as pd
import numpy as np
import matplotlib as plt
from matplotlib import pyplot as plt
import seaborn as sns
import re
import sklearn
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier, export_graphviz
#from sklearn.svm import SVC -- This is a problem to use with big
datasets
from sklearn.svm import LinearSVC
from sklearn.model_selection import KFold
from sklearn.naive_bayes import GaussianNB
from sklearn import metrics
from time import time

df2 = pd.read_csv("DS2/finalds2.csv")

#bit of Stats about categorical values
print(df2.head(10))
print(df2.describe())
#theCountry where the most number of malicious and benign are from
created Datasets
print
(df2[df2['Class']==0].groupby(['Country'])['Class'].value_counts().idxmax
())
print
(df2[df2['Class']==1].groupby(['Country'])['Class'].value_counts().idxmax
())
#theOrganisation where the most number of malicious and benign are from
created Datasets
#ASN not needed because it's representing the textual organisation
print
(df2[df2['Class']==0].groupby(['Organisation'])['Class'].value_counts().i
dxmax())
print
(df2[df2['Class']==1].groupby(['Organisation'])['Class'].value_counts().i
dxmax())
#most frequent TLD for corrispective classes
print
(df2[df2['Class']==0].groupby(['TLD'])['Class'].value_counts().idxmax())
print
(df2[df2['Class']==1].groupby(['TLD'])['Class'].value_counts().idxmax())

print(df2.head(10))
print(df2.describe())
```

```
missingList = df2.columns[df2.isna().any()].tolist()
for missing in missingList:
    if (missing == "Entropy"):
        df2['Entropy'] = df2['Entropy'].replace('?', 
np.nan).astype(float)
        df2['Entropy'] = df2['Entropy'].fillna(df2['Entropy'].max())
    elif (missing == 'IP'):
        df2['IP'].fillna('None', inplace = True)
    elif (missing == 'Country'):
        df2['Country'].fillna('None', inplace = True)
    elif (missing == 'City'):
        df2['City'].fillna('None', inplace = True)
    elif (missing == 'WhoisAge'):
        df2['WhoisAge'] = 
df2['WhoisAge'].fillna(df2['WhoisAge'].quantile(0.05))
    elif (missing == 'TLD'):
        df2['TLD'].fillna('None', inplace = True)
    elif (missing == 'ASN'):
        df2['ASN'].fillna('None', inplace = True)
        df2['ASN'] = df2['ASN'].astype(str)
    elif (missing == 'Organisation'):
        df2['Organisation'].fillna('None', inplace = True)
    elif (missing == '#SubDomains'):
        df2['#SubDomains'].fillna(0, inplace = True)


#see how data is distributed
#every type of histogram and boxplot would be saved as png. If
#don't want to save remove all lines with plt.savefig().

#split columns in two list: numerical values and categorical value
numerical = ['Entropy','WhoisAge','#SubDomains']
categorical = ['Online','Country','TLD','Organisation']
df2_plot = df2[numerical + categorical]

#histogram to see numbers of subDomains with colors
N, bins, patches = plt.hist(df2_plot['#SubDomains'], 30)
cmap = plt.get_cmap('jet')
low = cmap(0.5)
medium =cmap(0.2)
high = cmap(0.7)
for i in range(0,2):
    patches[i].set_facecolor(low)
for i in range(2,4):
    patches[i].set_facecolor(medium)
for i in range(4,20):
    patches[i].set_facecolor(high)
plt.xlabel("#SubDomains", fontsize=16)
plt.ylabel("Records", fontsize=16)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)

plt.show()


df2_plot[numerical].hist(bins=10, figsize=(15, 6), layout=(2, 2));
plt.show()

for column in categorical:
    ct = pd.crosstab(df2[column], df2['Class'])
    ct.plot(kind='line', stacked=True, color=['red','blue'], grid=False)
    plt.title(column + " X Type")
    #plt.savefig('img/' + column + 'Crosstab.png')
    plt.show()

for column in numerical:
    df2[column].hist(bins = 50)
    plt.title(column + " - Distribution")
    #plt.savefig('img/' + column + 'Histogram.png')
    plt.show(block = True)
    df2.boxplot(column = column)
    plt.title(column + " - Boxplot")
    #plt.savefig('img/' + column + 'Boxplot.png')
    plt.show(block=True)
    plt.interactive(False)
    ct = pd.crosstab(df2[column], df2['Class'])
    ct.plot(kind='line', stacked=True, color=['red','blue'], grid=False)
    plt.title(column + " X Type")
    #plt.savefig('img/' + column + 'Crosstab.png')
    plt.show()

#Transform Var WhoisAge applying logarithmic transformation

df2['WhoisAge'] = np.log(df2['WhoisAge']+1)#+1 to avoid 0 values.

#fit categorical data
varListFit =
['Domain','Online','IP','Country','City','TLD','Organisation']
le = LabelEncoder()
for i in varListFit:
    df2[i] = le.fit_transform(df2[i])

#split Datasets for training and test

train_set2 = df2.sample(frac=0.75, random_state=0)
test_set2 = df2.drop(train_set2.index)

#now passing to use prediction models giving the needed variables
#except for random forest predict model that will take every column
```

```
#in the classification model we prepare a table that contains all the
#data about various classification models
# initialize list of lists
resTable = []
name_model_for_ftable = ''
# Create the pandas DataFrame

#process of classification with ML
def classification_model(model,data, data_train, data_test, predictors,
outcome):
    #fit the model
    model.fit(data_train[predictors], data_train[outcome])
    model.fit(data[predictors], data[outcome])

    t1=time()
    #Make predictions on training set:
    predictions = model.predict(data_test[predictors])

    #print accuracy and duration

    accuracy = metrics.accuracy_score(predictions, data_test[outcome])
    # Perform k-fold cross-validation with 5 folds
    kf=KFold(n_splits=4)
    pred=[]
    for train, test in kf.split(data):
        # Filter training data
        train_predictors =(data_train[predictors])
        # The target we're using to train the algorithm.
        train_target = data_train[outcome]
        # Training the algorithm using the predictors and target.
        model.fit(train_predictors, train_target)
        # Record error from each cross-validation run
        pred.append(model.score(data_test[predictors],
data_test[outcome]))
    f_report = metrics.f1_score(data_test[outcome], predictions, average
= 'macro')
    dur = round(time()-t1, 3)
    tempA = "%s" % "{0:.3%}".format(accuracy)
    tempB = str(dur) + 's'
    print("Accuracy : %s" % "{0:.3%}".format(accuracy) + ' || ' + "Cross-
Validation Score : %s" % "{0:.3%}".format(np.mean(pred)) + ' || ' + "F-
Score: " + str(f_report) + ' || ' + "Duration: " + tempB)
    #fit the model again so that it can be referred outside the function:
    model.fit(data_train[predictors], data_train[outcome])
    model.fit(data[predictors], data[outcome])

#now we use various model of prediction to see performances and results
#each model has is own setup after several tests to get the better
values.
model = [DecisionTreeClassifier(max_depth=15),
LogisticRegression(max_iter=1000),
LinearSVC(dual=False),GaussianNB(),RandomForestClassifier(n_estimators=25
, min_samples_split=25, max_depth=9, max_features=1)]

predictor_var = ['WhoisAge','Entropy','TLD','Online','#SubDomains']

'''
#Checking the best variables for random forest to avoid overfitting
predictor_varRF =
['Entropy','WhoisAge','Organisation','Online','Country','TLD','City','IP'
,'#SubDomains']
model1 = RandomForestClassifier()
outcome = 'Class'
model1.fit(train_set2[predictor_varRF], train_set2[outcome])
featimp = pd.Series(model1.feature_importances_,
index=predictor_varRF).sort_values(ascending=False)
print (featimp)
'''

for i in model:
    outcome_var = 'Class'
    name_model_for_ftable = type(i).__name__
    if (name_model_for_ftable == 'RandomForestClassifier'):
        predictor_var =
['Entropy','WhoisAge','Organisation','Online','IP']
    classification_model(i, df2, train_set2, test_set2, predictor_var,
outcome_var)

'''
predictor_varRF = ['Entropy','WhoisAge','Organisation','Domain','TLD']
model1 = RandomForestClassifier()
outcome = 'Class'
model1.fit(train_set1[predictor_varRF], train_set1[outcome])
featimp = pd.Series(model1.feature_importances_,
index=predictor_varRF).sort_values(ascending=False)
print (featimp)

'''
```

Now the Table of results:

Dataset	Algorithm	Accuracy	Cross Validation	F-Score	Time (seconds)
1	**Decision Tree**	83.99%	81.88%	83.25%	0.886
	Logistic Regression	68.29%	68.35%	65.47%	7.41
	LinearSVC	68.41%	68.39%	65.58%	1.401
	Gaussian Naive Bayes	69.46%	69.43%	68.69%	0.18
	Random Forest	79.37%	79.23%	79.04%	5.108
2	**Decision Tree**	99.11%	98.84%	98.85%	0.372
	Logistic Regression	96.85%	96.86%	95.99%	6.615
	LinearSVC	96.73%	96.72%	95.85%	0.965
	Gaussian Naive Bayes	96.49%	96.56%	95.59%	0.082
	Random Forest	98.59%	98.58%	98.2%	2.46

Now some interpretation about results. As we can see we have big differences in all results between the datasets: probably the different distribution(massive number of generated domain) and the less number of records in DS2 help the AI to predict better the difference about malicious and benign domain(extracted from URL). In DS1 we have 3x more records instead of DS2 and more “heterogeneous” data in Domain column.

We use 3 different type of results:

- Accuracy: it is the measure of all the correctly identified cases. It is most used when all the classes are equally important. It gives result with this Formula:

$$\text{Accuracy} = \frac{\text{True Positive} + \text{True Negative}}{(\text{True Positive} + \text{False Positive} + \text{True Negative} + \text{False Negative})}$$

- Cross-Validation: is basically a resampling technique to make our model sure about its efficiency and accuracy on the unseen data. We use the k value to split the dataset in parts. Each iteration it use one partition as test and the others as training. Next iterations switch simply the test partition. In the end what we have is an avarage of different Accuracy results. In the end we can say that is more accurated Accuracy result that try to avoid the overfitting and underfitting of data.

- F-score: it is the harmonic mean (that in the end penalizes the extreme values) of:
  1. Precision: implied as the measure of the correctly identified positive cases from all the predicted positive cases. Useful when the costs of False Positives is High.

$$\text{Precision} = \frac{\text{True Positive}}{(\text{True Positive} + \text{False Positive})}$$

  2. Recall: measure of the correctly identified positive cases from all the actual positive cases. Important when cost of False Negatives is high:

$$\text{Recall} = \frac{\text{True Positive}}{(\text{True Positive} + \text{False Negative})}$$

In the end, the final formula:

$$\text{F1-score} = \left(\frac{\text{Recall}^{-1} + \text{Precision}^{-1}}{2}\right)^{-1} = 2 * \frac{(\text{Precision} * \text{Recall})}{(\text{Precision} + \text{Recall})}$$

Probably the F-Score is the most trustable result to follow in every module but it really depends on the application: I personally would follow this because I imagine that in a real work scenario, I must defend my network security putting immediatly suspicious traffic of data in a black list and then check this data in a sandbox to do analysis. In Maths terms is more strict than the other two results.

Anyway in the end, from my table we can see that there isn’t too much remarkable differences between results.

</part4>

Conclusion

This was a work done by me knowing almost nothing about the subject and it’s own instruments(except for Python). It was interesting to Build everything from scratch and understanding an environment that almost simulate a real world application in this scenario, gathering data, creating datasets, analizing dataset when all filled and then creating an AI for predict the malicious data. I think this is a good point to start to think how to work in a Cyber Intelligence team with cooperation of other types of team that are complained in Cyber Security.