“ Identification/Detection Of Malicious Activities Through Internet ’’

BACHELOR OF ENGINEERING
IN
COMPUTER SCIENCE & ENGINEERING

GALGOTIAS UNIVERSITY
(Established under Galgotias University Uttar Pradesh Act No. 14 of 2011)

Under the Supervision of :-
“Mr. Himanshu Sharma Sir”
Assistant Professor

Submitted by:-

S.No	Enrollment Number	Admission Number	Student Name	Degree /Branch	Sem
1	19021260008	19SCSE1260012	KRISHNA	B.TECH IN CSE WITH SPECIALIZATION IN CSDF	v
2.	19021260002	19SCSE1260006	ADARSH KUMAR SINGH	BTECH IN CSE WITH SPECIALIZATION IN CSDF	v

SCHOOL OF COMPUTING SCIENCE & ENGINEERING,

Galgotias University, Greater Noida,

Fall 2021-22

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING
GALGOTIAS UNIVERSITY, GREATER NOIDA

CANDIDATE'S DECLARATION

We hereby certify that the work which is being presented in the thesis/project/dissertation, entitled **"Identification/Detection Of Malicious Activities Through Internet"** in partial fulfillment of the requirements for the award of the project review in the School of Computing Science and Engineering of Galgotias University, Greater Noida, is an original work carried out during the period of Oct, 21 to Dec, 21 under the **Mr. Himanshu Sharma**, Department of Computer Science and Engineering/Computer Application and Information and Science, of School of Computing Science and Engineering , Galgotias University, Greater Noida

The matter presented in the project has not been submitted by us for the award of any other degree of this or any other places.

Adarsh Kumar Singh, 19SCSE1260006

Krishna, 19SCSE1260012

This is to certify that the above statement made by the candidates is correct to the best of my knowledge.

Mr. Himanshu Sharma
(Assistant Professor)

CERTIFICATE

The Final Thesis/Project/ Dissertation Viva-Voce examination of Adarsh Kumar Singh (19SCSE126006) & Krishna (19SCSE1260006) has been held on _________________ and his/her work is recommended for the award of Project Review.

Signature of Examiner(s) **Signature of Supervisor(s)**

Signature of Project Coordinator **Signature of Dean**

Date: December, 2021
Place: Greater Noida

Acknowledgement

In performing our assignment, we had to take the help and guidance of some respected person, who deserves our greatest gratitude. The completion of this assignment gives us much Pleasure. We would like to show our gratitude to **Mr. Himanshu Sharma Sir**, Project Guide, Galgotias University who introduced us to the Methodology of work, and whose passion for the "underlying structures" had a lasting effect and for giving us a good guideline for assignment throughout numerous consultations. We would also like to expand our deepest gratitude to all those who have directly and indirectly guided us in writing this assignment.

Many people, especially our classmates and team members, have made valuable comment suggestions on this proposal which gave us inspiration to improve our assignment. We thank all the people for their help directly and indirectly to complete our assignment.

Table Of Content

Abstract

Data and application security is most essential in today's environment due to the advancement as well as exchange of information and communication techniques that generate new value added services by different network threats. As a result, they developed diverse online services. However, cyber security threats are also growing as the contact points to the Internet are increasing. The world wide web is more vulnerable for malicious activities. Spam–advertisements, Sybil attacks, Rumour propagation, financial frauds, malware dissemination and Sql injection are some of the malicious activities on the web. It is very difficult to trace the impression of malicious activities on the web. Many researches are under development to find a mechanism to protect web users and avoid malicious activities. The aim of the survey is to provide a study on recent techniques to find malicious activities on the web. Detection of malicious activities and identification of threat types are critical to thwart these attacks. Knowing the type of a threat enables estimation of severity of the attack and helps adopt an effective countermeasure. Existing methods typically detect malicious activities of a single attack type. In this paper, we propose a method using machine learning to detect malicious activities of all the popular attack types and identify the nature of attack and malicious activities attempts to launch. Here, we use Machine learning, which has a variety of discriminative features including textual properties, link structures, webpage contents, DNS information, and network traffic. Many of these features are novel and highly effective. Recently, Machine learning (ML) is a widespread technique offered to feed the Intrusion Detection System (IDS) to detect malicious network activities. The core of ML models' detection efficiency relies on the dataset's quality to train the model. This research proposes a detection framework with an ML model for feeding IDS to detect network traffic anomalies. This detection model uses a dataset constructed from malicious and normal traffic. This research's significant challenges are the extracted features used to train the ML model about various attacks to distinguish whether it is an anomaly or regular traffic.

Introduction

Malware or malicious code is harmful code injected into legitimate programs to perpetrate illicit intentions. With the rapid growth of the Internet and heterogeneous devices connected over the network, the attack landscape has increased and has become a concern, affecting the privacy of users. The primary source of infection, causing malicious programs to enter the systems without users' knowledge. Mostly freely downloadable software's are a primary source of malware, which include freeware consisting of games, web browsers, free antivirus, etc. Largely financial transactions are performed using the Internet, these have caused huge financial losses for organizations and individuals. Malware writing has transformed into profit-making industries, thus attracting a large number of hackers. Current malware is broadly classified as polymorphic or metamorphic, and it remains undetected by a signature-based detector.

Malware writers employ diverse techniques to generate new variants that commonly include (a) instruction permutation, (b) register re-assignment, (c) code permutation using conditional instructions, (d) no-operation insertion, etc. Malware analysis is the process aimed to inspect and understand malicious behavior. Normally malware are analyzed by extracting strings, opcodes, sequence of bytes, APIs/system call, and the network trace.

While the World Wide Web has become a killer application on the Internet, it has also brought in an immense risk of cyber attacks. Adversaries have used the Web as a vehicle to deliver malicious attacks such as phishing, banking frauds, spamming and malware infection. For example, phishing typically involves sending an email seemingly from a trustworthy source to trick people to click a URL (Uniform Resource Locator) contained in the email that links to a counterfeit webpage.

To address Web-based attacks, a great effort has been directed towards detection of malicious Activities. A common countermeasure is to use a blacklist of certain malicious activities, which can be constructed from various sources, particularly human feedback that are highly accurate yet time-consuming. Blacklisting incurs no false positives, yet is effective only for known malicious Activities. It cannot detect unknown malicious activities. The very nature of exact matches in blacklisting renders it easy to be evaded.

This weakness of blacklisting has been addressed by anomaly-based detection methods designed to detect unknown malicious activities. In these methods, a classification model based on discriminative rules or features is built with either knowledge a priori or through machine learning. Selection of discriminative rules or features plays a critical role for the performance of a detector. Existing methods were designed to detect malicious activities or URLs of a single attack type, such as spamming, phishing, or malware.

Identification of attack types is useful since the knowledge of the nature of a potential threat allows us to take a proper reaction as well as a pertinent and effective countermeasure against the threat. One experimental study reports that the Sasser worm located a PC running a vulnerable operating system and successfully compromised the machine in less than four minutes from when the machine was connected to the Internet.

The speed and prevalence of automated attacks render ineffective any legacy defenses that rely on the manual inspection of each case. It is necessary to deploy an automated defense system that continuously monitors network traffic, determines whether the traffic from a particular source reveals a certain malicious activity, and then triggers an alarm when it finds such traffic. In this dissertation, we investigate the problem of designing efficient detection mechanisms for malicious network activity that are suitable for practical deployment in a real-time automated network defense system.

Recently, ML techniques were used to train Intrusion detection system (IDS) to capture malicious network traffic. The main idea of IDS based on ML analysis is finding patterns and building an IDS based on the dataset. The IDS can detect adequately. We need to have a real network traffic dataset and proper feature selection to learn enough. Therefore, we aim to propose a detection framework with an ML model to detect malicious traffic that relies on a dataset consisting of network traffic attributes to feed IDS. The presented model is prepared, constructed, fitted, and evaluated by the Python language. Our attractive model should construct and fit in memory, so it listens to the extracted features from network traffic to predict anomalies in real-time.

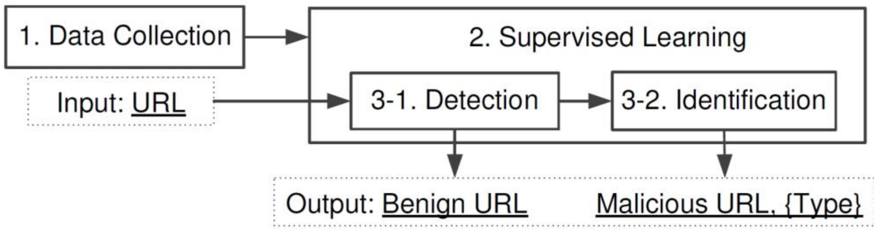


Figure 1:- This Shows working Of this proposed Methods

Literature Survey

Several research and techniques have been proposed on detection of malicious activities, spread of malware, etc., by making datasets and analyzing them with the help of ML. Frank Van Tienhoven et al.,have proposed a method to detect malicious URLs as a binary classification problem and studied the performances of machine learning methods like Naïve Bayes, support vector machine(SVM), Multi-layer perceptron, Decision trees, Random forest(RF), and K–nearest neighbours. The blacklist services are an array of techniques which combine manual reporting, honey pots, and web crawlers with site heuristics.

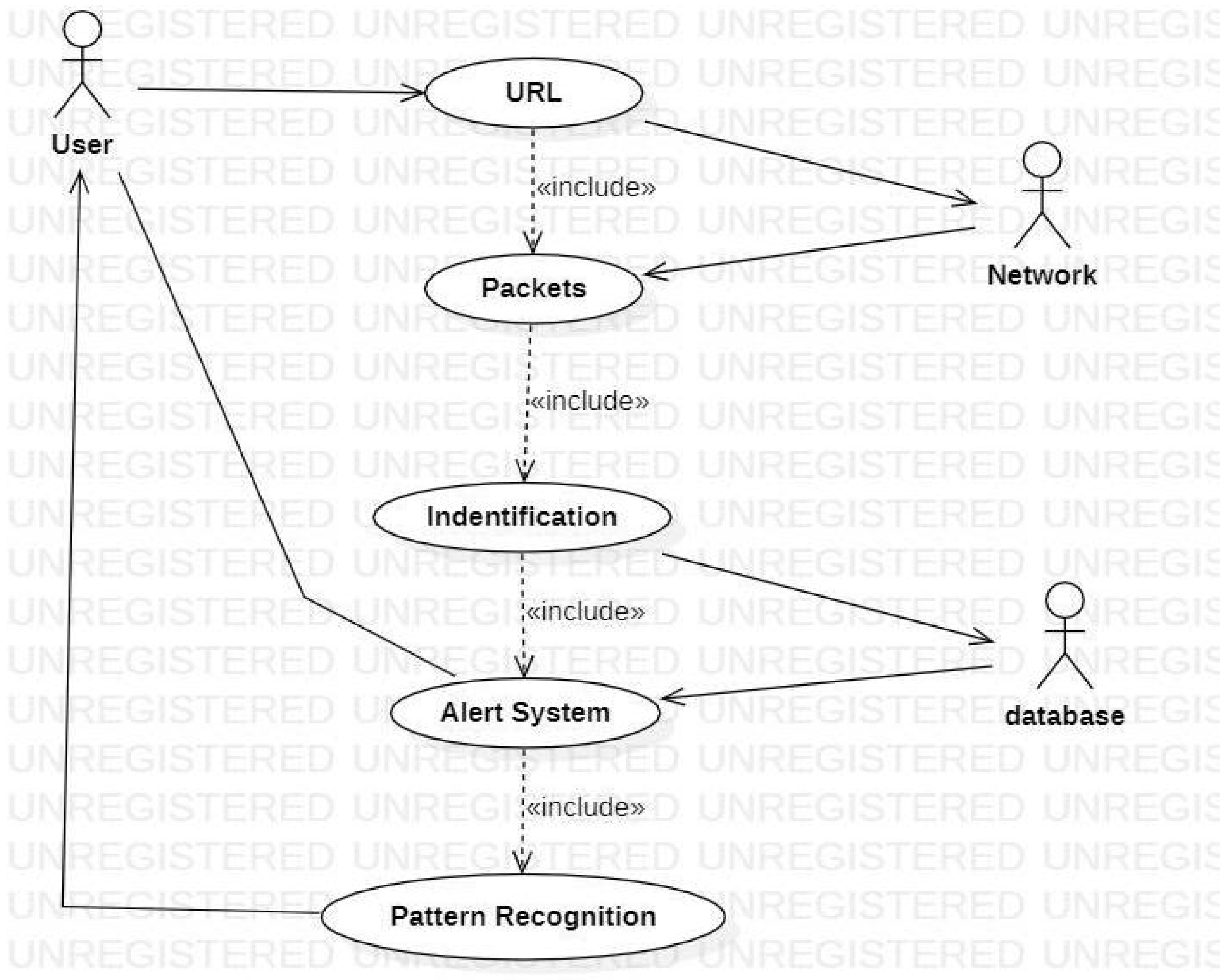
R.V. Bhor and H.K. Khanuja has developed a security mechanism and attack detection technique to avoid sql injection attacks. Sql injection and Denial of service (DOS) are the threats found in web applications. Sql injection attack is the process of altering a Sql statement by the use of web forms.

K. Srividya and A.Mary Sowjanya have developed a method for the analysis of internet messaging and detection of malicious activity. The authors have discussed the adverse effect of internet messaging in social networking sites like Facebook and Whatsapp. The methodology of the research is based on the Latent semantic analysis (LSA). The text messages were processed and alarm if malicious activity were following the emotion analysis technique rather than proper attention to internet messaging.

Pedro Marques has proposed a method to detect web scraping activity using diverse detectors. Robots were employed to extract content and data from a website. Search engine bots, and price comparison bots are considered as legitimate web scraping robots. Copyrighted content scraping and Boosting sale robots are illegitimate robots.

Devan Gol and Nisha Shah have discussed the detection of web application vulnerabilities based on Rational Unified Process (RUP). The authors argued that the existing vulnerable detection tools are failed to detect the latest attacks on the web. The research has demonstrated the vulnerabilities in web applications. Vulnerabilities were occurring from improper codes, computer viruses, or a cross sided script (XSS) and SQL injection attack (SQLIA). SQLIA is against a database driven application. It will inject invalid input strings into the database and modify them for deliberate usage. A successful attack will pass a SQL attack code into the back – end system and execute the vulnerable application.

Use Case Diagram



Some Of the Malicious activities their detection and Preventions are as follows:-

1. AWS honeypot

In computer terminology, a honeypot is a computer security mechanism set to detect, deflect, or, in some manner, counteract attempts at unauthorized use of information systems. Generally, a honeypot consists of data (for example, in a network site) that appears to be a legitimate part of the site and contains information or resources of value to attackers. It is actually isolated, monitored, and capable of blocking or analyzing the attackers. This is similar to police sting operations, colloquially known as "baiting" a suspect.
In human words(source: AWS AWF(web Application WireFall)): because of being online, companies face continual security threats and challenges, like :

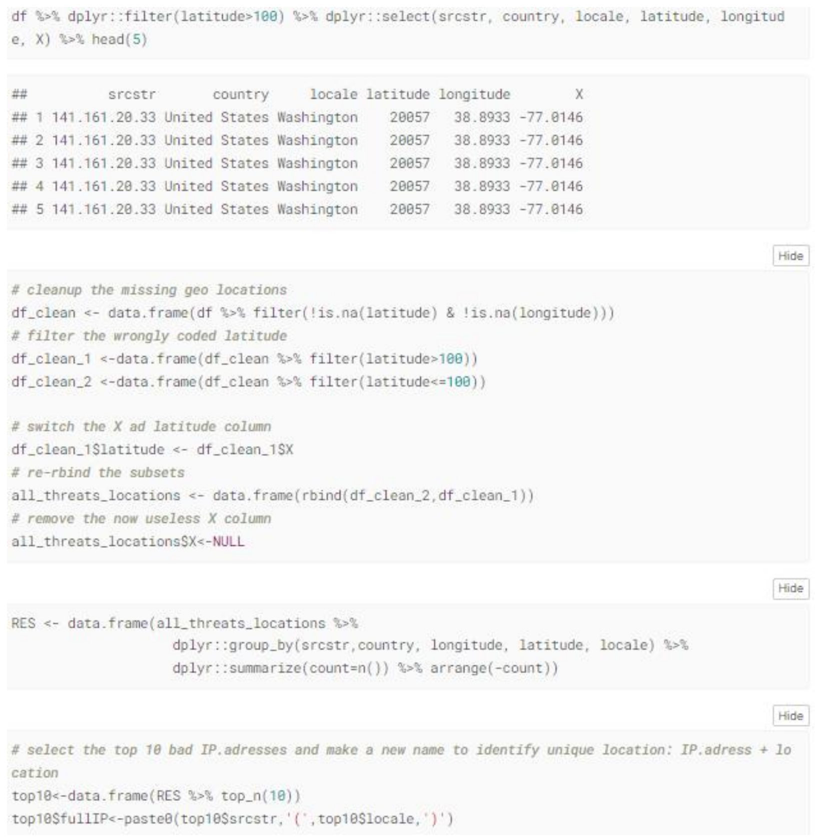
- HTTP flood attacks,
- distributed denial of service (DDoS) attacks,
- malicious activity in off hours and access attempts by bad IP addresses,
- SQL-injection attacks designed to extract data,
- cross-site scripting attacks (XSS) that could insert malicious code into web pages, designed to take down its online activity.

Data Preparation

```
df<-read.csv('../input/AWS_Honeypot_marx-geo.csv',sep=',',stringsAsFactors=F)
```

Cleaning Feature Creation

- there are several missing values that need to be removed because longitude and latitude are missing, since these are crucial features for the geolocation.
- maybe an imputation by IP.adress is possible
- at first I thought that the last column X was useless but it appears that it contains the correct values for few rows having a wrong latitude(value is over 20000)



```
df %>% dplyr::filter(latitude>100) %>% dplyr::select(srcstr, country, locale, latitude, longitud
e, X) %>% head(5)
```

```
##          srcstr       country     locale latitude longitude        X
## 1 141.161.20.33 United States Washington    20057   38.8933 -77.0146
## 2 141.161.20.33 United States Washington    20057   38.8933 -77.0146
## 3 141.161.20.33 United States Washington    20057   38.8933 -77.0146
## 4 141.161.20.33 United States Washington    20057   38.8933 -77.0146
## 5 141.161.20.33 United States Washington    20057   38.8933 -77.0146
```

```
# cleanup the missing geo locations
df_clean <- data.frame(df %>% filter(!is.na(latitude) & !is.na(longitude)))
# filter the wrongly coded latitude
df_clean_1 <-data.frame(df_clean %>% filter(latitude>100))
df_clean_2 <-data.frame(df_clean %>% filter(latitude<=100))

# switch the X ad latitude column
df_clean_1$latitude <- df_clean_1$X
# re-rbind the subsets
all_threats_locations <- data.frame(rbind(df_clean_2,df_clean_1))
# remove the now useless X column
all_threats_locations$X<-NULL
```

```
RES <- data.frame(all_threats_locations %>%
                    dplyr::group_by(srcstr,country, longitude, latitude, locale) %>%
                    dplyr::summarize(count=n()) %>% arrange(-count))
```

```
# select the top 10 bad IP.adresses and make a new name to identify unique location: IP.adress + lo
cation
top10<-data.frame(RES %>% top_n(10))
top10$fullIP<-paste0(top10$srcstr,'(',top10$locale,')')
```

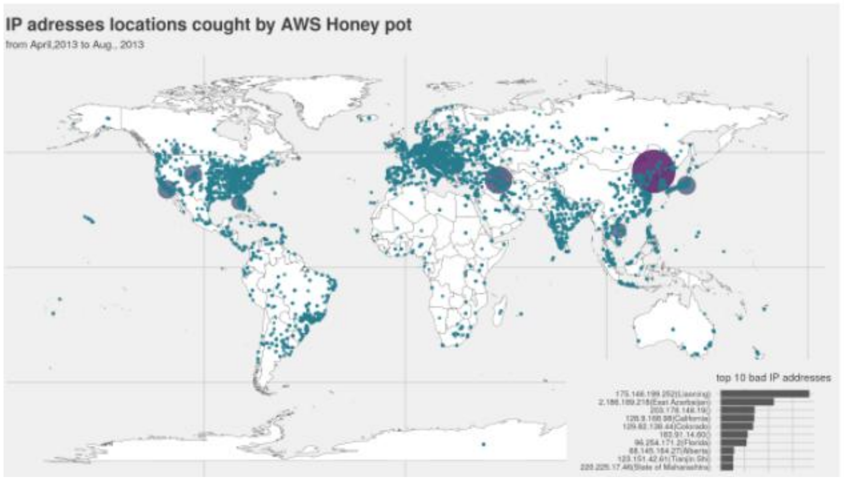
Geolocation Of attacks

- map + histogram of the top 10 locations
- custom color palette
- remove the size and color legend
- add the histogram of the top 10 bad IP.adresses, it will give the scale of the number of bad IP.adresses attacks

```
histo<-ggplotGrob(
  top10 %>% ggplot(aes(x=reorder(fullIP,count),y=count)) +
    geom_bar(stat='identity') + coord_flip() + theme_fivethirtyeight() +
    theme(axis.text=element_text(size=8)) + labs(subtitle='top 10 bad IP addresses'))

countries_map <-map_data("world")
world_map<-ggplot() +
  geom_map(data = countries_map, map = countries_map,aes(x = long, y = lat, map_id = region, grou
p = group), fill = "white", color = "black", size = 0.1) +
  theme_fivethirtyeight() +
  theme(axis.text=element_blank())
```

Hide

```
world_map +
  geom_point(data=RES,aes(x=longitude,y=latitude,size=count,color=count),alpha=.75) +
  scale_color_gradient2(name='',low = "#B8DE29FF", mid = "#287D8EFF", high = "#440154FF") +
  guides(color=FALSE,size=F) +
  scale_radius(range=c(1,20)) +
  labs(title='IP adresses locations cought by AWS Honey pot',
       subtitle='from April,2013 to Aug., 2013') +
  annotation_custom(grob = histo, xmin = 80, xmax = 210, ymin = -100, ymax = -40)
```

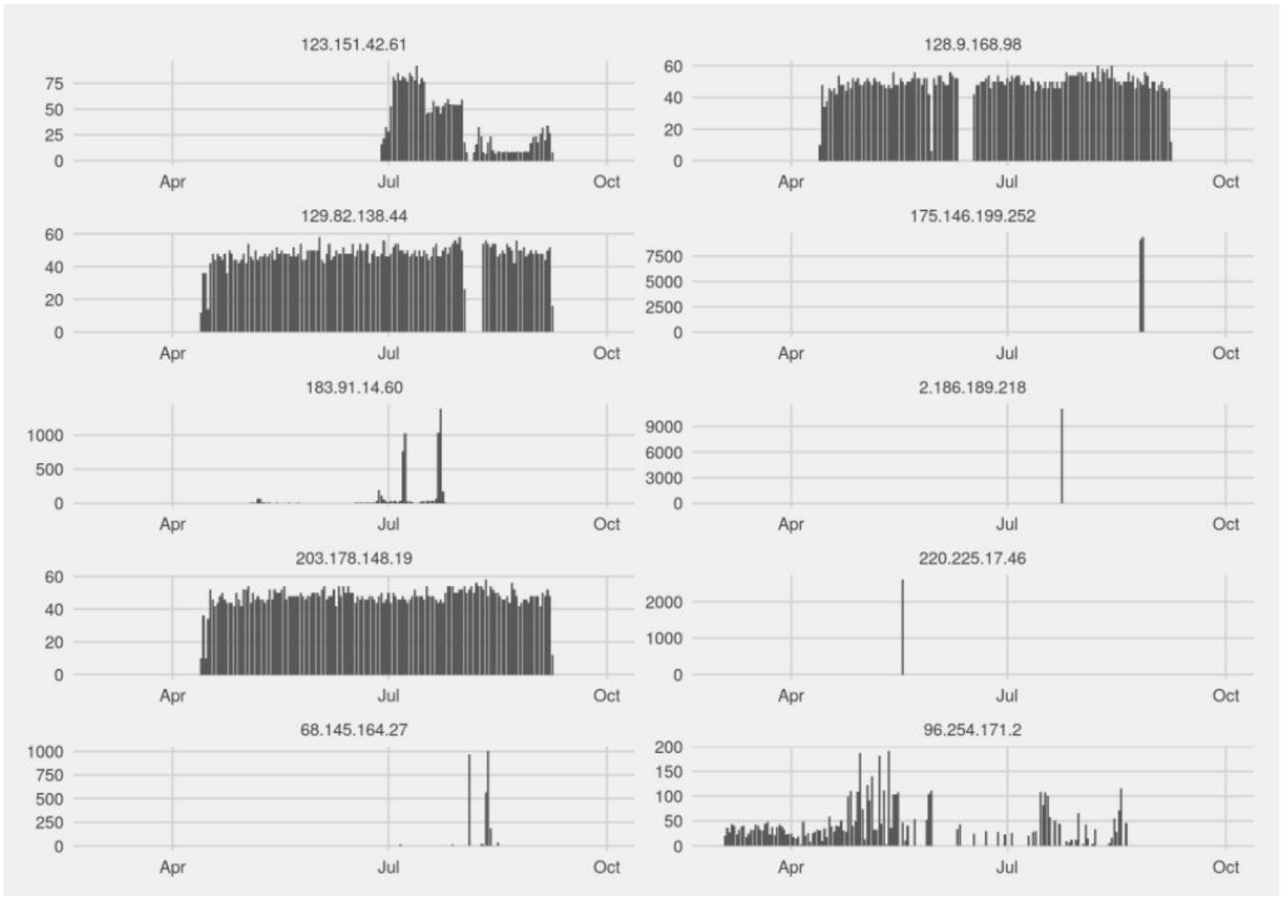

IP adresses locations cought by AWS Honey pot
from April,2013 to Aug., 2013
top 10 bad IP addresses
175.146.199.252(Liaoning)
2.186.189.218(East Azarbaijan)
203.178.148.19()
128.9.168.98(California)
129.82.138.44(Colorado)
183.91.14.60()
96.254.171.2(Florida)
68.145.164.27(Alberta)
123.151.42.61(Tianjin Shi)
220.225.17.46(State of Maharashtra)

Time line of the top 10 attackers

- decode year/month/day from datetime and convert to As.Date
- filter the top 10 bad IP.adresses
- set limit in time and y-axis free
- all_threats_locations$month<-sapply(all_threats_locations$datetime,**function**(x) as.numeric(strsplit(strsplit(x,' ')[[1]][1],'/')[[1]][1]))
- all_threats_locations$day<-sapply(all_threats_locations$datetime,**function**(x) as.numeric(strsplit(strsplit(x,' ')[[1]][1],'/')[[1]][2]))
- all_threats_locations$year<-2000 + sapply(all_threats_locations$datetime,**function**(x) as.numeric(strsplit(strsplit(x,' ')[[1]][1],'/')[[1]][3]))
- all_threats_locations$hour<-sapply(all_threats_locations$datetime,**function**(x) as.numeric(strsplit(strsplit(x,' ')[[1]][2],':')[[1]][1]))
- all_threats_locations$min<-sapply(all_threats_locations$datetime,**function**(x) as.numeric(strsplit(strsplit(x,' ')[[1]][2],':')[[1]][2]))
- all_threats_locations$DateTS<-as.POSIXct(
-   paste0(all_threats_locations$year,'-',
-         all_threats_locations$month,'-',

- all_threats_locations$day,' ',
- all_threats_locations$hour,':',
- all_threats_locations$min,':00'),format= "%Y-%m-%d %H:%M:%S")
- *#all_threats_locations$DateTS<-as.Date(paste0(all_threats_locations$year,'-',all_threats_locations$month,'-',all_threats_locations$day,))*
-
- top10.ip.adress<-top10$srcstr
- attackers<- data.frame(all_threats_locations %>% dplyr::filter(srcstr %**in**% top10.ip.adress))

```
# for this plot the grouping is by day

lims <- as.POSIXct(strptime(c("2013-03-01 00:00:00","2013-10-01 23:59:59"), format = "%Y-%m-%d %H:%M:%S"))

attackers %>%

  dplyr::select(year, month, day, srcstr) %>%

  mutate(dd = as.POSIXct(as.Date(paste0(year,'-',month,'-',day), format= "%Y-%m-%d"))) %>%

  dplyr::group_by(srcstr,dd) %>%

  dplyr::summarize(count=n()) %>%

  ggplot(aes(x=dd,y=count,group=1)) + geom_histogram(stat='identity',aes(group=1)) +

  theme_fivethirtyeight() +

  scale_x_datetime(limits =lims) + facet_wrap(~srcstr, ncol=2, scales='free')
```

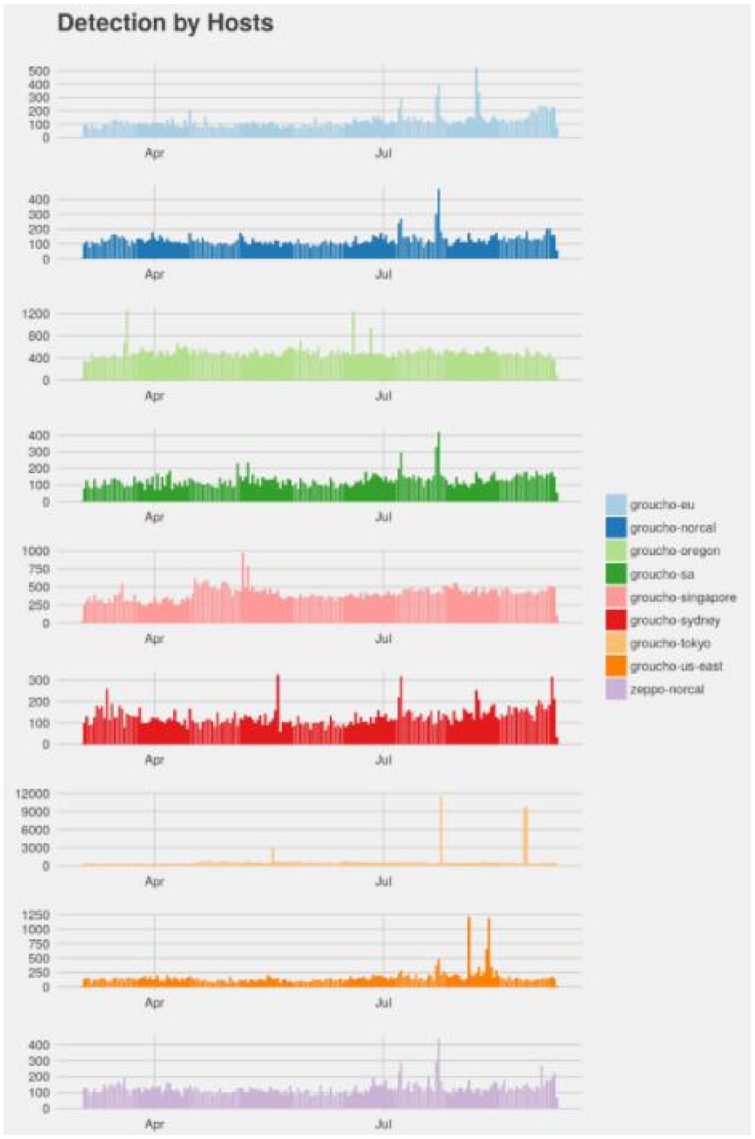


Detection by Host

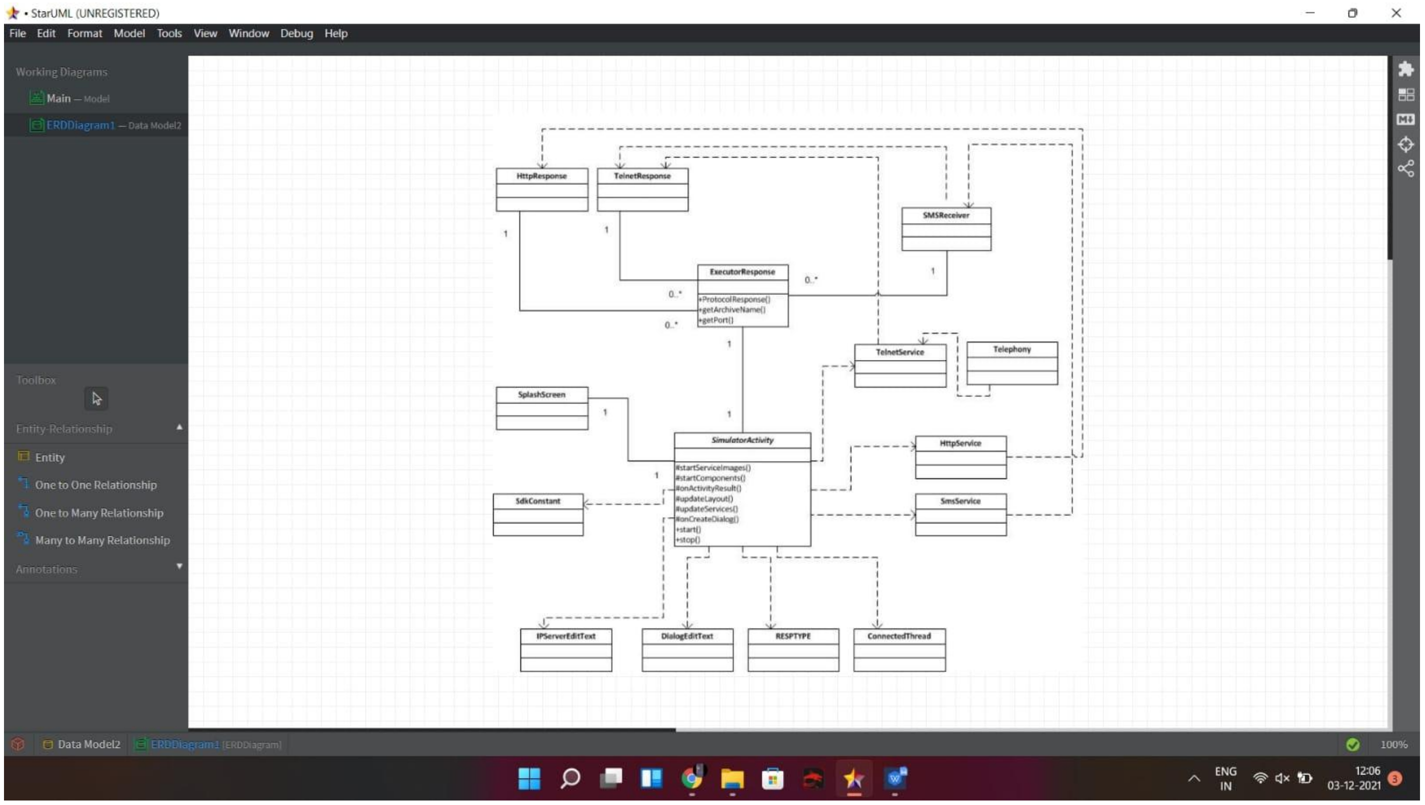
There are several host, ie honey pot location, in the world. Grouping the data by Date and Host will show if they have detected a common attack

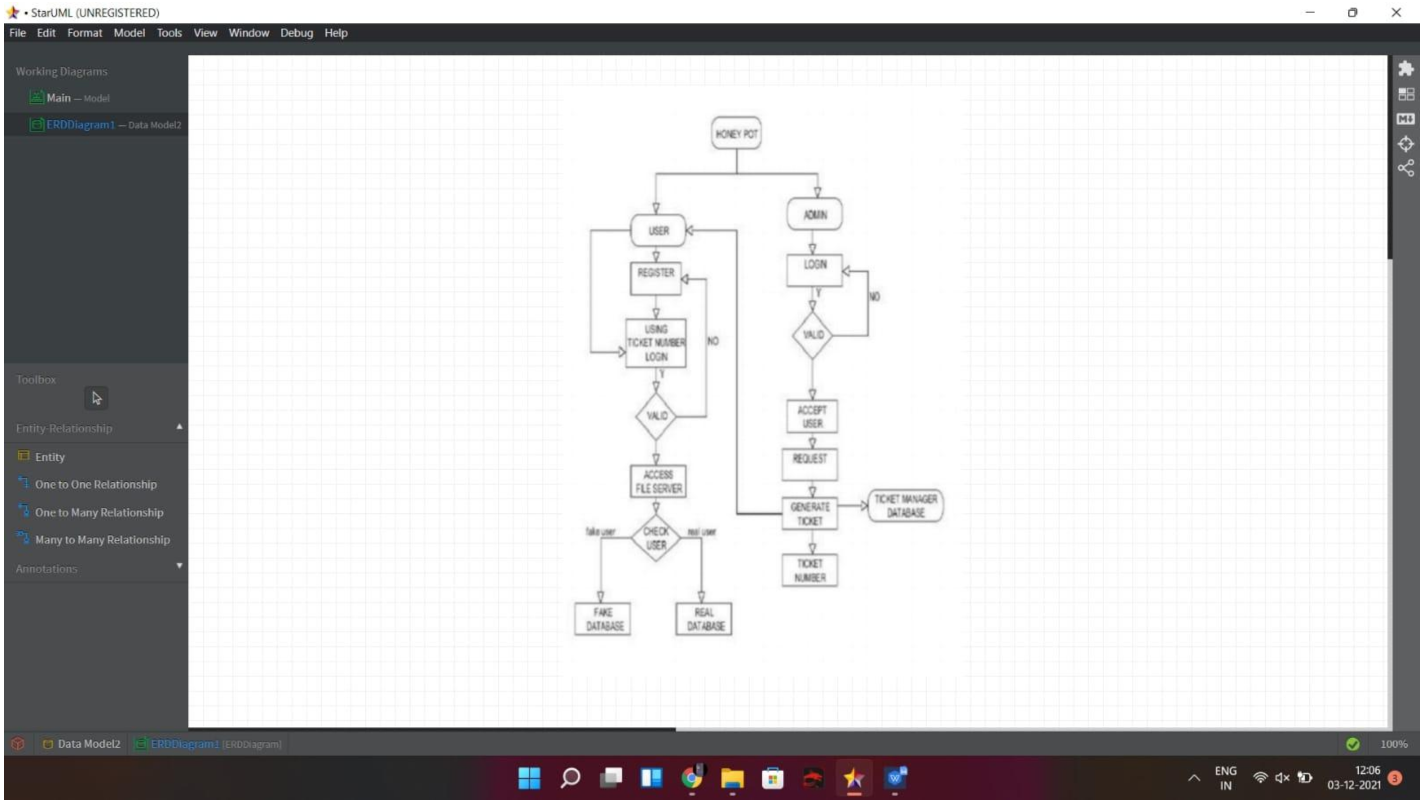
Hide

```
all_threats_locations %>% dplyr::select(year, month, day, host) %>% mutate(dd = as.POSIXct(as.Date(paste0(year,'-',month,'-',day), format= "%Y-%m-%d"))) %>% dplyr::group_by(host,dd) %>% dplyr::summarize(count=n()) %>%

  ggplot(aes(x=dd,y=count,group=host,fill=host)) + geom_histogram(stat='identity',aes(group=host)) + scale_fill_brewer(name='',palette='Paired') +

  theme_fivethirtyeight() + facet_wrap(~host,scales='free',ncol=1) + theme(legend.position='right',legend.direction='vertical',strip.text.x = element_text(size=0)) + labs(title='Detection by Hosts')
```


Detection by Hosts
groucho-eu
groucho-norcal
groucho-oregon
groucho-sa
groucho-singapore
groucho-sydney
groucho-tokyo
groucho-us-east
zeppo-norcal
Apr
Jul

UML Diagram:-


StarUML (UNREGISTERED)
File Edit Format Model Tools View Window Debug Help
Working Diagrams
Main — Model
ERDDiagram1 — Data Model2
Toolbox
Entity-Relationship
Entity
One to One Relationship
One to Many Relationship
Many to Many Relationship
Annotations
HttpResponse
TelnetResponse
SMSReceiver
ExecutorResponse
+ProtocolResponse()
+getArchiveName()
+getPort()
1
1
0..*
0..*
0..*
1
1
TelnetService
Telephony
SplashScreen
1
1
SimulatorActivity
#startServiceImages()
#startComponents()
#onActivityResult()
#updateLayout()
#updateServices()
#onCreateDialog()
+start()
+stop()
1
HttpService
SdkConstant
SmsService
IPServerEditText
DialogEditText
RESPTYPE
ConnectedThread
Data Model2
ERDDiagram1 [ERDDiagram]
100%
ENG IN
12:06
03-12-2021

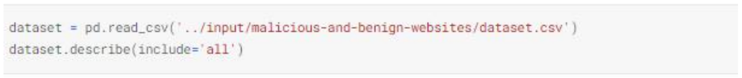

StarUML (UNREGISTERED)
File Edit Format Model Tools View Window Debug Help
Working Diagrams
Main — Model
ERDDiagram1 — Data Model2
Toolbox
Entity-Relationship
Entity
One to One Relationship
One to Many Relationship
Many to Many Relationship
Annotations
HONEY POT
USER
ADMIN
REGISTER
LOGIN
Y
NO
USING TICKET NUMBER LOGIN
NO
VALID
Y
VALID
ACCEPT USER
REQUEST
ACCESS FILE SERVER
GENERATE TICKET
TICKET MANAGER DATABASE
fake user
CHECK USER
real user
TICKET NUMBER
FAKE DATABASE
REAL DATABASE
Data Model2
ERDDiagram1 [ERDDiagram]
100%
ENG IN
12:06
03-12-2021

2. Malicious and Benign Websites - Neural Network

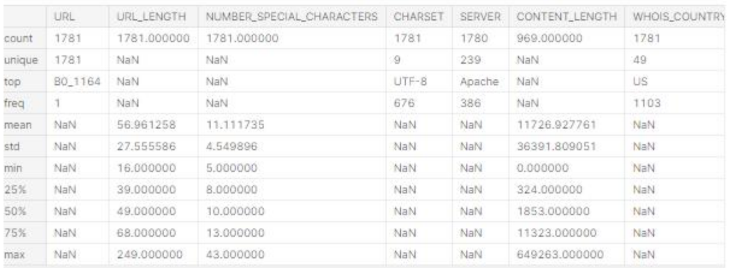
Web Security is a challenging task amidst ever rising threats on the Internet. With billions of websites active on the Internet, and hackers evolving newer techniques to trap web users, machine learning offers promising techniques to detect malicious websites. The dataset described in this manuscript is meant for such machine learning-based analysis of malicious and benign webpages. The data has been collected from the Internet using a specialized focused web crawler. The dataset comprises various extracted attributes, and also raw webpage content including JavaScript code. It supports both supervised and unsupervised learning. Malicious websites are of great concern due it is a problem to analyze one by one and to index each URL in a black list. Unfortunately, there is a lack of datasets with malicious and benign web characteristics.
This is an important topic and one of the most difficult thing to process, according to other articles and another open resource, we used three black list:

- machinelearning.inginf.units.it/data-andtools/hidden-fraudulent-urls-dataset
- malwaredomainlist.com
- zeuztacker.abuse.ch

OVERVIEW



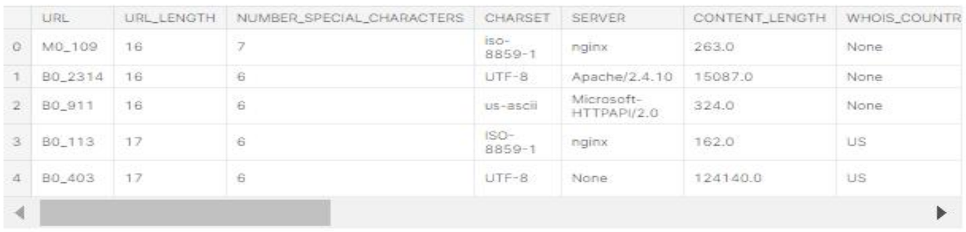
```
dataset = pd.read_csv('../input/malicious-and-benign-websites/dataset.csv')
dataset.describe(include='all')
```



	URL	URL_LENGTH	NUMBER_SPECIAL_CHARACTERS	CHARSET	SERVER	CONTENT_LENGTH	WHOIS_COUNTRY
count	1781	1781.000000	1781.000000	1781	1780	969.000000	1781
unique	1781	NaN	NaN	9	239	NaN	49
top	B0_1164	NaN	NaN	UTF-8	Apache	NaN	US
freq	1	NaN	NaN	676	386	NaN	1103
mean	NaN	56.961258	11.111735	NaN	NaN	11726.927761	NaN
std	NaN	27.555586	4.549896	NaN	NaN	36391.809051	NaN
min	NaN	16.000000	5.000000	NaN	NaN	0.000000	NaN
25%	NaN	39.000000	8.000000	NaN	NaN	324.000000	NaN
50%	NaN	49.000000	10.000000	NaN	NaN	1853.000000	NaN
75%	NaN	68.000000	13.000000	NaN	NaN	11323.000000	NaN
max	NaN	249.000000	43.000000	NaN	NaN	649263.000000	NaN

Data Analysis and Preparation

```
dataset.head()
```



	URL	URL_LENGTH	NUMBER_SPECIAL_CHARACTERS	CHARSET	SERVER	CONTENT_LENGTH	WHOIS_COUNTR
0	M0_109	16	7	iso-8859-1	nginx	263.0	None
1	B0_2314	16	6	UTF-8	Apache/2.4.10	15087.0	None
2	B0_911	16	6	us-ascii	Microsoft-HTTPAPI/2.0	324.0	None
3	B0_113	17	6	ISO-8859-1	nginx	162.0	US
4	B0_403	17	6	UTF-8	None	124140.0	US

5 rows × 21 columns

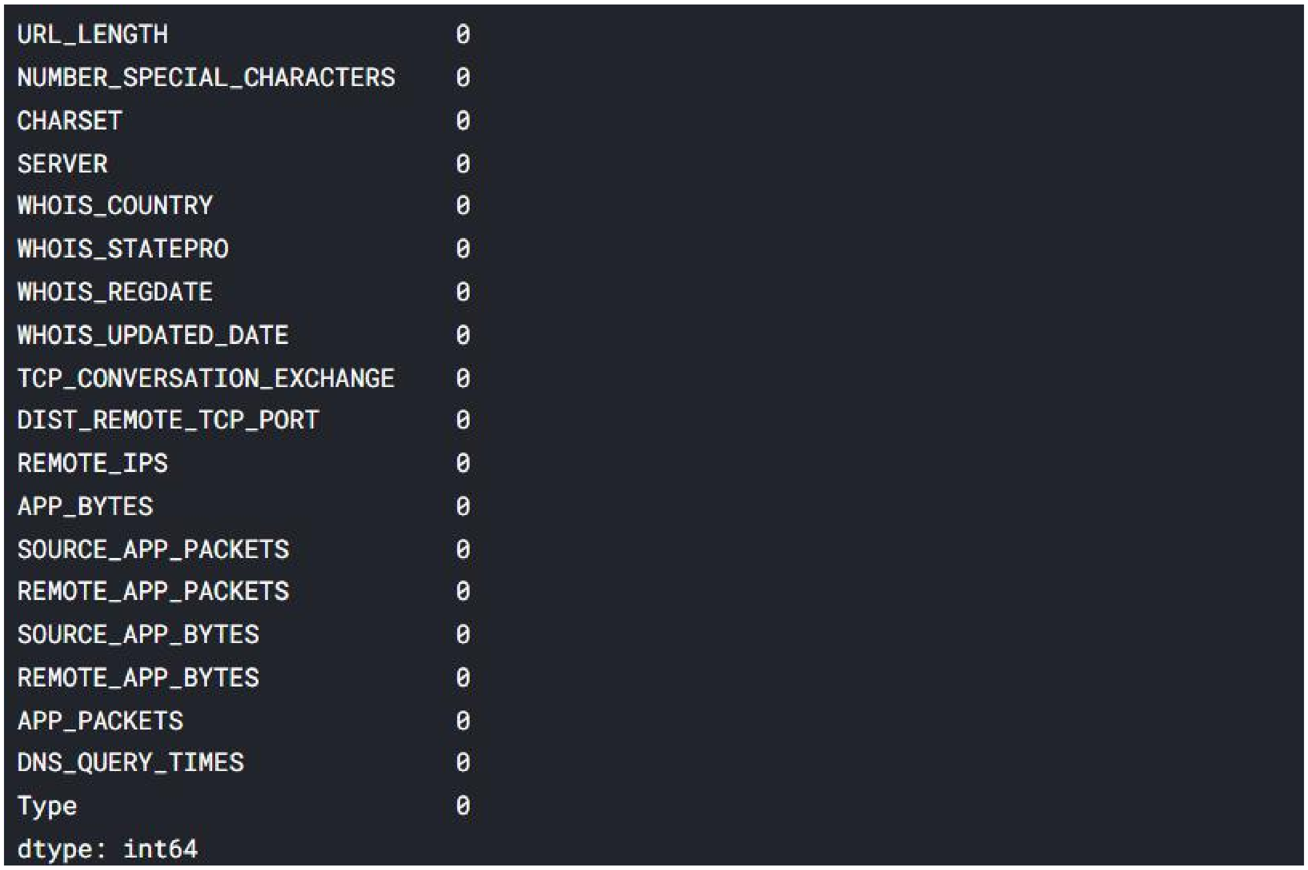
The URL column is a unique identifier so we may as well remove that.

```
dataset.drop('URL', axis =1, inplace=True)
```

dataset.drop('URL', axis =1, inplace=True)

In [5]:

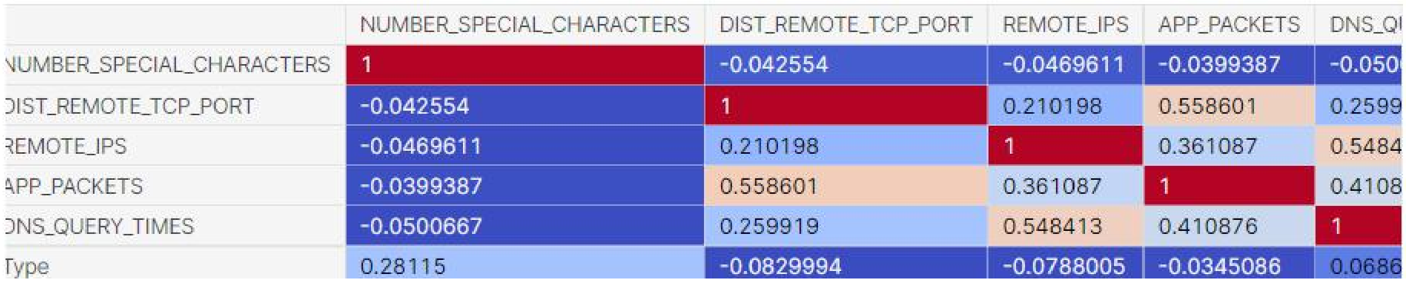
linkcode
# Look for null values
print(dataset.isnull().sum())



```
URL_LENGTH                   0
NUMBER_SPECIAL_CHARACTERS    0
CHARSET                      0
SERVER                       0
WHOIS_COUNTRY                0
WHOIS_STATEPRO               0
WHOIS_REGDATE                0
WHOIS_UPDATED_DATE           0
TCP_CONVERSATION_EXCHANGE    0
DIST_REMOTE_TCP_PORT         0
REMOTE_IPS                   0
APP_BYTES                    0
SOURCE_APP_PACKETS           0
REMOTE_APP_PACKETS           0
SOURCE_APP_BYTES             0
REMOTE_APP_BYTES             0
APP_PACKETS                  0
DNS_QUERY_TIMES              0
Type                         0
dtype: int64
```

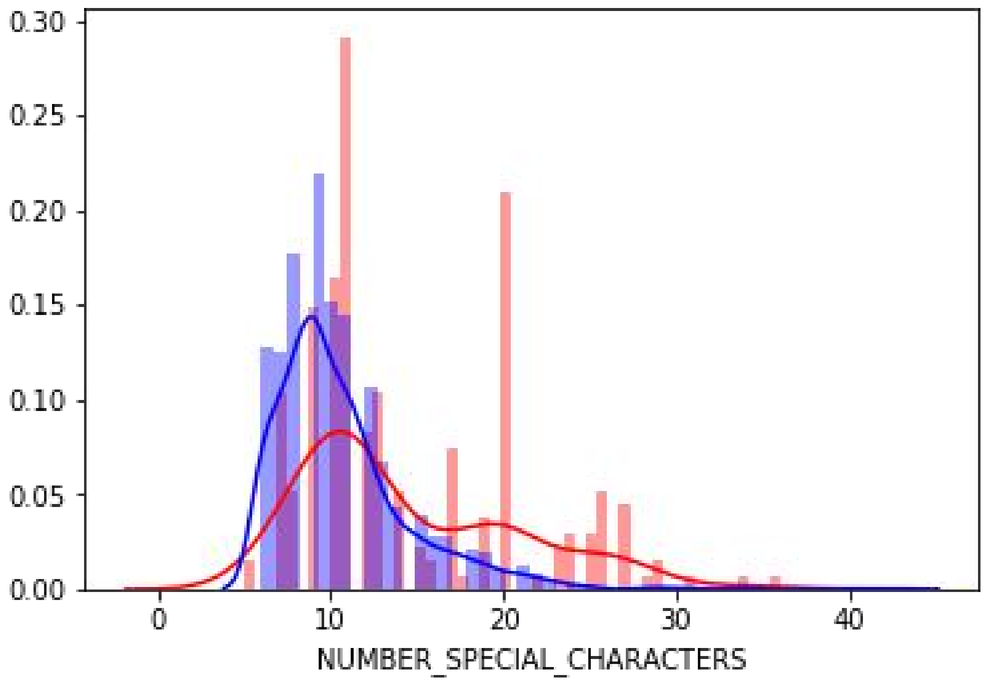
There are null values each for the DNS_QUERY_TIMES and SERVER columns, so we could easily drop these records / place a dummy value instead without affecting the data too much. The CONTENT_LENGTH column is a bit more concerning, we can't afford to drop that many records (almost half the dataset) and interpolating might distort the data somewhat. Given that there are plenty of other features, I'm choosing to drop the column.

```
dataset.drop(['TCP_CONVERSATION_EXCHANGE','URL_LENGTH','APP_BYTES','SOURCE_APP_PACKETS','REMOTE_APP_PACKETS','SOURCE_APP_BYTES','REMOTE_APP_BYTES'], axis = 1, inplace=True)
corr = dataset.corr()
corr.style.background_gradient(cmap='coolwarm')
```

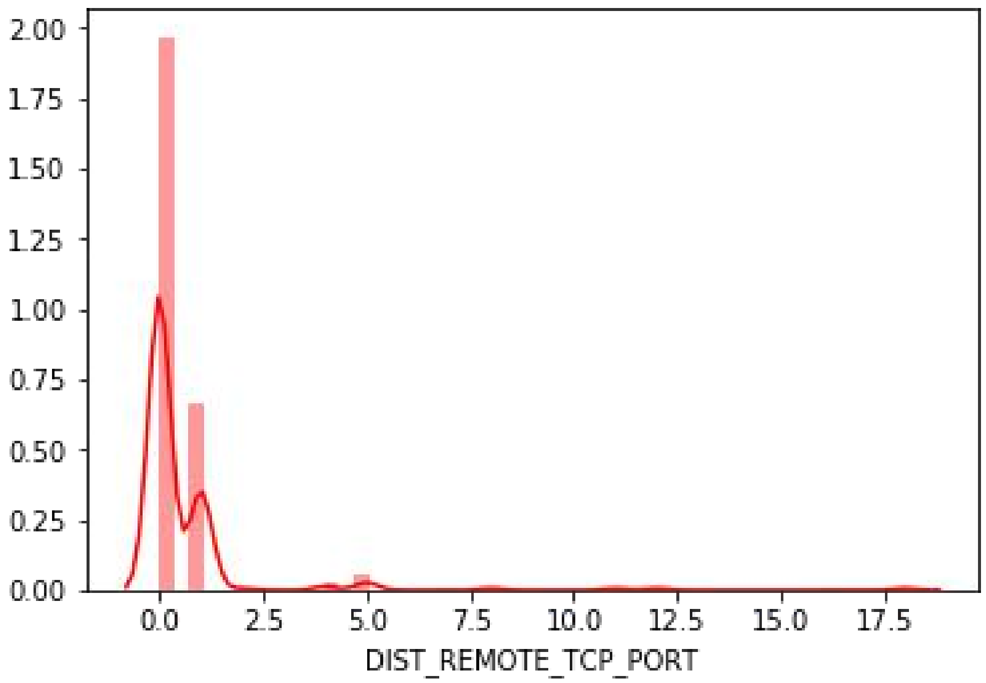


	NUMBER_SPECIAL_CHARACTERS	DIST_REMOTE_TCP_PORT	REMOTE_IPS	APP_PACKETS	DNS_Q
NUMBER_SPECIAL_CHARACTERS	1	-0.042554	-0.0469611	-0.0399387	-0.050
DIST_REMOTE_TCP_PORT	-0.042554	1	0.210198	0.558601	0.2599
REMOTE_IPS	-0.0469611	0.210198	1	0.361087	0.5484
APP_PACKETS	-0.0399387	0.558601	0.361087	1	0.4108
DNS_QUERY_TIMES	-0.0500667	0.259919	0.548413	0.410876	1
Type	0.28115	-0.0829994	-0.0788005	-0.0345086	0.0686

```
 import seaborn as sns
sns.distplot(dataset.loc[dataset['Type'] == 1]['NUMBER_SPECIAL_CHARACTERS'], bins = 50, color='red')
sns.distplot(dataset.loc[dataset['Type'] == 0]['NUMBER_SPECIAL_CHARACTERS'], bins = 50, color='blue')
```



```
sns.distplot(dataset.loc[dataset['Type'] == 1]['DIST_REMOTE_TCP_PORT'], bins = 50, color='red')
```



Let's have a play with the parameters to see if this can be improved.

In [29]:

```
def predict( X_train, y_train, **kwargs):
    mlp = MLPClassifier(**kwargs, random_state=1)
    mlp.fit(X_train, y_train)
    return mlp.predict(X_test)
```

In [30]:

```
def calculateScoresNoOutput(y_test, predictions):
    accuracy = 100*accuracy_score(y_test, predictions)
    precision = 100*precision_score(y_test, predictions)
    recall = 100*recall_score(y_test, predictions)
    f1 = 100*f1_score(y_test, predictions)
    return {'Accuracy':accuracy, 'F1': f1}
```

In [31]:

```
linkcode
# Let's try the different solvers
solvers = ['lbfgs', 'sgd', 'adam']
results = []
for solver in solvers:
    result_dict = calculateScoresNoOutput(y_test, predict(X_train, y_train, solver=solver))
    result_dict['Solver'] = solver
    results.append(result_dict)
df = pd.DataFrame(results, columns = ['Solver','Accuracy', 'F1'])
df
```

Out[31]:

Solver	Accuracy	F1	
0	lbfgs	88.700565	52.380952
1	sgd	87.947269	17.948718
2	adam	88.323917	38.000000

```
def try_different_values(values, column_name, X_train, y_train, **kwargs):
    results = []
    for value in values:
        kwargs[column_name] = value
        result_dict = calculateScoresNoOutput(y_test, predict(X_train, y_train, **kwargs))
        result_dict[column_name] = value
        results.append(result_dict)
    df = pd.DataFrame(results, columns = [column_name,'Accuracy', 'F1'])
    return df
```

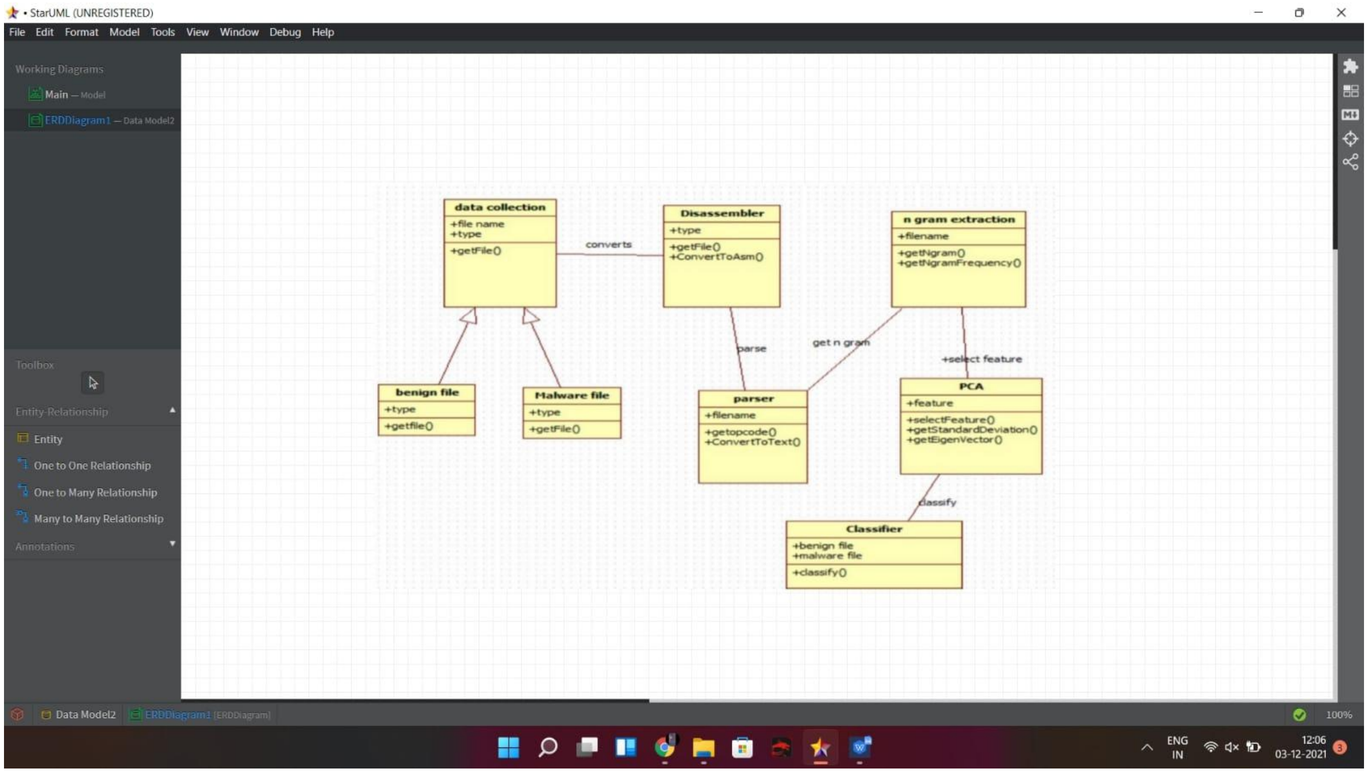
In [33]:

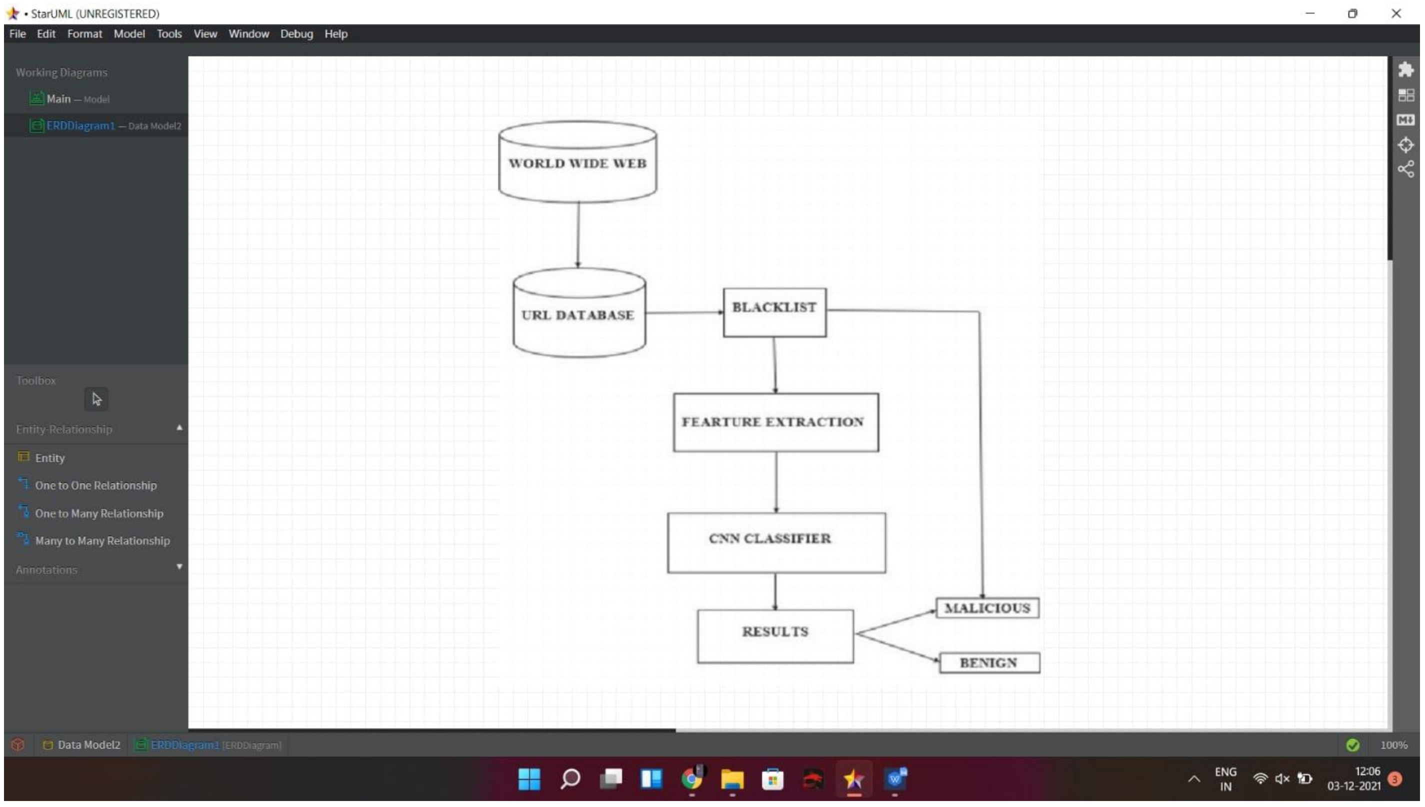
```
activations = ['identity', 'logistic', 'tanh', 'relu']
try_different_values(activations, 'activation', X_train, y_train, solver='lbfgs')
```

Out[33]:

	activation	Accuracy	F1
0	identity	86.440678	16.279070
1	logistic	90.207156	57.377049
2	tanh	89.077213	55.384615
3	relu	88.700565	52.380952

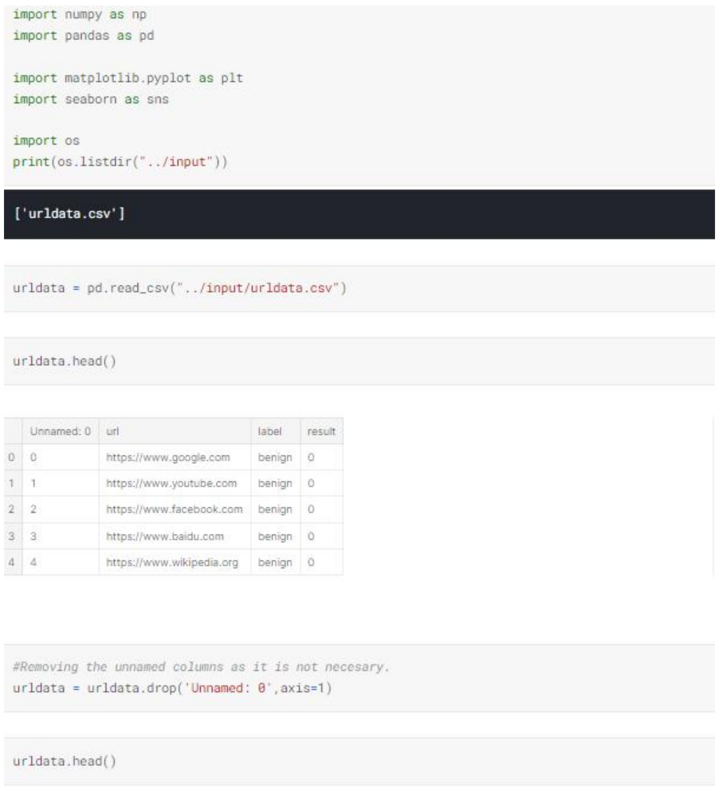
UML Diagram:-


StarUML (UNREGISTERED)
File Edit Format Model Tools View Window Debug Help
Working Diagrams
Main — Model
ERDDiagram1 — Data Model2
data collection
+file name
+type
+getFile()
converts
Disassembler
+type
+getFile()
+ConvertToAsm()
n gram extraction
+filename
+getNgram()
+getNgramFrequency()
benign file
+type
+getfile()
Malware file
+type
+getFile()
parse
get n gram
parser
+filename
+getopcode()
+ConvertToText()
+select feature
PCA
+feature
+selectFeature()
+getStandardDeviation()
+getEigenVector()
classify
Classifier
+benign file
+malware file
+classify()
Toolbox
Entity-Relationship
Entity
One to One Relationship
One to Many Relationship
Many to Many Relationship
Annotations
Data Model2
ERDDiagram1 [ERDDiagram]
100%
ENG IN
12:06
03-12-2021


StarUML (UNREGISTERED)
File Edit Format Model Tools View Window Debug Help
Working Diagrams
Main — Model
ERDDiagram1 — Data Model2
WORLD WIDE WEB
URL DATABASE
BLACKLIST
FEARTURE EXTRACTION
CNN CLASSIFIER
RESULTS
MALICIOUS
BENIGN
Toolbox
Entity-Relationship
Entity
One to One Relationship
One to Many Relationship
Many to Many Relationship
Annotations
Data Model2
ERDDiagram1 [ERDDiagram]
100%
ENG IN
12:06
03-12-2021

3. Malicious URL Detection using MLP

Currently, the risk of network information insecurity is increasing rapidly in number and level of danger. The methods mostly used by hackers today are to attack end-to-end technology and exploit human vulnerabilities. These techniques include social engineering, phishing, pharming, etc. One of the steps in conducting these attacks is to deceive users with malicious Uniform Resource Locators (URLs). As a result, malicious URL detection is of great interest nowadays. There have been several scientific studies showing a number of methods to detect malicious URLs based on machine learning and deep learning techniques. In this paper, we propose a malicious URL detection method using machine learning techniques based on our proposed URL behaviors and attributes. Moreover, big data technology is also exploited to improve the capability of detecting malicious URLs based on abnormal behaviors. In short, the proposed detection system consists of a new set of URLs, features and behaviors, a machine learning algorithm, and big data technology. The experimental results show that the proposed URL attributes and behavior can help improve the ability to detect malicious URL significantly. This is suggested that the proposed system may be considered as an optimized and friendly used solution for malicious URL detection.



```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns

import os
print(os.listdir("../input"))
```

```
['urldata.csv']
```

```
urldata = pd.read_csv("../input/urldata.csv")
```

```
urldata.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

```
#Removing the unnamed columns as it is not necesary.
urldata = urldata.drop('Unnamed: 0',axis=1)
```

```
urldata.head()
```

DATA PREPROCESSING

```
#Importing dependencies
from urllib.parse import urlparse
from tld import get_tld
import os.path
```

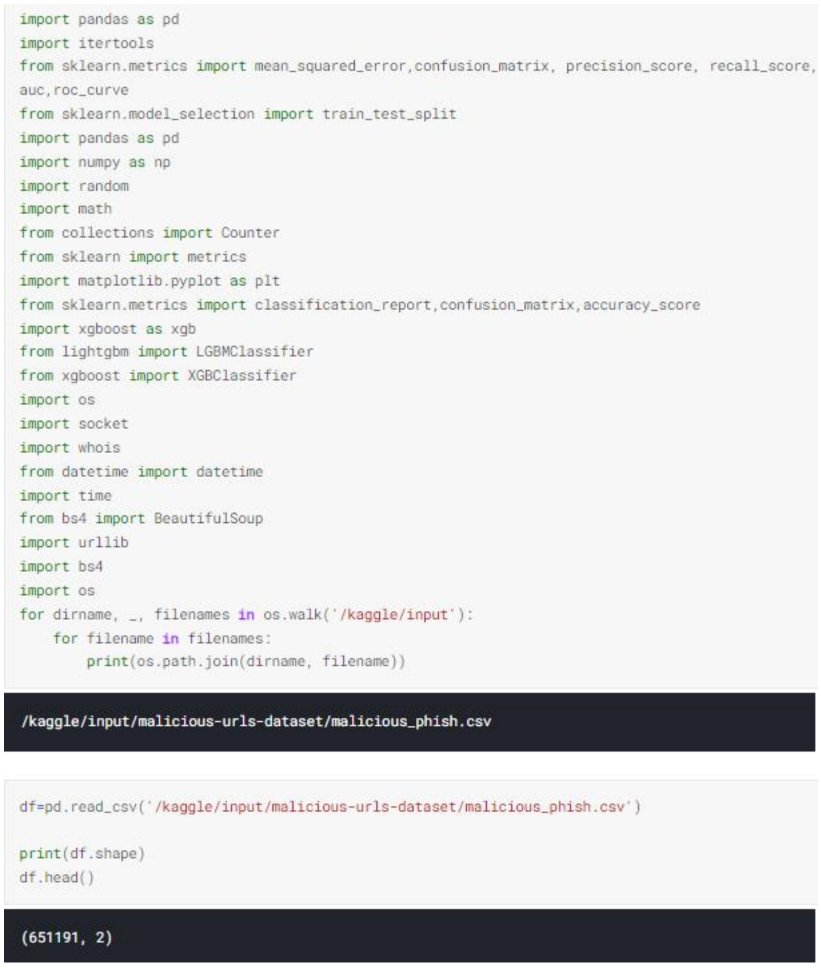
```
#Length of URL
urldata['url_length'] = urldata['url'].apply(lambda i: len(str(i)))
```

```
#Hostname Length
urldata['hostname_length'] = urldata['url'].apply(lambda i: len(urlparse(i).netloc))
```

```
#Path Length
urldata['path_length'] = urldata['url'].apply(lambda i: len(urlparse(i).path))
```

```
#First Directory Length
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))
```

```
#Length of Top Level Domain
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))
```



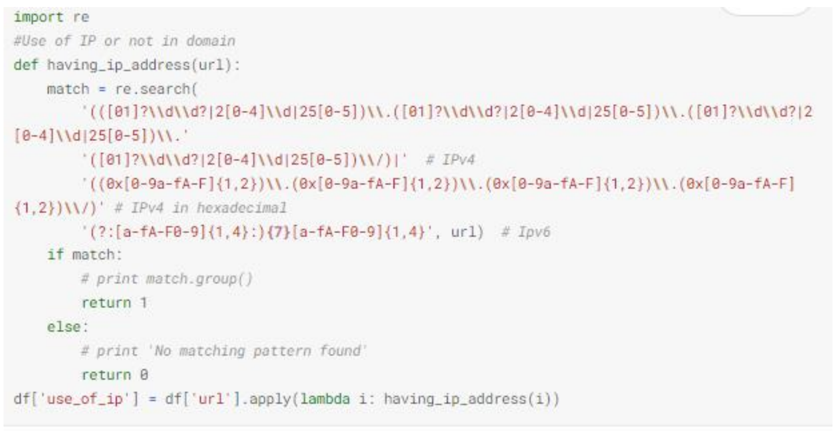
```
import pandas as pd
import itertools
from sklearn.metrics import mean_squared_error,confusion_matrix, precision_score, recall_score,
auc,roc_curve
from sklearn.model_selection import train_test_split
import pandas as pd
import numpy as np
import random
import math
from collections import Counter
from sklearn import metrics
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score
import xgboost as xgb
from lightgbm import LGBMClassifier
from xgboost import XGBClassifier
import os
import socket
import whois
from datetime import datetime
import time
from bs4 import BeautifulSoup
import urllib
import bs4
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
/kaggle/input/malicious-urls-dataset/malicious_phish.csv
```

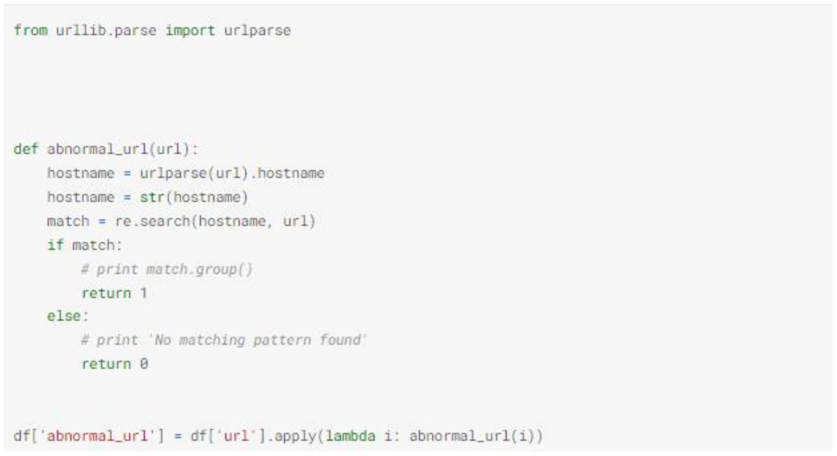
```
df=pd.read_csv('/kaggle/input/malicious-urls-dataset/malicious_phish.csv')

print(df.shape)
df.head()
```

```
(651191, 2)
```

FEATURE ENGINEERING



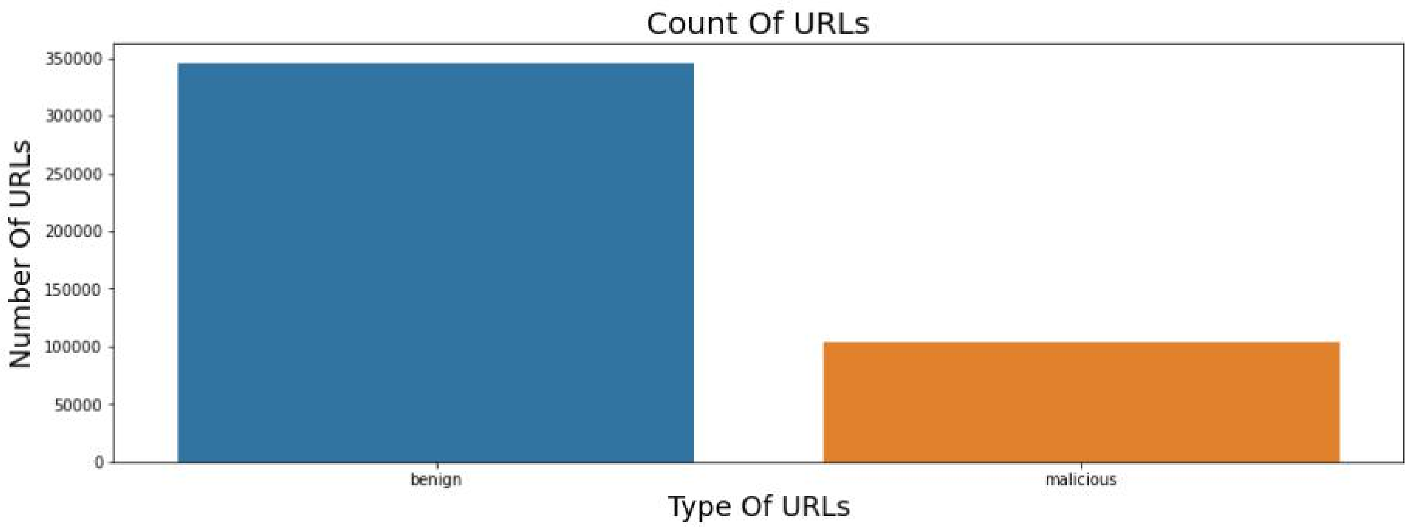
```
import re
#Use of IP or not in domain
def having_ip_address(url):
    match = re.search(
        '(([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\.([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\.([01]?\\d\\d?|2
[0-4]\\d|25[0-5])\\.'
        '([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\/)|'  # IPv4
        '((0x[0-9a-fA-F]{1,2})\\.(0x[0-9a-fA-F]{1,2})\\.(0x[0-9a-fA-F]{1,2})\\.(0x[0-9a-fA-F]
{1,2})\\/)' # IPv4 in hexadecimal
        '(?:[a-fA-F0-9]{1,4}:){7}[a-fA-F0-9]{1,4}', url)  # Ipv6
    if match:
        # print match.group()
        return 1
    else:
        # print 'No matching pattern found'
        return 0
df['use_of_ip'] = df['url'].apply(lambda i: having_ip_address(i))
```



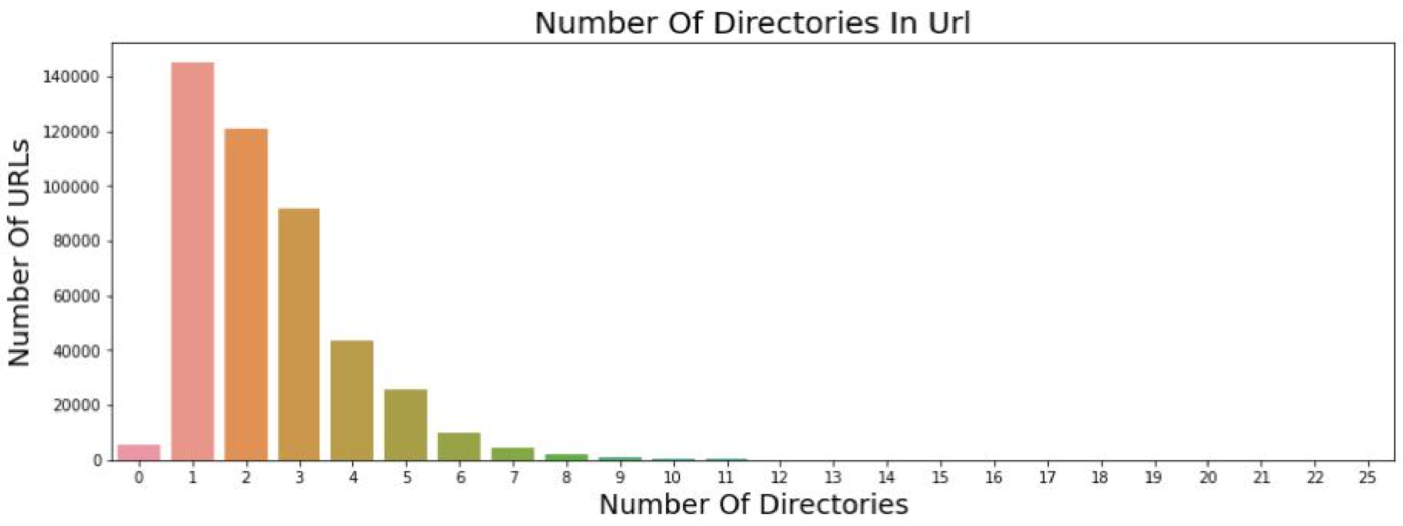
```
from urllib.parse import urlparse



def abnormal_url(url):
    hostname = urlparse(url).hostname
    hostname = str(hostname)
    match = re.search(hostname, url)
    if match:
        # print match.group()
        return 1
    else:
        # print 'No matching pattern found'
        return 0


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

Data Visualization

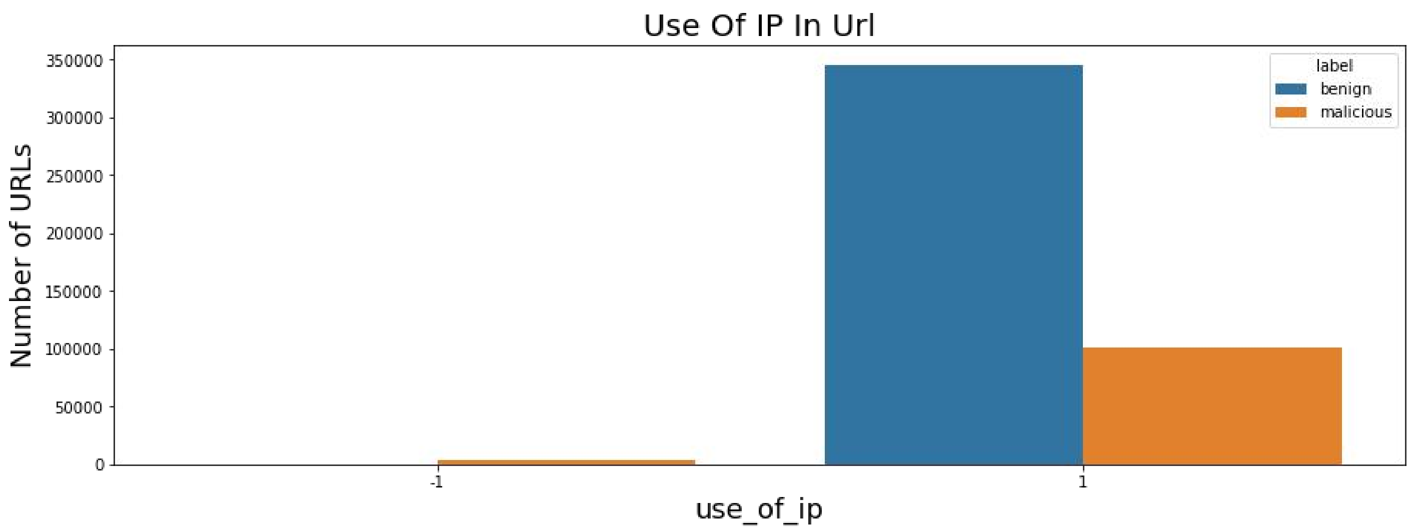
```
plt.figure(figsize=(15,5))
sns.countplot(x='label',data=urldata)
plt.title("Count Of URLs",fontsize=20)
plt.xlabel("Type Of URLs",fontsize=18)
plt.ylabel("Number Of URLs",fontsize=18)
```



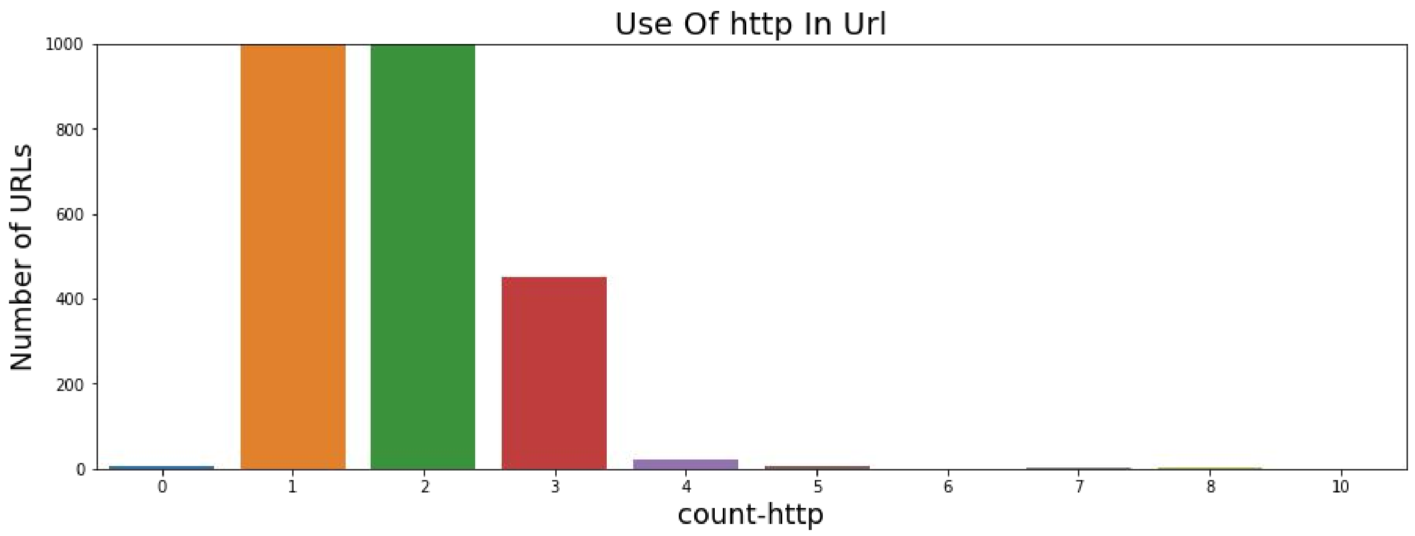
```
plt.figure(figsize=(15,5))
plt.title("Use Of IP In Url",fontsize=20)
plt.xlabel("Use Of IP",fontsize=18)

sns.countplot(urldata['use_of_ip'])
plt.ylabel("Number of URLs",fontsize=18)
```



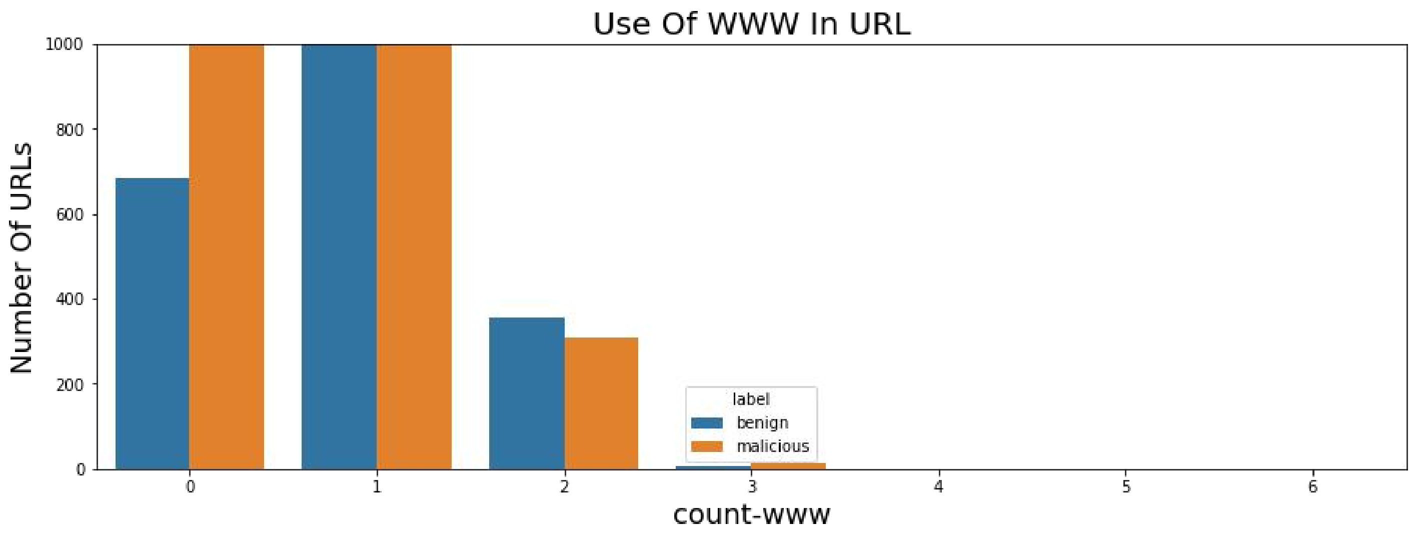
```
plt.figure(figsize=(15,5))
plt.title("Use Of IP In Url",fontsize=20)
plt.xlabel("Use Of IP",fontsize=18)
plt.ylabel("Number of URLs",fontsize=18)
sns.countplot(urldata['use_of_ip'],hue='label',data=urldata)
plt.ylabel("Number of URLs",fontsize=18)
```



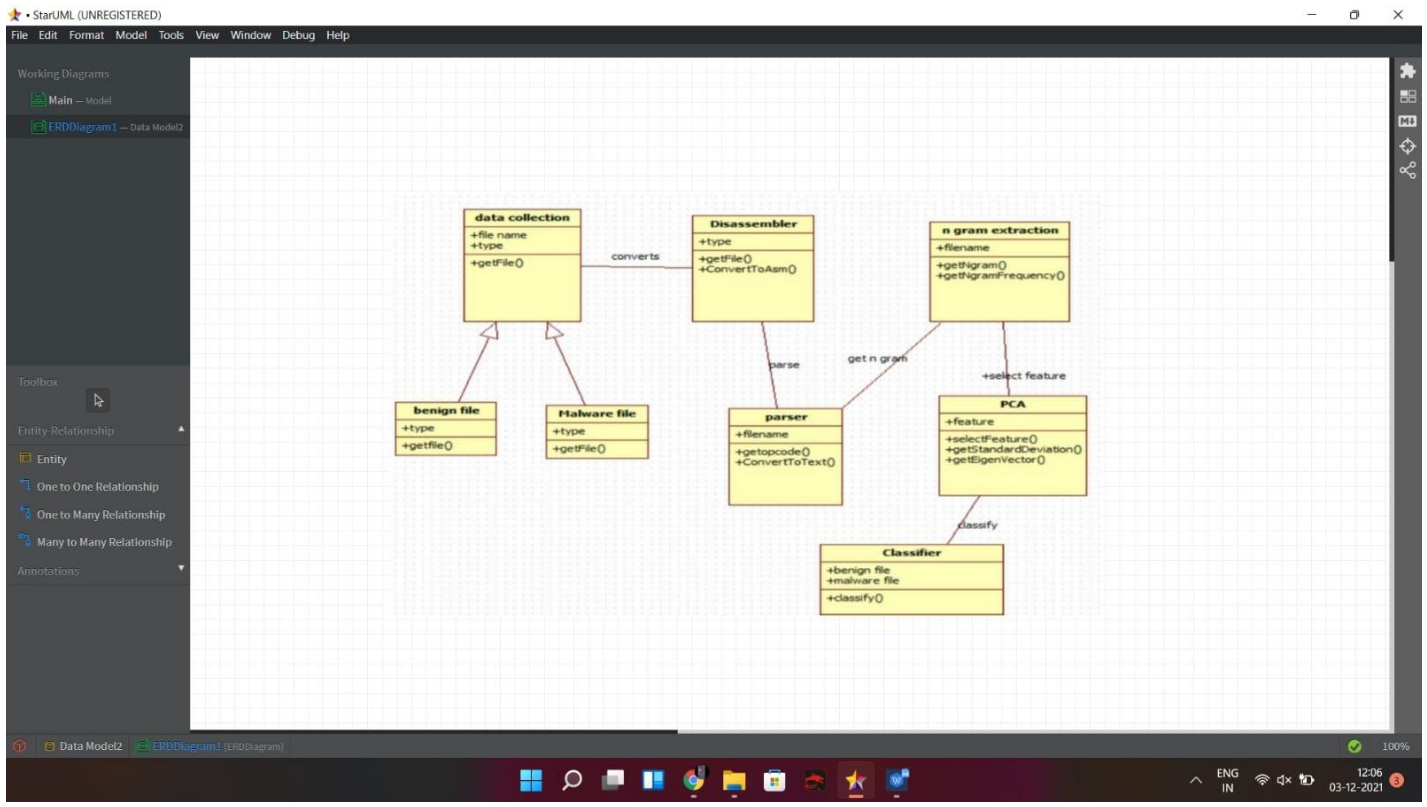
```
plt.figure(figsize=(15,5))
plt.title("Use Of http In Url",fontsize=20)
plt.xlabel("Use Of IP",fontsize=18)
plt.ylim((0,1000))
sns.countplot(urldata['count-http'])
plt.ylabel("Number of URLs",fontsize=18)
```

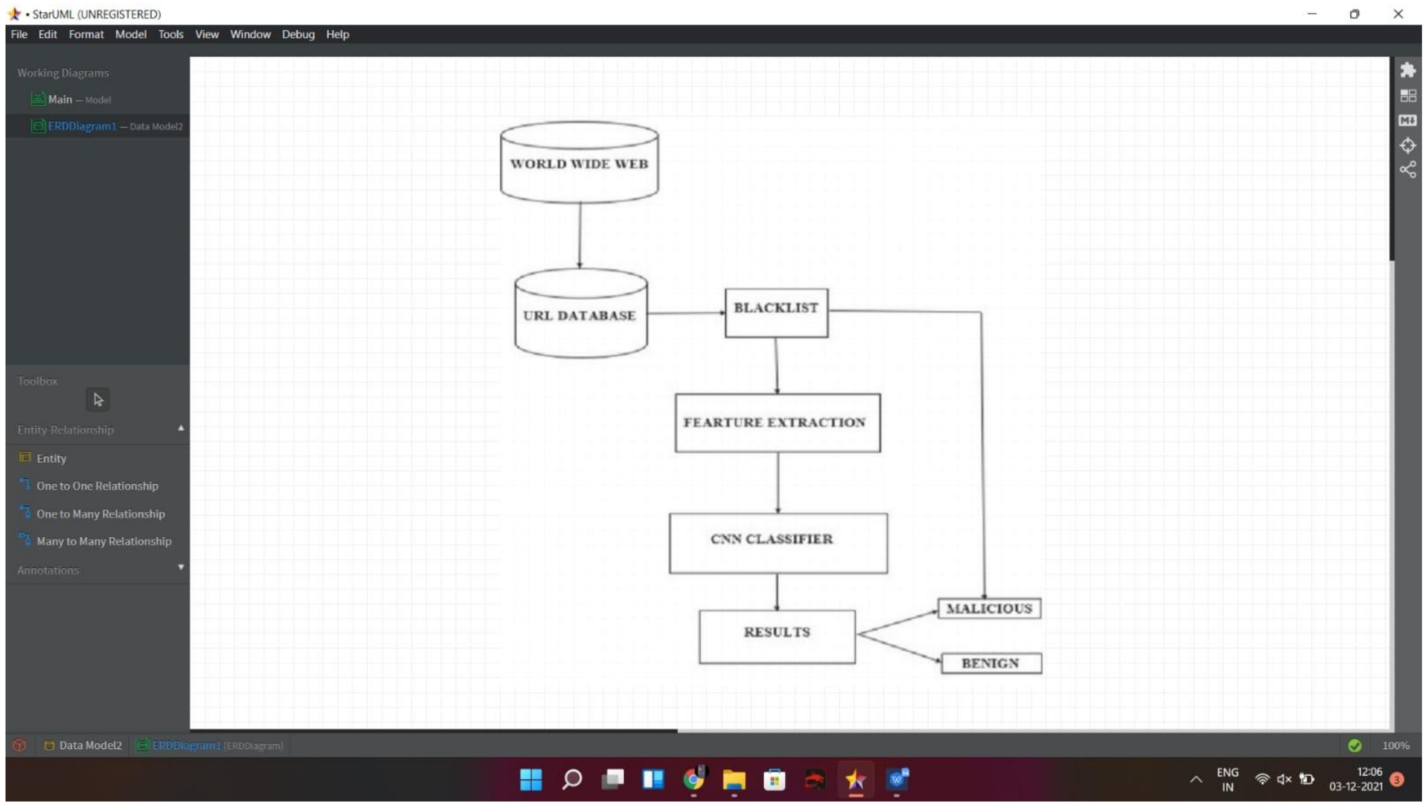


```
plt.figure(figsize=(15,5))
plt.title("Use Of WWW In URL",fontsize=20)
plt.xlabel("Count Of WWW",fontsize=18)

sns.countplot(urldata['count-www'],hue='label',data=urldata)
plt.ylim(0,1000)
plt.ylabel("Number Of URLs",fontsize=18)
```



UML Diagram:-


StarUML (UNREGISTERED)
File Edit Format Model Tools View Window Debug Help
Working Diagrams
Main — Model
ERDDiagram1 — Data Model2
Toolbox
Entity-Relationship
Entity
One to One Relationship
One to Many Relationship
Many to Many Relationship
Annotations
data collection
+file name
+type
+getFile()
converts
Disassembler
+type
+getFile()
+ConvertToAsm()
n gram extraction
+filename
+getNgram()
+getNgramFrequency()
benign file
+type
+getfile()
Malware file
+type
+getFile()
parse
get n gram
+select feature
parser
+filename
+getopcode()
+ConvertToText()
PCA
+feature
+selectFeature()
+getStandardDeviation()
+getEigenVector()
classify
Classifier
+benign file
+malware file
+classify()
Data Model2
ERDDiagram1 [ERDDiagram]
100%
ENG IN
12:06
03-12-2021


StarUML (UNREGISTERED)
File Edit Format Model Tools View Window Debug Help
Working Diagrams
Main — Model
ERDDiagram1 — Data Model2
Toolbox
Entity-Relationship
Entity
One to One Relationship
One to Many Relationship
Many to Many Relationship
Annotations
WORLD WIDE WEB
URL DATABASE
BLACKLIST
FEARTURE EXTRACTION
CNN CLASSIFIER
RESULTS
MALICIOUS
BENIGN
Data Model2
ERDDiagram1 [ERDDiagram]
100%
ENG IN
12:06
03-12-2021

4. Anomaly Detection - Credit Card Fraud Analysis

Anomaly detection is a technique that is used to identify unusual or strange patterns that do not match expected behaviour, preferably known as outliers. It has quite many uses in businesses, from intrusion detection (detecting strange , unusual patterns in network traffic which could indicate a hack) to monitoring system health (detecting a malignancy on an MRI) and from detecting fraud in credit card transactions to the detection of errors in the operation environments.

Anomaly Detection Techniques

1-Simple statistic method:- The most basic method for detecting data irregularities is to mark data points that differ from typical statistical features of a distribution, such as mean, median, mode, and quantiles.
The challenges of this method could be Because the line between normal and abnormal conduct is often blurry, the data contains noise that could be mistaken for abnormal behaviour. The exact definition of abnormal or normal may frequently change, as the malicious adversaries alter regularly. As a result, the moving average barrier may not always be applicable

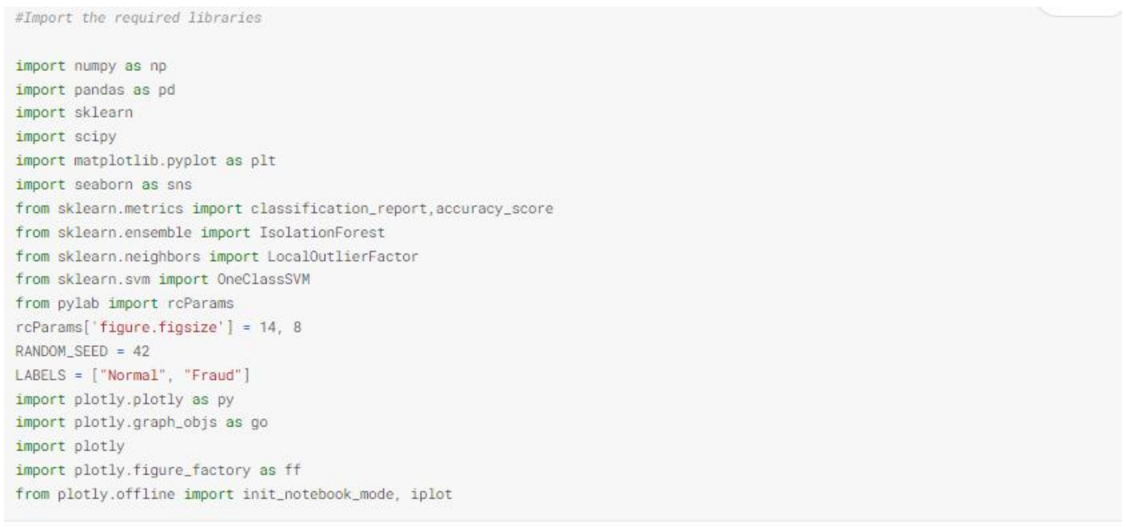
2-Machine Learning approach

2.1. Density-based anomaly detection is based on the k-nearest neighbours algorithm.
Assumption: Normal data points occur around a dense neighbourhood and abnormalities are far away.

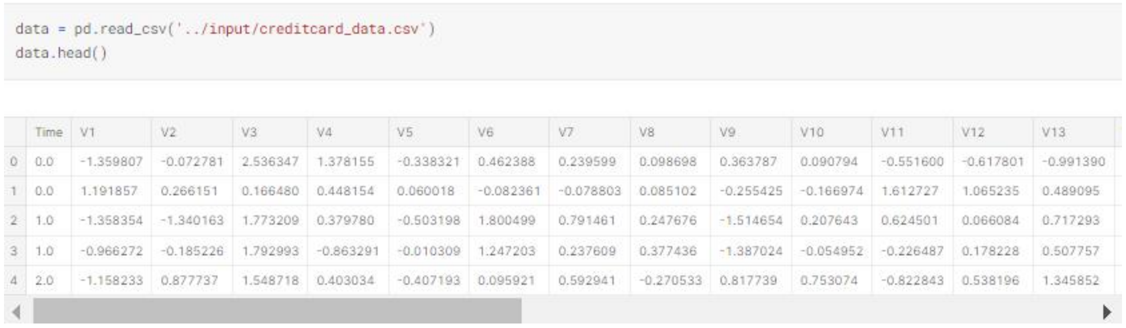
2.2. Clustering-Based Anomaly Detection Clustering is one of the most popular concepts in the domain of unsupervised learning.
Assumption: Data points that are similar tend to belong to similar groups or clusters, as determined by their distance from local centroids.

2.3.Support Vector Machine-Based Anomaly Detection, a support vector machine is another successful tool for finding anomalies. Although supervised learning is the most common application of an SVM, there are variants (such as OneClassCVM) that may be used to discover anomalies as unsupervised problems .

Isolation Forest Anomaly Detection Algorithm Density-Based Anomaly Detection (Local Outlier Factor)Algorithm Support Vector Machine Anomaly Detection Algorithm Credit Card Fraud Detection Problem Statement: The Credit Card Fraud Detection Problem includes modeling past credit card transactions with the knowledge of the ones that turned out to be fraud. This model is then used to identify whether a new transaction is fraudulent or not. Our aim here is to detect 100 % of the fraudulent transactions while minimizing the incorrect fraud classifications.

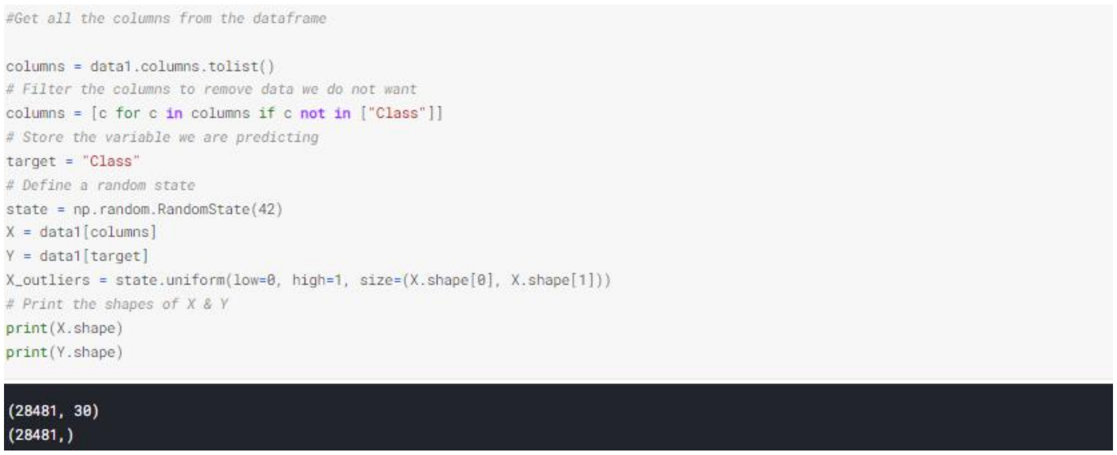


```
#Import the required libraries

import numpy as np
import pandas as pd
import sklearn
import scipy
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import classification_report,accuracy_score
from sklearn.ensemble import IsolationForest
from sklearn.neighbors import LocalOutlierFactor
from sklearn.svm import OneClassSVM
from pylab import rcParams
rcParams['figure.figsize'] = 14, 8
RANDOM_SEED = 42
LABELS = ["Normal", "Fraud"]
import plotly.plotly as py
import plotly.graph_objs as go
import plotly
import plotly.figure_factory as ff
from plotly.offline import init_notebook_mode, iplot
```



```
data = pd.read_csv('../input/creditcard_data.csv')
data.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.090794	-0.551600	-0.617801	-0.991390
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974	1.612727	1.065235	0.489095
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207643	0.624501	0.066084	0.717293
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.054952	-0.226487	0.178228	0.507757
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074	-0.822843	0.538196	1.345852



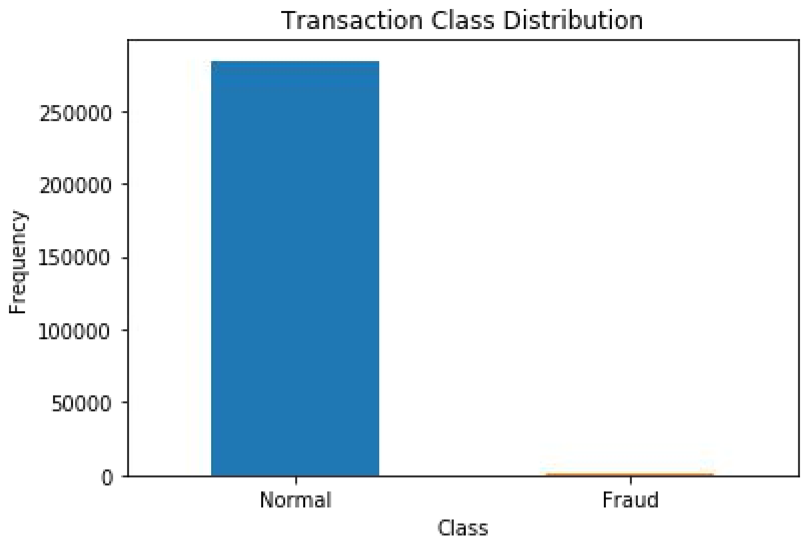
```
#Get all the columns from the dataframe

columns = data1.columns.tolist()
# Filter the columns to remove data we do not want
columns = [c for c in columns if c not in ["Class"]]
# Store the variable we are predicting
target = "Class"
# Define a random state
state = np.random.RandomState(42)
X = data1[columns]
Y = data1[target]
X_outliers = state.uniform(low=0, high=1, size=(X.shape[0], X.shape[1]))
# Print the shapes of X & Y
print(X.shape)
print(Y.shape)
```

```
(28481, 30)
(28481,)
```

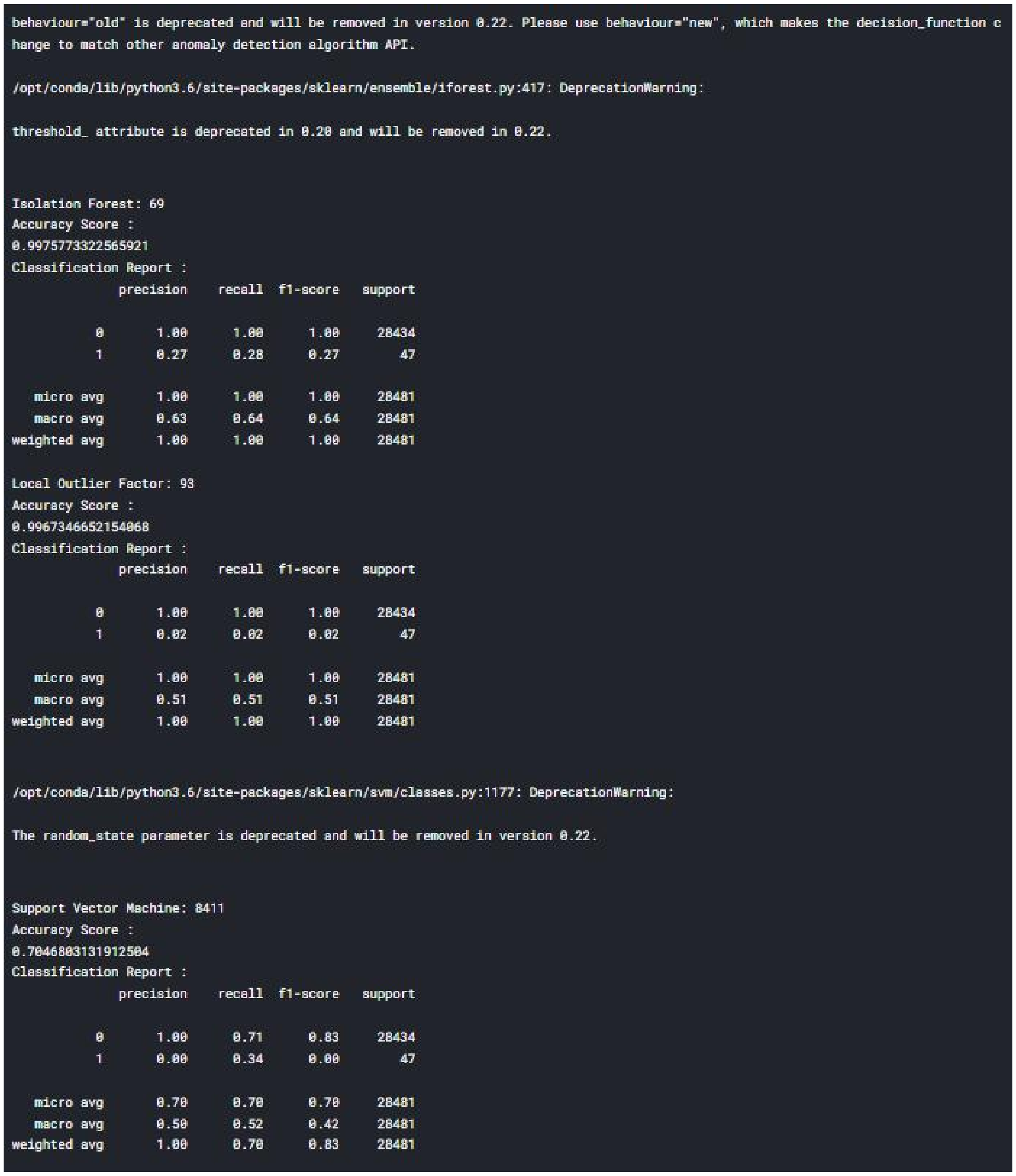
#Determine the number of fraud and valid transactions in the entire dataset

count_classes = pd.value_counts(data['Class'], sort = True)

```
count_classes.plot(kind = 'bar', rot=0)
plt.title("Transaction Class Distribution")
plt.xticks(range(2), LABELS)
plt.xlabel("Class")
plt.ylabel("Frequency");
```



```
n_outliers = len(Fraud)
for i, (clf_name,clf) in enumerate(classifiers.items()):
    #Fit the data and tag outliers
    if clf_name == "Local Outlier Factor":
        y_pred = clf.fit_predict(X)
        scores_prediction = clf.negative_outlier_factor_
    elif clf_name == "Support Vector Machine":
        clf.fit(X)
        y_pred = clf.predict(X)
    else:
        clf.fit(X)
        scores_prediction = clf.decision_function(X)
        y_pred = clf.predict(X)
    #Reshape the prediction values to 0 for Valid transactions , 1 for Fraud transactions
    y_pred[y_pred == 1] = 0
    y_pred[y_pred == -1] = 1
    n_errors = (y_pred != Y).sum()
    # Run Classification Metrics
    print("{}: {}".format(clf_name,n_errors))
    print("Accuracy Score :")
    print(accuracy_score(Y,y_pred))
print("Classification Report :")
print(classification_report(Y,y_pred))
```

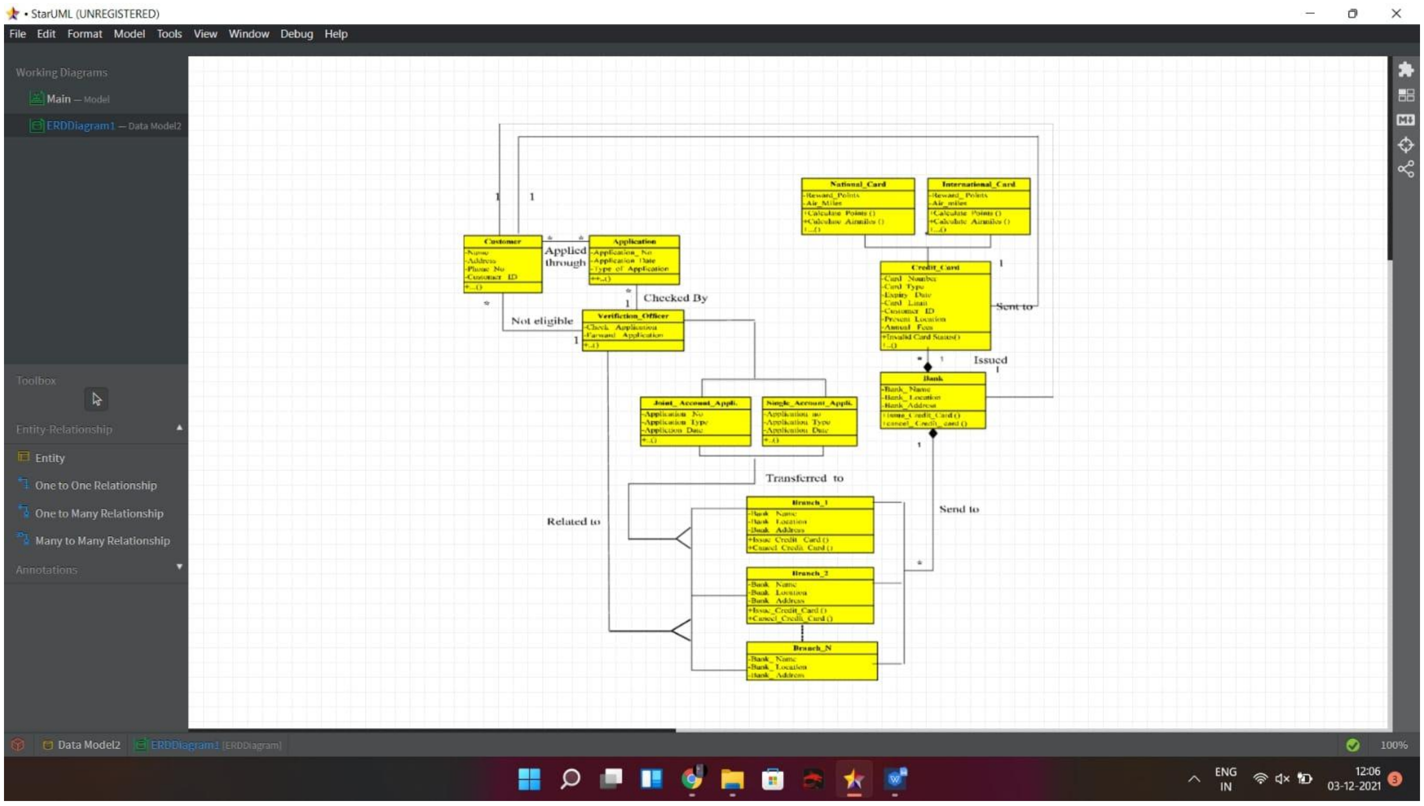


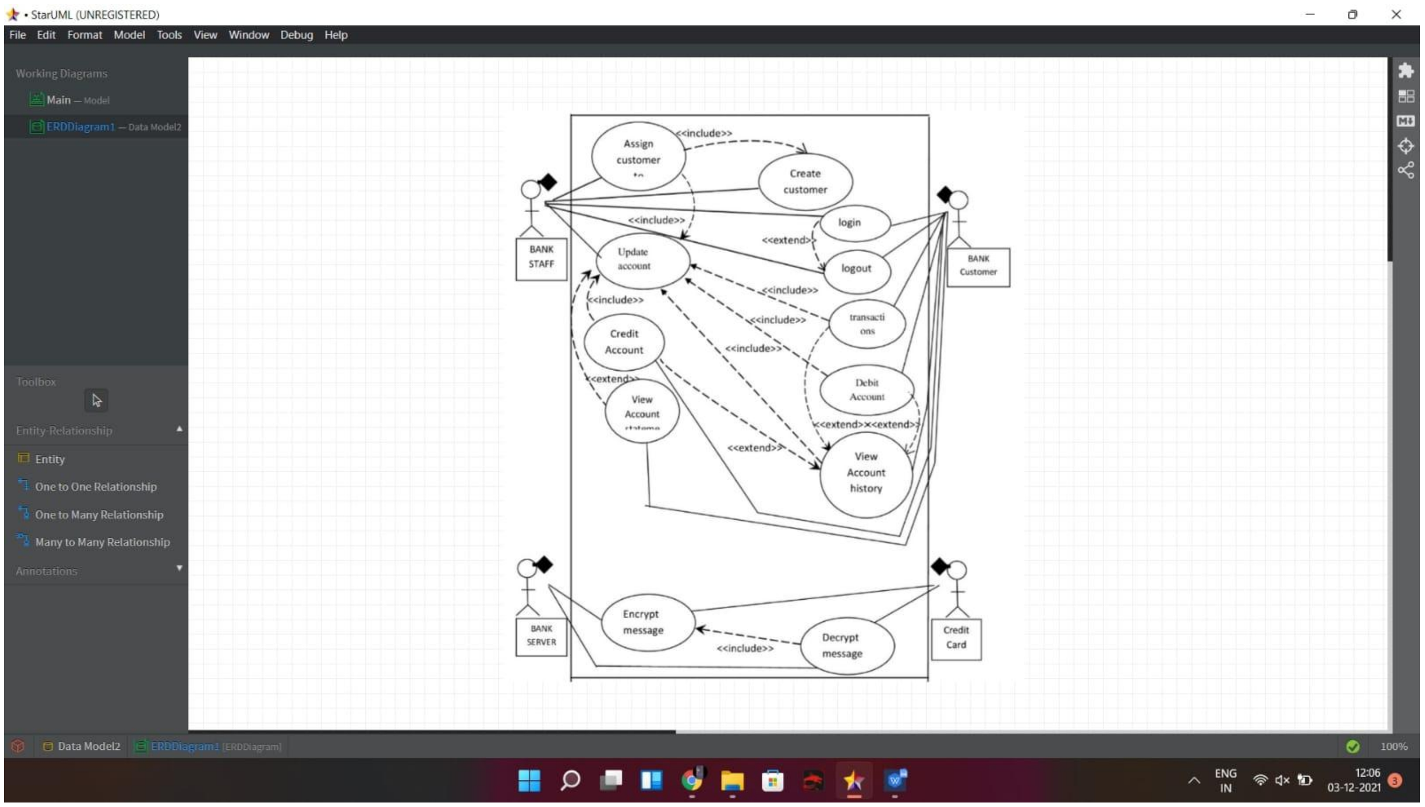
```
behaviour="old" is deprecated and will be removed in version 0.22. Please use behaviour="new", which makes the decision_function c
hange to match other anomaly detection algorithm API.

/opt/conda/lib/python3.6/site-packages/sklearn/ensemble/iforest.py:417: DeprecationWarning:

threshold_ attribute is deprecated in 0.20 and will be removed in 0.22.


Isolation Forest: 69
Accuracy Score :
0.9975773322565921
Classification Report :
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     28434
           1       0.27      0.28      0.27        47

   micro avg       1.00      1.00      1.00     28481
   macro avg       0.63      0.64      0.64     28481
weighted avg       1.00      1.00      1.00     28481

Local Outlier Factor: 93
Accuracy Score :
0.9967346652154068
Classification Report :
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     28434
           1       0.02      0.02      0.02        47

   micro avg       1.00      1.00      1.00     28481
   macro avg       0.51      0.51      0.51     28481
weighted avg       1.00      1.00      1.00     28481


/opt/conda/lib/python3.6/site-packages/sklearn/svm/classes.py:1177: DeprecationWarning:

The random_state parameter is deprecated and will be removed in version 0.22.


Support Vector Machine: 8411
Accuracy Score :
0.7046803131912504
Classification Report :
              precision    recall  f1-score   support

           0       1.00      0.71      0.83     28434
           1       0.00      0.34      0.00        47

   micro avg       0.70      0.70      0.70     28481
   macro avg       0.50      0.52      0.42     28481
weighted avg       1.00      0.70      0.83     28481
```

Observations :

- Isolation Forest detected 69 errors versus Local Outlier Factor detecting 93 errors vs. SVM detecting 8411 errors
- Isolation Forest has a 99.75% more accurate than LOF of 99.67% and SVM of 70.46
- When comparing error precision & recall for 3 models , the Isolation Forest performed much better than the LOF as we can see that the detection of fraud cases is around 27 % versus LOF detection rate of just 2 % and SVM of 0
- So overall Isolation Forest Method performed much better in determining the fraud cases which is around 30%.
- We can also improve on this accuracy by increasing the sample size or use deep learning algorithms however at the cost of computational expense.We can also use complex anomaly detection models to get better accuracy in determining more fraudulent cases

Uml Diagram

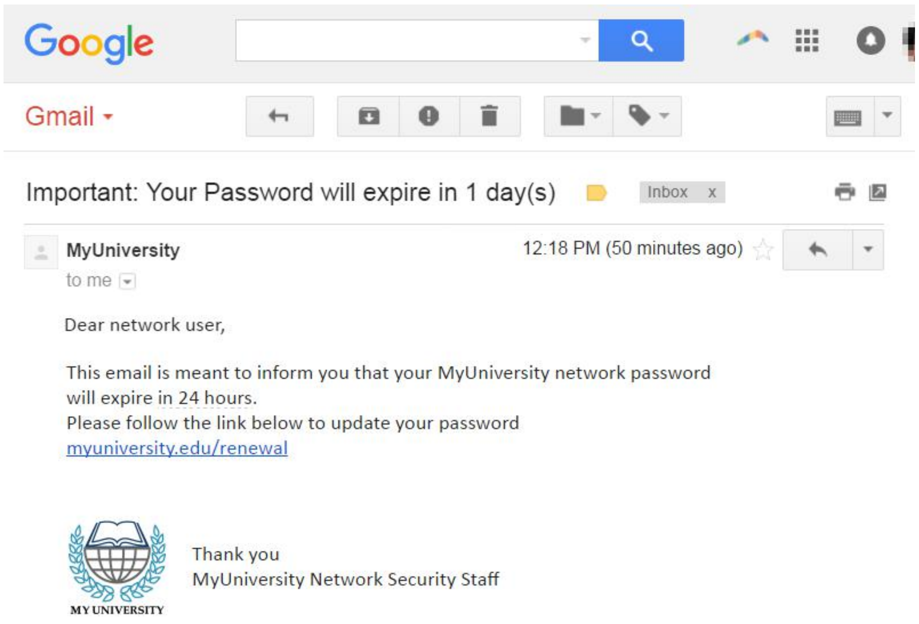



StarUML (UNREGISTERED)
File Edit Format Model Tools View Window Debug Help
Working Diagrams
Main — Model
ERDDiagram1 — Data Model2
Toolbox
Entity-Relationship
Entity
One to One Relationship
One to Many Relationship
Many to Many Relationship
Annotations
BANK STAFF
BANK Customer
BANK SERVER
Credit Card
Assign customer to
Create customer
login
logout
Update account
transactions
Credit Account
Debit Account
View Account stateme
View Account history
Encrypt message
Decrypt message
<<include>>
<<extend>>
Data Model2
ERDDiagram1 [ERDDiagram]
100%
ENG IN
12:06
03-12-2021

5. Phishing Sites Detector & Complete info

Phishing is a type of social engineering assault or attack that is frequently used to obtain sensitive information from users, such as login credentials and credit card details. It happens when a hacker poses as a trustworthy entity and convinces a victim to open an email, instant message, or text message.

A mass email, purporting to be from myuniversity.edu, is sent to as many faculty members as possible. The user's password is due to expire, according to the email. In order to renew their password, they must go to myuniversity.edu/renewal within 24 hours.



Important: Your Password will expire in 1 day(s)

MyUniversity
to me

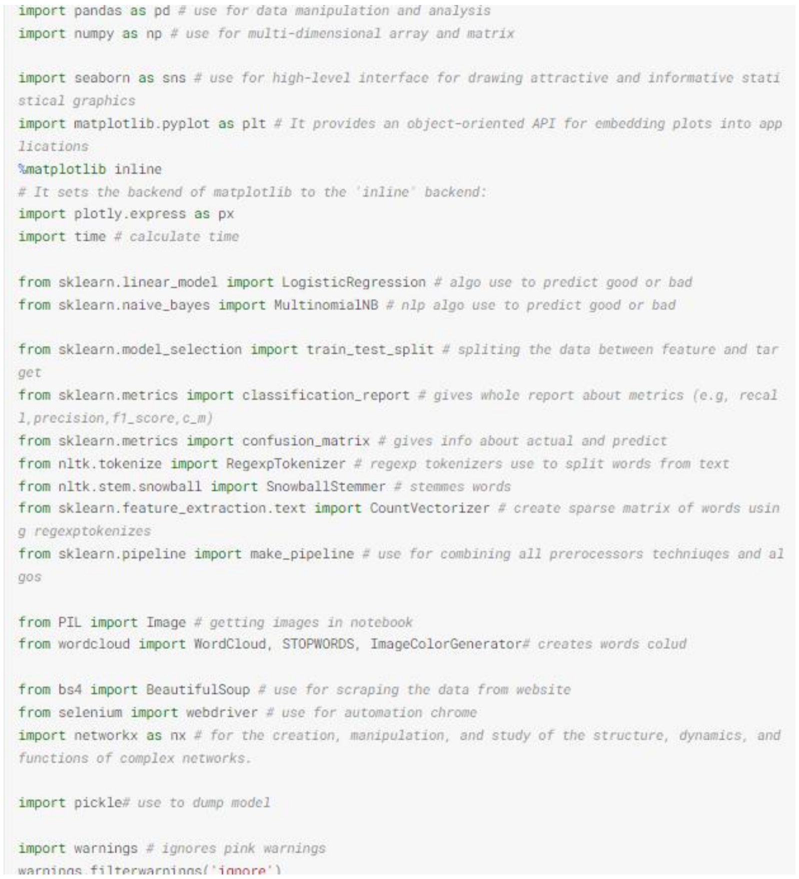
12:18 PM (50 minutes ago)

Dear network user,

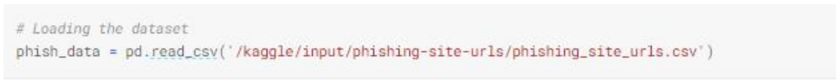
This email is meant to inform you that your MyUniversity network password will expire in 24 hours.
Please follow the link below to update your password
myuniversity.edu/renewal

Thank you
MyUniversity Network Security Staff

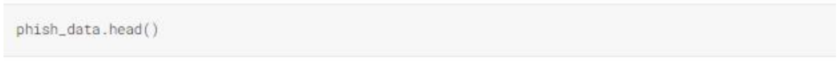
Several Things can occur by clicking the link-
1. The user is sent to myuniversity.edurenewal.com, a fake page that looks identical to the real renewal page and asks for both new and old passwords. While watching the page, the attacker steals the original password and uses it to gain access to restricted sections of the university network.
2. The user is forwarded to the password renewal page. While being routed, however, a malicious script runs in the background, stealing the user's session cookie. As a result of the mirrored XSS attack, the attacker now has privileged access to the university network.



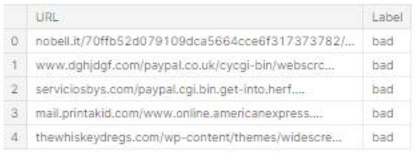
```
import pandas as pd # use for data manipulation and analysis
import numpy as np # use for multi-dimensional array and matrix

import seaborn as sns # use for high-level interface for drawing attractive and informative stati
stical graphics
import matplotlib.pyplot as plt # It provides an object-oriented API for embedding plots into app
lications
%matplotlib inline
# It sets the backend of matplotlib to the 'inline' backend:
import plotly.express as px
import time # calculate time

from sklearn.linear_model import LogisticRegression # algo use to predict good or bad
from sklearn.naive_bayes import MultinomialNB # nlp algo use to predict good or bad

from sklearn.model_selection import train_test_split # spliting the data between feature and tar
get
from sklearn.metrics import classification_report # gives whole report about metrics (e.g, recal
l,precision,f1_score,c_m)
from sklearn.metrics import confusion_matrix # gives info about actual and predict
from nltk.tokenize import RegexpTokenizer # regexp tokenizers use to split words from text
from nltk.stem.snowball import SnowballStemmer # stemmes words
from sklearn.feature_extraction.text import CountVectorizer # create sparse matrix of words usin
g regexptokenizes
from sklearn.pipeline import make_pipeline # use for combining all prerocessors techniuqes and al
gos

from PIL import Image # getting images in notebook
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator# creates words colud

from bs4 import BeautifulSoup # use for scraping the data from website
from selenium import webdriver # use for automation chrome
import networkx as nx # for the creation, manipulation, and study of the structure, dynamics, and
functions of complex networks.

import pickle# use to dump model

import warnings # ignores pink warnings
warnings.filterwarnings('ignore')
```



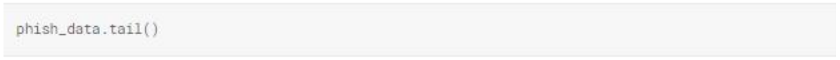
```
# Loading the dataset
phish_data = pd.read_csv('/kaggle/input/phishing-site-urls/phishing_site_urls.csv')
```



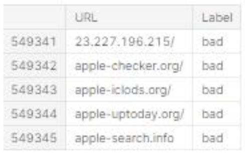
```
phish_data.head()
```



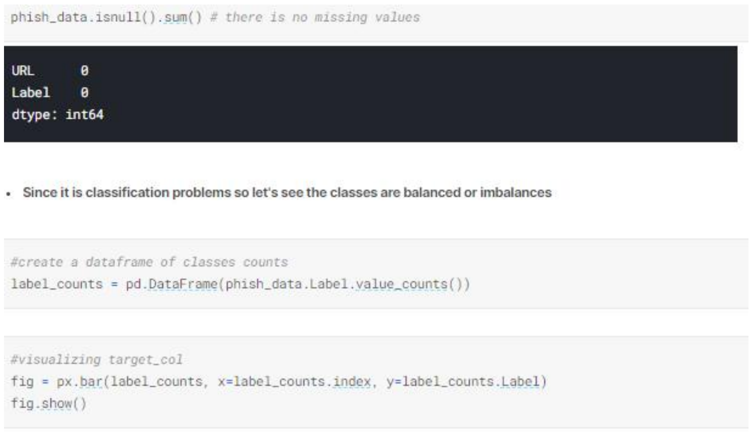
	URL	Label
0	nobell.it/70ffb52d079109dca5664cce6f317373782/...	bad
1	www.dghjdgf.com/paypal.co.uk/cycgi-bin/webscrc...	bad
2	serviciosbys.com/paypal.cgi.bin.get-into.herf....	bad
3	mail.printakid.com/www.online.americanexpress....	bad
4	thewhiskeydregs.com/wp-content/themes/widescre...	bad



```
phish_data.tail()
```



	URL	Label
549341	23.227.196.215/	bad
549342	apple-checker.org/	bad
549343	apple-iclods.org/	bad
549344	apple-uptoday.org/	bad
549345	apple-search.info	bad



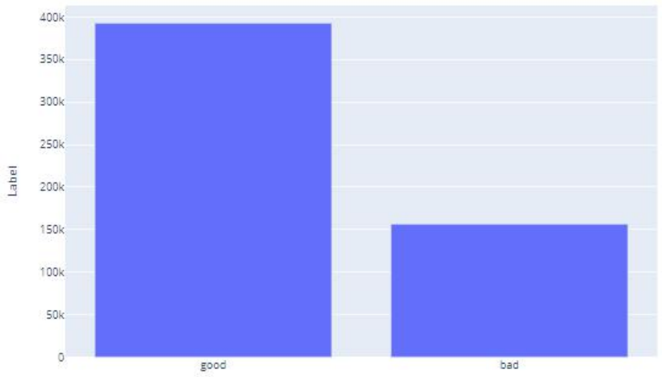
```
phish_data.isnull().sum() # there is no missing values
```

```
URL      0
Label    0
dtype: int64
```

- Since it is classification problems so let's see the classes are balanced or imbalances

```
#create a dataframe of classes counts
label_counts = pd.DataFrame(phish_data.Label.value_counts())
```

```
#visualizing target_col
fig = px.bar(label_counts, x=label_counts.index, y=label_counts.Label)
fig.show()
```



```
def plot_wordcloud(text, mask=None, max_words=400, max_font_size=120, figure_size=(24.0,16.0),
                   title = None, title_size=40, image_color=False):
    stopwords = set(STOPWORDS)
    more_stopwords = {'com','http'}
    stopwords = stopwords.union(more_stopwords)

    wordcloud = WordCloud(background_color='white',
                    stopwords = stopwords,
                    max_words = max_words,
                    max_font_size = max_font_size,
                    random_state = 42,
                    mask = mask)
    wordcloud.generate(text)

    plt.figure(figsize=figure_size)
    if image_color:
        image_colors = ImageColorGenerator(mask);
        plt.imshow(wordcloud.recolor(color_func=image_colors), interpolation="bilinear");
        plt.title(title, fontdict={'size': title_size,
                                  'verticalalignment': 'bottom'})
    else:
        plt.imshow(wordcloud);
        plt.title(title, fontdict={'size': title_size, 'color': 'green',
                                  'verticalalignment': 'bottom'})
    plt.axis('off');
    plt.tight_layout()
d = '../input/masks/masks-wordclouds/'
```

In [25]:

```
data = good_sites.text_sent
data.reset_index(drop=True, inplace=True)
```

In [26]:

linkcode

```
common_text = str(data)
common_mask = np.array(Image.open(d+'star.png'))
plot_wordcloud(common_text, common_mask, max_words=400, max_font_size=120,
               title = 'Most common words use in good urls', title_size=15)
```

Creating Model

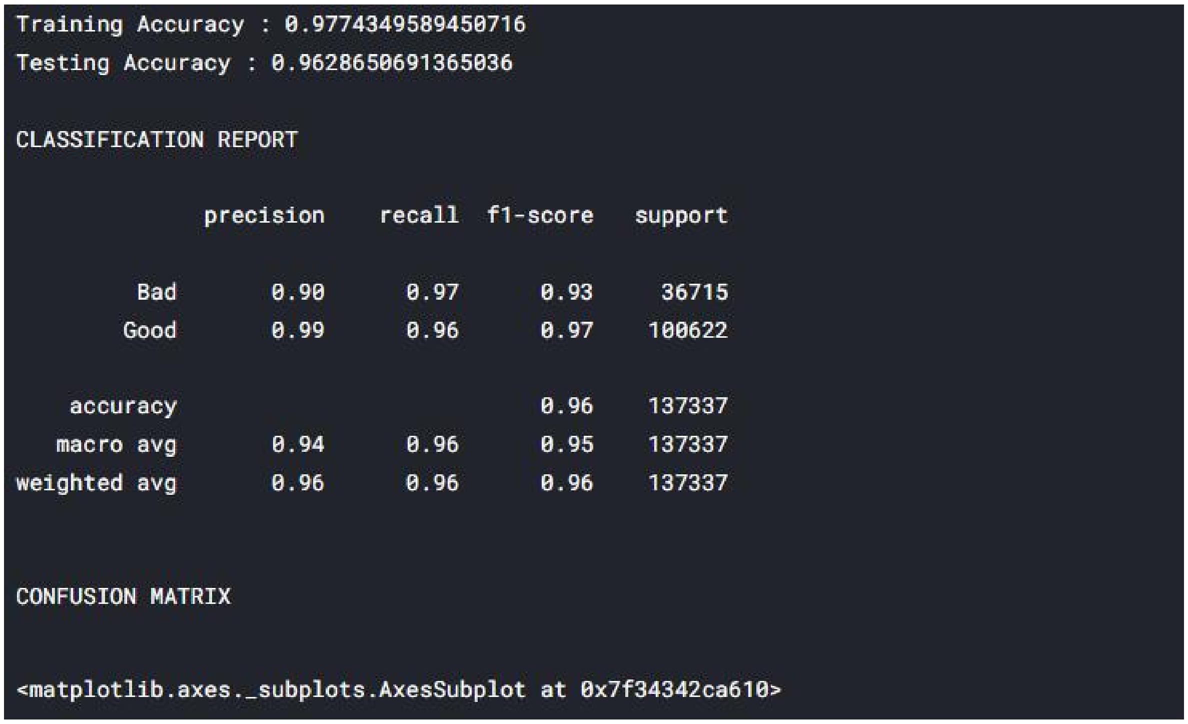
Logistic Regression

```
Scores_ml = {}
Scores_ml['Logistic Regression'] = np.round(lr.score(testX,testY),2)
```

In [38]:

linkcode

```
print('Training Accuracy :',lr.score(trainX,trainY))
print('Testing Accuracy :',lr.score(testX,testY))
con_mat = pd.DataFrame(confusion_matrix(lr.predict(testX), testY),
            columns = ['Predicted:Bad', 'Predicted:Good'],
            index = ['Actual:Bad', 'Actual:Good'])


print('\nCLASSIFICATION REPORT\n')
print(classification_report(lr.predict(testX), testY,
                            target_names =['Bad','Good']))

print('\nCONFUSION MATRIX')
plt.figure(figsize= (6,4))
sns.heatmap(con_mat, annot = True,fmt='d',cmap="YlGnBu")
```



```
Training Accuracy : 0.9774349589450716
Testing Accuracy : 0.9628650691365036

CLASSIFICATION REPORT

              precision    recall  f1-score   support

         Bad       0.90      0.97      0.93     36715
        Good       0.99      0.96      0.97    100622

    accuracy                           0.96    137337
   macro avg       0.94      0.96      0.95    137337
weighted avg       0.96      0.96      0.96    137337


CONFUSION MATRIX


<matplotlib.axes._subplots.AxesSubplot at 0x7f34342ca610>
```

- **About dataset**
- Data is containg 5,49,346 unique entries.
- There are two columns.
- Label column is prediction col which has 2 categories A. Good - which means the urls is not containing malicious stuff and **this site is not a Phishing Site.** B. Bad - which means the urls contains malicious stuffs and **this site isa Phishing Site.**
- There is no missing value in the dataset.

Preprocessing

- **N**ow that we have the data, we have to vectorize our URLs. I used CountVectorizer and gather words using tokenizer, since there are words in urls that are more important than other words e.g ‘virus’, ‘.exe’ ,’.dat’ etc. Lets convert the URLs into a vector form.

Protections

How to Protect Your Computer

Below are some key steps to protecting your computer from intrusion:

1. **Keep Your Firewall Turned On:** A firewall helps protect your computer from hackers who might try to gain access to crash it, delete information, or even steal passwords or other sensitive information. Software firewalls are widely recommended for single computers. The software is prepackaged on some operating systems or can be purchased for individual computers. For multiple networked computers, hardware routers typically provide firewall protection.

2. **Install or Update Your Antivirus Software:** Antivirus software is designed to prevent malicious software programs from embedding on your computer. If it detects malicious code, like a virus or a worm, it works to disarm or remove it. Viruses can infect computers without users' knowledge. Most types of antivirus software can be set up to update automatically.

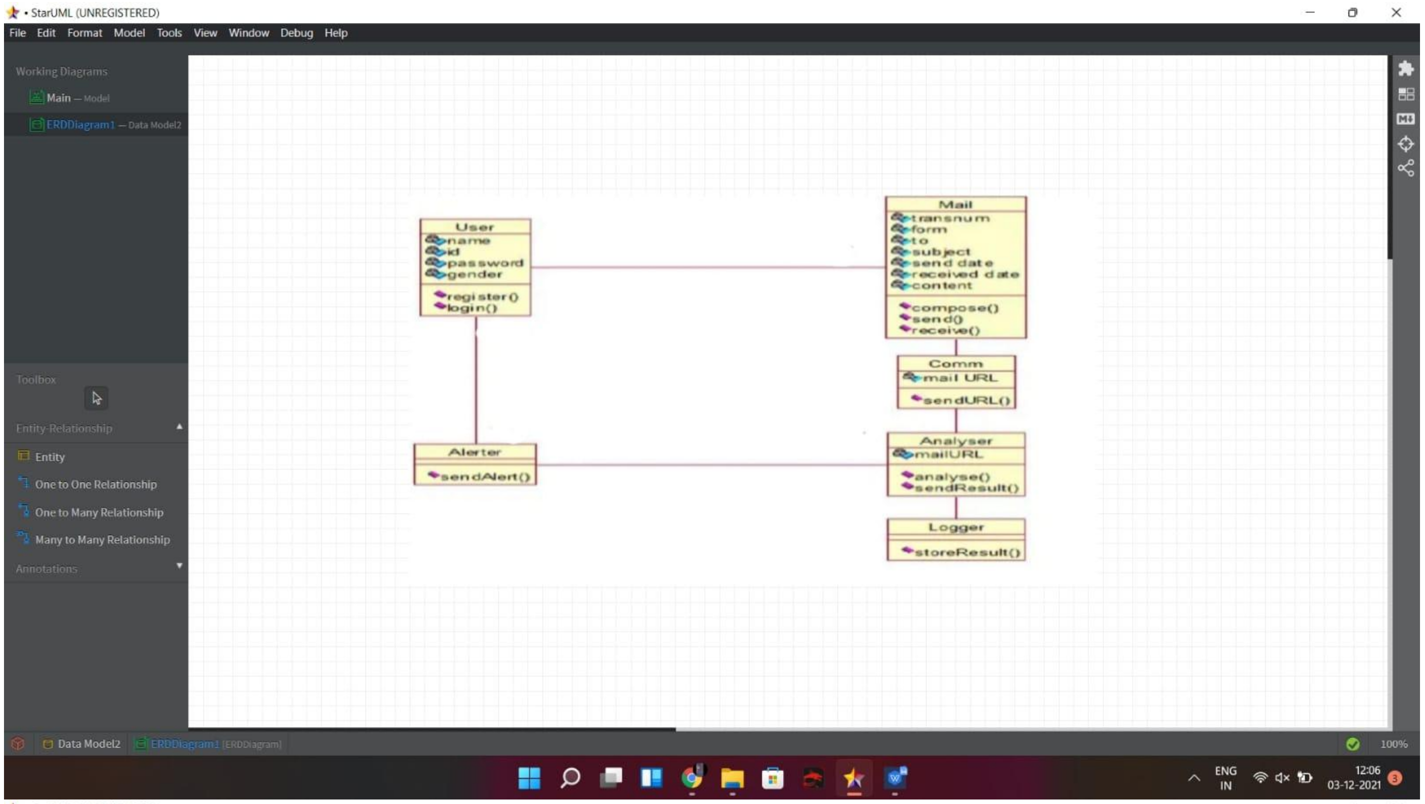
3. **Install or Update Your Antispyware Technology:** Spyware is just what it sounds like—software that is surreptitiously installed on your computer to let others peer into your activities on the computer. Some spyware collects information about you without your consent or produces unwanted pop-up ads on your web browser. Some operating systems offer free spyware protection, and inexpensive software is readily available for download on the Internet or at your local computer store. Be wary of ads on the Internet offering downloadable antispyware—in some cases these products may be fake and may actually contain spyware or other malicious code. It's like buying groceries—shop where you trust.

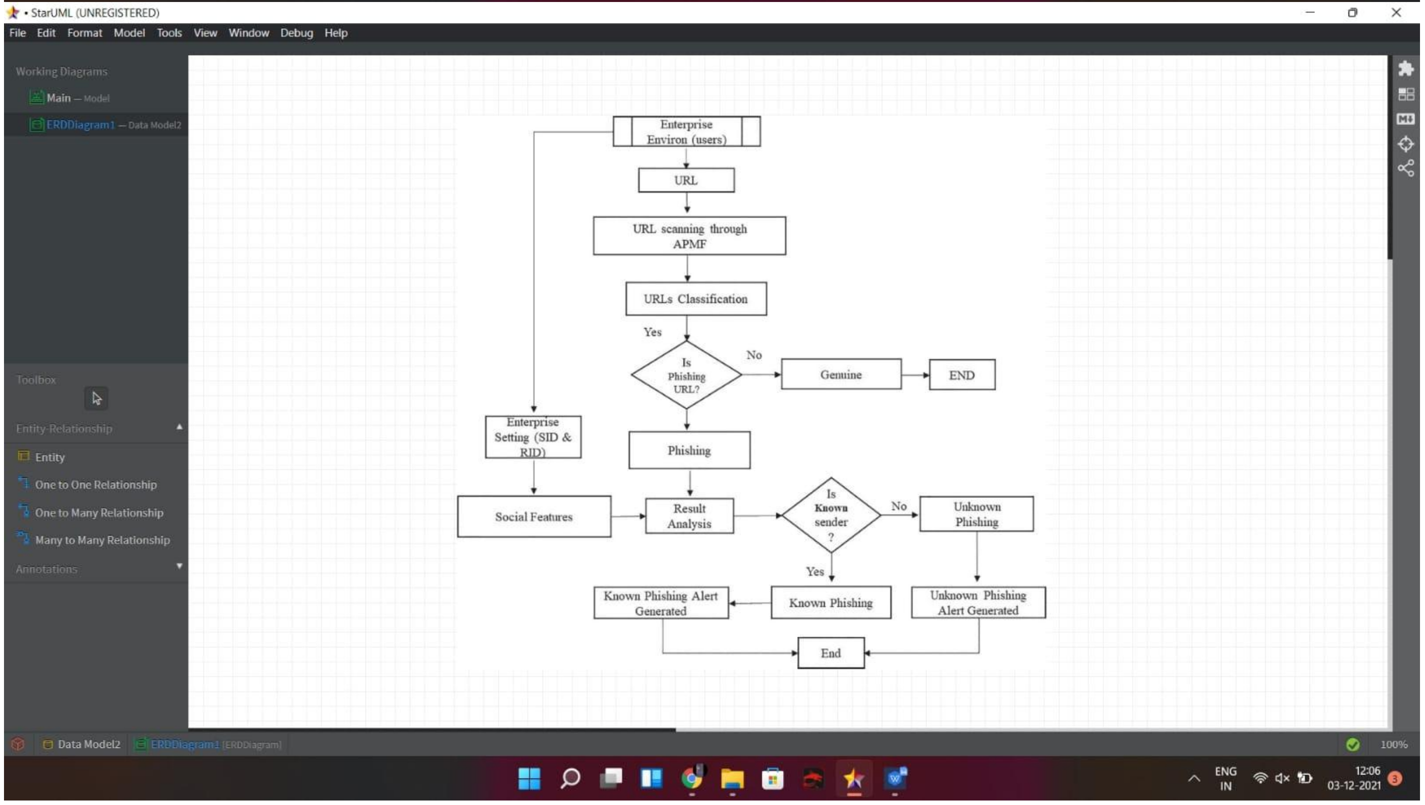
4. **Keep Your Operating System Up to Date:** Computer operating systems are periodically updated to stay in tune with technology requirements and to fix security holes. Be sure to install the updates to ensure your computer has the latest protection.

5. **Be Careful What You Download:** Carelessly downloading e-mail attachments can circumvent even the most vigilant anti-virus software. Never open an e-mail attachment from someone you don't know, and be wary of forwarded attachments from people you do know. They may have unwittingly advanced malicious code.

6. **Turn Off Your Computer:** With the growth of high-speed Internet connections, many opt to leave their computers on and ready for action. The downside is that being "always on" renders computers more susceptible. Beyond firewall protection, which is designed to fend off unwanted attacks, turning the computer off effectively severs an attacker's connection—be it spyware or a botnet that employs your computer's resources to reach out to other unwitting users.

Uml Diagram





Merits & Demerits Of Proposed Methods

<u>Merits</u>

I. These Methods provide a clear view of what's going on within your network. It is a valuable source of information about suspicious or malicious network traffic.
II. This methods adds a layer of defense to your security profile, providing a useful backstop to some of your other security measures.
III. Many Of these Methods can also identify or collect evidence regarding an activity, which may indicate how an attack is going to happen.
IV. When a virus first hits your network, an IDS can tell you which machines it compromised, as well as how it is propagating through the network to infect other machines. This can be a great help in slowing or stopping a virus's progress and making sure you remove it.
V. A properly configured IDS can produce data that can form the basis for a civil or criminal case against someone who misuses your network.

<u>Demerits</u>

I. It requires more maintenance.
II. Technologies are improving, but these methods don't always catch everything.
III. There is a need for Teams of cyber security experts.

Reference

1. https://ieeexplore.ieee.org/document/9214121
2. http://sersc.org/journals/index.php/IJAST/article/view/14826
3. https://thesai.org/Downloads/Volume10No2/Paper_26-A_Survey_on_Techniques_to_Detect_Malicious_Activities.pdf
4. http://nms.lcs.mit.edu/papers/thesis-final.pdf
5. https://thesai.org/Downloads/Volume11No1/Paper_19-Malicious_URL_Detection_based_on_Machine_Learning.pdf
6. http://ijniet.org/wp-content/uploads/2015/07/2.pdf
7.