A

PROJECT SEMINAR REPORT

ON

A ML FRAMEWORK FOR DGA-MALWARE DETECTION

Submitted in partial fulfilment for the Award of Degree of

BACHELOR OF ENGINEERING

IN

INFORMATION TECHNOLOGY

BY

K. GAGAN KUMAR (160117737092)

&

G. PRASHANTH (160117737101)

Under the guidance of

Ms.R. DEEPA,

Asst. Professor, IT Dept.

DEPARTMENT OF INFORMATION TECHNOLOGY

CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY (A)

(Affiliated to Osmania University; Accredited by NBA (AICTE) and NAAC (UGC), ISO Certified 9001:2015), Kokapet (V), GANDIPET(M), HYDERABAD – 500 075

Website: www.cbit.ac.in

2020-2021

CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY (A)
Kokapet(Village), Gandipet, Hyderabad, Telangana-500075. www.cbit.ac.in
ISO 9001:2015

COMMITTED TO RESEARCH, INNOVATION AND EDUCATION
42 years

CERTIFICATE

This is to certify that the project seminar report entitled “**A ML FRAMEWORK FOR DGA - MALWARE DETECTION**” submitted to **Chaitanya Bharathi Institute of Technology**, in partial fulfilment of the requirements for the award of degree of **B.E (Information Technology)** during the academic year 2020-21 is a record of original work carried out by **K.GAGAN KUMAR(160117737092)** and **G.PRASHANTH(160117737101)** during the period of study in the Dept. of IT, CBIT, Hyderabad.

Project Guide	Head of the Department
Ms. R. DEEPA.	**Dr. K. Radhika.**
Asst. Professor, IT Dept.	Professor, IT Dept,
CBIT, Hyderabad.	CBIT, Hyderabad.

DECLARATION

I declare that the project report entitled "**A ML FRAMEWORK FOR DGA-MALWARE DETECTION**" is being submitted by me in the Department of Information Technology, Chaitanya Bharathi Institute of Technology (A), Osmania University.

This is record of bonafide work carried out by me under the guidance and supervision of **Ms. R. Deepa**, Assistant Professor, Dept. of IT, C.B.I.T.

No part of the work is copied from books/journals/internet and wherever the portion is taken, the same has been duly referred to in the text. The reports are based on the project work done entirely by me and not copied from any other source.

K.Gagan Kumar (160117737092)

G.Prashanth (160117737101)

ACKNOWLEDGEMENTS

It is my privilege to acknowledge with deep sense of gratitude and devotion for keen personal interest and invaluable guidance rendered by our Project Guide **Ms. R. Deepa**, Assistant Professor, Department of Information, Chaitanya Bharathi Institute of Technology.

I take the opportunity to express my thanks to **Dr K. Radhika**, Professor & Head of IT Department, CBIT for her valuable suggestions and moral support.

I am grateful to my Principal **Dr. G. P. Saradhi Varma**, Chaitanya Bharathi Institute of Technology, for his cooperation and encouragement. Finally, I also thank all the staff members, faculty of Dept. of IT, CBIT, and my friends, who with their valuable suggestions and support, directly or indirectly helped me in completing this project work.

ABSTRACT

Malware has always been a threat to the computer world, but with fast growth in the use of the Internet, malware severely affects the computer world. Malware predictors and detectors are critical tools in defence against malware. The existing malware detectors and predictors have been created, the effectiveness of these detectors and predictors depend upon the techniques being used. This study specifically addressed the following objectives: propose a model to predict malware behaviour using machine activity data and apply the Random Forest, LSTM model, Logistic Regression. In the proposed work of this research, useful machine learning models are developed and implemented with a malware database. The proposed multi-layer machine learning model is used for training and predictive malware analysis on multiple parameters, including error factor, accuracy rate, and overall performance. Result of the model from the evaluation measures provides a high accuracy rate and a lesser mean absolute error value. There are very few parameters in the random forest as well, and these can be optimized using generalization theory without having to separate validation sets during training.

TABLE OF CONTENTS

Title: **Page no.**

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

Digital Technology has a part of today's life in which the worlds of business and education depend on technology and its applications. However, these advances have also opened opportunities for the attacking community, and within a few years, malware has become the primary security threat, affecting computers and the network widely. Malicious software is known as malware. Computer malware is a program that, when executed, re-produce and infects a computer, poses a threat to the integrity of the system. The scope of the malware harm could be anywhere from removing files, destroying software to reformatting the hard disk.

The malware will spread the systems in a variety of ways. One way is to download from the internet, and once the malware finds its way to the systems, the action will begin. Some of the time, the malware will not harm the system; otherwise, it could affect the performance and cause an overload method. On the other hand, some malware is hidden in the process, which is difficult to detect by the current malware detection. Based on the above challenges, it is important to carry out more in-depth analysis to understand the malware for better detection and predictability.

On the feasibility of online malware detection with performance counters, this was discussed by machine learning to detect discrepancies between normal and malicious performance counter measurements during execution. Detection techniques proposed in Aware processor: a platform for efficient online malware detection, referred to as conventional machine-based detection, aim at finding a classifier that distinguishes malicious and benign information. In an attempt to differentiate between the actions of benign and malicious data, a trained classifier is a single model. The problem, however, is that most malware is introduced into programs that are otherwise harmless. For a specific program, its execution could be either benign or malicious depending on whether the malware is installed activated, making it difficult to classify the software during practice, the classifier is also really the classifier is also searching for the option of benevolent and malicious examples in the training set and may lead to undesirable false positives and false negatives.

Malware types: To have a better understanding of the methods and logic behind the malware, it is useful to classify it. Malware can be divided into several classes depending on its purpose. The classes are as follows: Virus. This is the simplest form of software. It is simply any piece of software that is loaded and launched without user’s permission while reproducing itself or infecting (modifying)other software. **Worm**: This malware type is very similar to the virus. The difference is that worms can spread over the network and replicate to Trojan machines. This malware class is used to define the malware types that aim to appear as legitimate software. Because of this, the general spreading vector utilized in this class is social engineering, i.e. making people think that they are downloading legitimate software. **Adware**: The only purpose of this malware type is displaying advertisements on the computer. Often adware can be seen as a subclass of spyware and it will very unlikely lead to dramatic results.

Spyware: As it implies from the name, the malware that performs espionage can be referred to as spyware. Typical actions of spyware include tracking search history to send personalized advertisements, tracking activities to sell them to the third parties subsequently. **Rootkit:** Its functionality enables the attacker to access the data with higher permissions than is allowed. For example, it can be used to give an unauthorized user administrative access. Rootkits always hide its existence and quite often are unnoticeable on the system, making the detection and therefore removal incredibly hard. **Backdoor**: The backdoor is a type of malware that provides an additional secret “entrance” to the system for attackers. By itself, it does not cause any harm but provides attackers with a broader attack surface. Because of this, backdoors are never used independently. Usually, they are preceded by malware attacks of other types.

Keylogger: The idea behind this malware class is to log all the keys pressed by the user, and, therefore, store all data, including passwords, bank card numbers and other sensitive information. **Ransomware:** This type of malware aims to encrypt all the data on the machine and ask a victim to transfer some money to get the decryption key. Usually, a machine infected by ransomware is “frozen” as the user cannot open any file, and the desktop picture is used to provide information on the attacker's demands. **Remote Administration Tools (RAT)**: This malware type allows an attacker to gain access to the system and make possible modifications as if it was accessed physically. Intuitively, it can be described in the example of the TeamViewer, but with malicious intentions.

OVERVIEW

What is DGA?

Domain generation algorithms (DGA) are algorithms seen in various families of malware that are used to periodically generate a large number of domain names that can be used as rendezvous points with their command and control servers. The large number of potential rendezvous points make it difficult for law enforcement to effectively shut down botnets, since infected computers will attempt to contact some of these domain names every day to receive updates or commands. The use of public-key cryptography in malware code makes it infeasible for law enforcement and other actors to mimic commands from the malware controllers as some worms will automatically reject any updates not signed by the malware controllers.

A botnet is a number of Internet-connected devices, each of which is running one or more bots. Botnets can be used to perform Distributed Denial-of -Service (DDoS) attacks, steal data,[1] send spam, and allow the attacker to access the device and its connection. The owner can control the botnet using command and control (C&C) software. A command-and-control [C&C] server is a computer controlled by an attacker or cybercriminal which is used to send commands to systems compromised by malware and receive stolen data from a target network. Many campaigns have been found using cloud-based services, such as webmail and file-sharing services, as C&C servers to blend in with normal traffic and avoid detection.

GOOD DGA, BAD DGA

The creators of DGA algorithms want to keep the uniqueness of the DGA so they can distinguish their C&C traffic from legitimate traffic, also avoid collision with other DGAs. Our research has shown us that some DGAs are smarter than others.

DICTIONARY BASED DGA

A little twist in the way algorithmically generated domains were created in the dictionary-based method. As we have seen, security researchers use features in the DNS string to separate malicious DGA traffic from legitimate traffic. The modelling work looks at attributes such as randomness, entropy and other lexical string features, which frequently generate domains with a 'random', 'non-human readable' look. (see for example, Some cleverly designed DGAs such as Suppobox try to evade this randomness by using dictionary words:

HIGH COLLISION DGA

DGAs like Pykspa and Virut are getting lower grades in our notebook: they have strong collisions with other legitimate names and other DGAs.

Pykspa is a worm whose DGA is reverse-engineered at . This DGA generates thousands of possible DGA domains using common TLDs like com, biz, net, org, info and cc, and its core domain has 6-15 chars. These thousands of domains flood the recursive DNS traffic. Because of the common TLD set and the short domain length for these huge amounts of domains, security researchers have a hard time to clearly identify and block them, even if they know the DGA + seed to predict. For example, some short domains like `wgxodod.info.` `ydnpxkv.info.` `hrv ccq.org.` have a good chance to collide with other DGAs (such as Locky), or with legitimate .com names.

Virut is another type of DGA where the domain name only has 6 a-z chars with .com TLD, and the algorithm itself has a simplistic design, so the chance of a generated domain colliding with a legitimate service is very high. We have observed many domains like `wenxin.com`, which was a legitimate domain, yet it was reported as Virus by some security researchers. And by the way, the domain `akamai.com` follows the exact pattern of a Virut DGA. But don't get too concerned. Blocking these high collision DGA domains in a safe way requires security researchers to combine the domain prediction method with DNS traffic; Our team has recently implemented a real-time new core domain detection system (for domains never seen before), where only the predicted DGA are blocked only if identified also as a new core domain.

Non DGA

In DNS traffic, we have observed many 'DGA-look-alike', which are not in fact DGA domains. For example, in recent traffic we saw these 7 char.ru domains with very high infection rate:

Examples: bhzlyxh.ru., qsxxzni.ru., gw ji ru.ru., fyxkmbh.ru., qwoumzw.ru.

1.2 APPLICATIONS

- The proposed solutions have achieved good results on a malware data set in a real-world Environment.
- Especially the multi-model ensemble for problem one can get very high accuracy for classification.
- Stop Exploitation of data.
- Controlling threat attacks.

1.3 PROBLEM DEFINITION

To accurately identify and cluster domains that originate from known DGA-based techniques where we target to develop a security approach that autonomously mitigates network communications to unknown threats in a sequence. Firewall blacklisting constantly expands as the multiple sources of inputs expand iterating rules. However, sequences in a DGA may not be known to these inputs promptly. Moreover, for the malware that communicates with an appropriate domain correctly, a threat actor must register each respective domain name in the sequence to maintain the C2 or risk the loss of a node in the distribution. Figure 1 gives a scenario for such a case. Our research problem is to accurately identify and cluster domains that originate from known DGA-based techniques where we target to develop a security approach that autonomously mitigates network communications to unknown threats in a sequence. So, to create a model to detect the Domain generated algorithms domains, with higher accuracy.

1.4 AIM OF THE PROJECT

Proposed a model to predict malware behaviour using machine activity data. Apply the random forest algorithm, Logistic Regression and Deep learning model -LSTM in predicting malicious behaviour and find out the best model. THREAT INTELLIGENCE FEED AND ONGOING THREAT DATA DGA's are plentiful through multiple online examples that are found from Google searching and Github repositories. However, sophisticated threat actors purposely create tailored DGA to evaluate current detection systems. Using real-time active malicious domains derived from DGAs on the public Internet measures the accuracy of the proposed approach. Specifically, threat intelligence feeds collected from Bambenek Consulting over a period of one year were obtained through daily manual querying demonstrated trends of ongoing threats. The structure of the data is presented in a CSV format of domain names, originating malware, and DGA membership with the daily file size of approximately 110MB.

1.5 ORGANIZATION OF THE REPORT

Chapter 1 deals with the introduction of the project and explains the purpose of the project. **Chapter 2** deals with the literature survey **Chapter 3** deals with the requirements that are needed in-order to execute the project **Chapter 4** deals with the methodology, system design and features of the project **Chapter 5** deals with the implementation of the project **Chapter 6** involves the results of the project **Chapter 7** involves conclusion and future scope of the project.

2. LITERATURE SURVEY

2.1 PAPER 1

Title: MACHINE LEARNING FRAMEWORK FOR DOMAIN GENERATION ALGORITHM-BASED MALWARE DETECTION.

Author: Tommy Chin,Kaiqi Xiong,Chengbin Hu,Yi Li.

Year of Publication: 2019.

In the study, they have assumed that DGA domains have groups of very significant characters from normal domains. By grouping domains according to their features, the authors applied a machine learning classifier to distinguish DGA domains from normal domains easily. Several machine-learning techniques have been studied to classify malicious codes. They include neural networks, support vector machines (SVM) and boosted classifiers. There are also several studies aiming to predict DGA domain names from historical DGA domains. Woodbridge et al. used DNS queries to find the pattern of different families of DGAs. Their approach does not need a feature extraction step. Instead, it leverages long short-term memory (LSTM) networks for real-time DGA prediction.

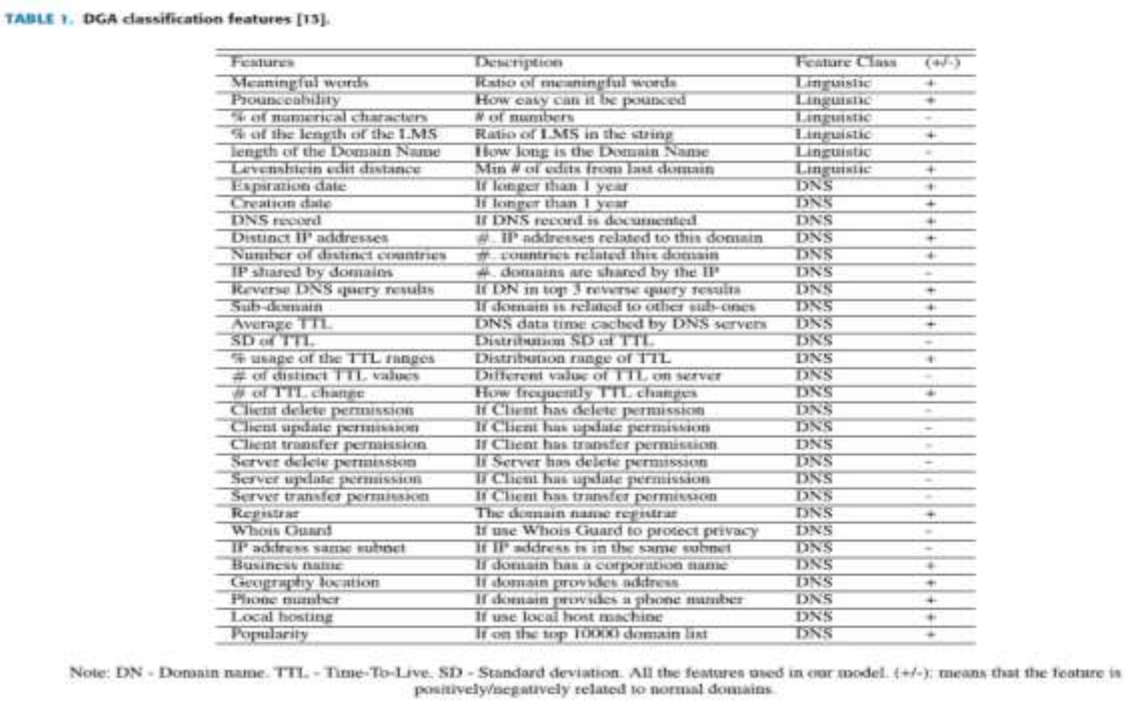


TABLE 1. DGA classification features [15].

Features	Description	Feature Class	(+/-)
Meaningful words	Ratio of meaningful words	Linguistic	+
Prounceability	How easy can it be pounced	Linguistic	+
% of numerical characters	# of numbers	Linguistic	-
% of the length of the LMS	Ratio of LMS in the string	Linguistic	+
length of the Domain Name	How long is the Domain Name	Linguistic	-
Levenshtein edit distance	Min # of edits from last domain	Linguistic	+
Expiration date	If longer than 1 year	DNS	+
Creation date	If longer than 1 year	DNS	+
DNS record	If DNS record is documented	DNS	+
Distinct IP addresses	#. IP addresses related to this domain	DNS	+
Number of distinct countries	#. countries related this domain	DNS	+
IP shared by domains	#. domains are shared by the IP	DNS	-
Reverse DNS query results	If DN in top 3 reverse query results	DNS	+
Sub-domain	If domain is related to other sub-ones	DNS	+
Average TTL	DNS data time cached by DNS servers	DNS	+
SD of TTL	Distribution SD of TTL	DNS	-
% usage of the TTL ranges	Distribution range of TTL	DNS	+
# of distinct TTL values	Different value of TTL on server	DNS	-
# of TTL change	How frequently TTL changes	DNS	+
Client delete permission	If Client has delete permission	DNS	-
Client update permission	If Client has update permission	DNS	-
Client transfer permission	If Client has transfer permission	DNS	-
Server delete permission	If Server has delete permission	DNS	-
Server update permission	If Client has update permission	DNS	-
Server transfer permission	If Client has transfer permission	DNS	-
Registrar	The domain name registrar	DNS	+
Whois Guard	If use Whois Guard to protect privacy	DNS	-
IP address same subnet	If IP address is in the same subnet	DNS	-
Business name	If domain has a corporation name	DNS	+
Geography location	If domain provides address	DNS	+
Phone number	If domain provides a phone number	DNS	+
Local hosting	If use local host machine	DNS	+
Popularity	If on the top 10000 domain list	DNS	+

Note: DN - Domain name. TTL - Time-To-Live. SD - Standard deviation. All the features used in our model. (+/-): means that the feature is positively/negatively related to normal domains.

Figure 2.1.1: DGA classification features.

Dynamic Blacklist:

The domain names are the only information we need to perform classification and prediction. We apply a domain-request packet filter to filter out the trivial information, which is useless. The filtered domain names are stored in the dynamic blacklist, which is initially empty and will be updated dynamically. Then filtered domain names are sent to the feature extractor in the next step.

Feature Extraction:

The feature extractor is used to extract features from the domain names filtered in the first component. Each domain name is considered as a string. To efficiently classify domains, we use two types of features: linguistic features and DNS features. Linguistic features include Length, Meaningful Word Ratio, Percentage of Numerical Characters etc. DNS features include IP Address.

Advantages:

1.Achieved good results on a malware data set in a real-world environment.
2.The multi-model ensemble for problem one can get very high accuracy for classification.

Disadvantages:

Limitation of size in storing large data.

2.2 PAPER 2

Title: Pontus: A Linguistics-based DGA Detection System**.**

Author: Dingkui Yan, Huilin Zhang.

Year of Publication:2019.

In this paper, they propose a DGA detection system, called Pontus, which is based upon powerful linguistics-based features. The features of Pontus are extracted exclusively from the individual domain name, Pontus still has a good classification performance. Their system is based upon the key insight that benign domain names and mAGD's differ greatly in the linguistic aspect. Benign domain names often represent some specific meanings, such as a brand name, a person's name. Those domain names usually adhere to the regular linguistic pattern for fluent reading or easy remembering. However, the random-looking mAGD's disobey regular linguistic patterns. Though wordlist based mAGD's follow the regular linguistic pattern, they can be split into 2 or 3 words completely. Sometimes, the 2 or 3 words are separated by hyphens.

The input to Pontus is domain names, and the output is the label of each domain name, benign or mAGD.
Pontus has two major phases:
1) Training phase and 2) Classification phase.
The training phase uses labelled domain names to train a supervised classifier.
The classification phase detects mAGD's from DNS traffic.

Advantages:

Easily building a model.

Detection of DGA domains based on DomainName is easy.

Disadvantages:

Only limited to Domain Name.

IP addresses are widely used for searching, only domain names don't provide high security.

2.3 PAPER 3

Title: Blacklist-based Malicious IP Traffic Detection

Author: Vaclav aprenosilIbrahinGhafir

Year of Publication:2017.

They proposed a methodology for detecting any connection to or from malicious IP. The detection method is based on a blacklist of malicious IPs. They process the network traffic and match the source and destination IP addresses for each connection with IP blacklist. The blacklist is automatically updated each day based on different intelligence feeds at once and the detection is in the real time. They have implemented the detection method on top of Bro, which is a passive, open-source network traffic analyser.

Advantages:

Quick process to validate against blacklist.

Disadvantages:

Only limited to Ips.

Newly entered domains are validated with a stored blacklist which is not dynamic.

3. SYSTEM REQUIREMENTS SPECIFICATIONS

3.1 SOFTWARE REQUIREMENTS

3.1.1 Colab

Google Colaboratory is a free online cloud-based Jupyter notebook environment that allows us to train our machine learning and deep learning models on CPUs, GPUs, and TPUs.

Colaboratory is a Google research project created to help disseminate machine learning education and research. It's a Jupyter notebook environment that requires no setup to use and runs entirely in the cloud. Colab notebooks allow you to combine **executable code** and **rich text i**n a single document, along with **images, HTML, LaTeX** and more. When you create your own Colab notebooks, they are stored in your Google Drive account. You can easily share your Colab notebooks with co-workers or friends, allowing them to comment on your notebooks or even edit them. To find out more, see Overview of Colab. To create a new Collab note-book you can use the File menu. Data science with Colab you can harness the full power of popular Python libraries to analyse and visualise data. The code cell below uses numpy to generate some random data and uses matplotlib to visualise it. To edit the code, just click the cell and start editing.

You can import your own data into Colab notebooks from your Google Drive account, including from spreadsheets, as well as from GitHub and many other sources. Machine learning with Colab you can import an image dataset, train an image classifier on it, and evaluate the model, all in just a few lines of code. Colab notebooks execute code on Google's cloud servers, meaning you can leverage the power of Google hardware, including GPUs and TPUs, regardless of the power of your machine. All you need is a browser.

Colab is used extensively in the machine learning community with applications including:

- · Getting started with TensorFlow.
- · Developing and training neural networks.
- · Experimenting with TPUs.
- · Disseminating AI research.
- · Creating tutorials.

3.1.2 Python

Python is an interpreter, high-level and general-purpose programming language. Python's design philosophy emphasizes code readability with its notable use of significant whitespace. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects. Python is dynamically typed and garbage collected. It supports multiple programming paradigms, including structured (particularly, procedural), object-oriented, and functional programming. Python is often described as a "batteries included" language due to its comprehensive standard library.

Key advantages of learning Python

- Python is Interactive − You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.
- Python is Object-Oriented − Python supports Object-Oriented style or technique of programming that encapsulates code within objects.
- Python is a Beginner's Language − Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

Characteristics of Python

- Following are important characteristics of Python Programming –
- It can be used as a scripting language or can be compiled to bytecode for building large applications.
- It provides very high-level dynamic data types and supports dynamic type checking.
- It supports automatic garbage collection.
- It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.

Applications of Python:

- Easy-to-learn − Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.
- Easy-to-read − Python code is more clearly defined and visible to the eyes.
- Easy-to-maintain − Python's source code is fairly easy-to-maintain.
- A broad standard library − Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.

- Interactive Mode − Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.
- Portable − Python can run on a wide variety of hardware platforms and has the same interface on all platforms.

3.1.3 Keras

Keras is one of the leading high-level neural networks APIs. It is written in Python and supports multiple back-end neural network computation engines. Keras contains numerous implementations of commonly used neural-network building blocks such as layers, objectives, activation functions, optimizers, and a host of tools to make working with image and text data easier to simplify the coding necessary for writing deep neural network code.Keras was created to be user friendly, modular, easy to extend, and to work with Python. The API was "designed for human beings, not machines," and "follows best practices for reducing cognitive load". Neural layers, cost functions, optimizers, initialization schemes, activation functions, and regularization schemes are all standalone modules that you can combine to create new models. New modules are simple to add, as new classes and functions. Models are defined in Python code, not separate model configuration files.

The **Model** is the core Keras data structure. There are two main types of models available in Keras: the Sequential model, and the Model class used with the functional API.

3.2 HARDWARE REQUIREMENTS:

Table 3.1: Hardware specifications

Operating System	Windows Operating System
Hard Drive Minimum	Minimum 32 GB; Recommended 64 GB or more
RAM	Minimum 1 GB; Recommended 4 GB or more

4. PROPOSED SYSTEM

4.1 SYSTEM ARCHITECTURE:

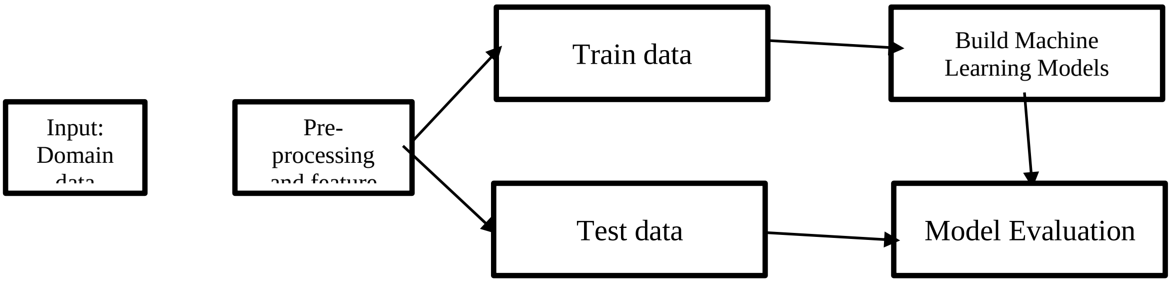


Figure 4.1: Flow chart of the proposed system

INPUT:

The three datasets are considered one dataset for Benign Domains, Alexa Top 1 Million Sites, which are a combination of good domains are taken from kaggle; two datasets for DGA Domains Bambenek Consulting provided malicious algorithmically generated domains and360 Lab DGA Domains.

Benign Domains:

- Alexa Top 1 Million Sites: The Alexa Top Sites web service provides access to lists of websites ordered by Alexa Traffic Rank. (Size: 2,476,328) (https://www.kaggle.com/cheedcheed/top1m)

Malicious DGA Domains:

- Bambenek Consulting provided malicious algorithmically generated domains (License) (Size: 872,763)
- 360 Lab DGA Domains: A collection of domains generated by DGA and it is maintained by 360--a Chinese security vendor. This dataset keeps updated every day. (Size: 1,169,720)

PREPROCESSING:

The data is pre-processed to remove unwanted and unnecessary data.In this stage we are handling noise data by ignoring the data if the data is either duplicate or empty. Pre-processing refers to the transformations applied to our data before feeding it to the algorithm.

Data preprocessing is a technique that is used to convert the raw data into a clean data set. In other words, whenever the data is gathered from different sources it is collected in raw format which is not feasible for the analysis.

FEATURE EXTRACTION:

The process of extracting data from the files is called feature extraction. The goal of feature extraction is to obtain a set of informative and non-redundant data. It is essential to understand that features should represent the important and relevant information about our dataset since without it we cannot make an accurate prediction. That is why feature extraction is often a non-obvious task, which requires a lot of testing and research. Moreover, it is very domain-specific, so general methods apply here poorly. Another Important requirement for a decent feature set is non-redundancy. Having redundant features i.e. features that outline the same information, as well as redundant information attributes that are closely dependent on each other, can make the algorithm biased and, therefore, provide an inaccurate result. In addition to that, if the input data is too big to be fed into the algorithm (has too many features), then it can be transformed to a reduced feature vector 15 (vector, having a smaller number of features).

The process of reducing the vector dimensions is referred to as feature selection. At the end of this process, we expect the selected features to outline the relevant information from the initial set so that it can be used instead of initial data without any accuracy loss.

The feature extractor is used to extract features from the domain names filtered in the first component. Each domain name is considered as a string. To efficiently classify domains, weuse two types of features: linguistic features and DNS features. We start with the discussion of linguistic features and then the DNS features.

There are six linguistic features: Length, Meaningful Word Ratio, Percentage of Numerical Characters, Pronounceability Score, Percentage of the Length of the Longest Meaningful String (LMS), and Levenshtein Edit Distance. The detailed description and calculation of each linguistic feature are given as follow: Length: We use |d| to represent the length of a domain name. Meaningful Word Ratio:

This feature measures the ratio of meaningful words in a string (domain name). The ratio is calculated as follows: f1 = Xni=1 |wi | |d| (1) where wi is the i-th meaningful substring of this string, |wi | is the length of ith meaningful substring. Since DGA domain names usually contain meaningless words; therefore, a small value of a ratio usually means that the domain could be a DGA domain name and a higher ratio implies a safer domain name. We strict the length of each meaningful substring |wi | in the string to be at least 4 letters because most legitimate domain names have meaningful substrings with more than 3 letters. For example, for a domain name of iylvword, we have f1= (|word|)/8 = 4/8 = 0.5. If a domain name is myproject, we have f1 = (|my| + |project|)/9 = (2+ 7)/9 = 1 because the domain is fully composed of two meaningful words.

Features of the DGA dataset

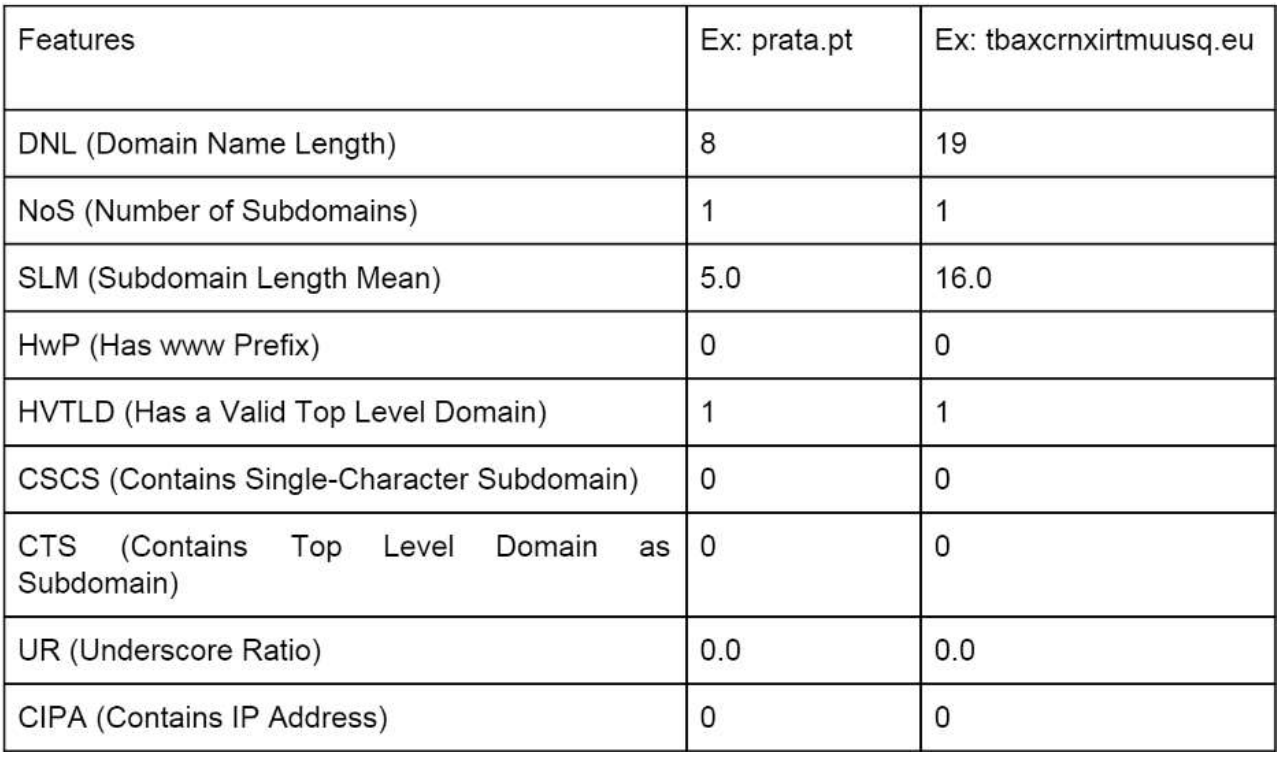
- DGA_Family: represents the family of DGA
- Domain
- Type: represents that a domain is a DGA domain or Normal DGA (This is the target variable which need to be predicted)

- Assign the values 0,1 to the Normal and DGA domains.

- **DNL (Domain Name Length):** represents the length of a domain
- **NoS (Number of Subdomains):** represents the number of subdomains *(*Ignore valid public suffixes).
- SLM (Subdomain Length Mean): represents the mean of subdomain length (Ignore valid public suffixes).
- HwP (Has www Prefix):
- **HVTLD (Has a Valid Top Level Domain):**
- **CSCS (Contains Single-Character Subdomain):** (Ignore valid public suffixes)
- **CTS (Contains Top Level Domain as Subdomain):** *(*Ignore valid public suffixes)
- **UR (Underscore Ratio): Represents the ratio of underscore** *(*Ignore valid public suffixes)
- CIPA (Contains IP Address): (Ignore valid public suffixes)
- **Contains-digit (Contains digit):** (Ignore valid public suffixes)
- **Vowel-ratio (The ratio of vowel):** (Ignore valid public suffixes)
- Digit-ratio (The ratio of digit): (Ignore valid public suffixes)
- RRC (The ratio of repeated characters in a subdomain): *(*Ignore valid public suffixes*)*
- RCC (The ratio of consecutive consonants): (Ignore valid public suffixes)
- RCD (The ratio of consecutive digits): (Ignore valid public suffixes)
- Entropy (The entropy of subdomain): (Ignore valid public suffixes)

Feature Engineering For the machine learning part, only the attribute of the domain itself is not enough for a machine learning algorithm. It needs some features. Applying features engineering first. Based on our knowledge and reference materials, three kinds of features will be generated: Structural Features; Linguistic Features; Statistical Features. For the first part of feature engineering: Features

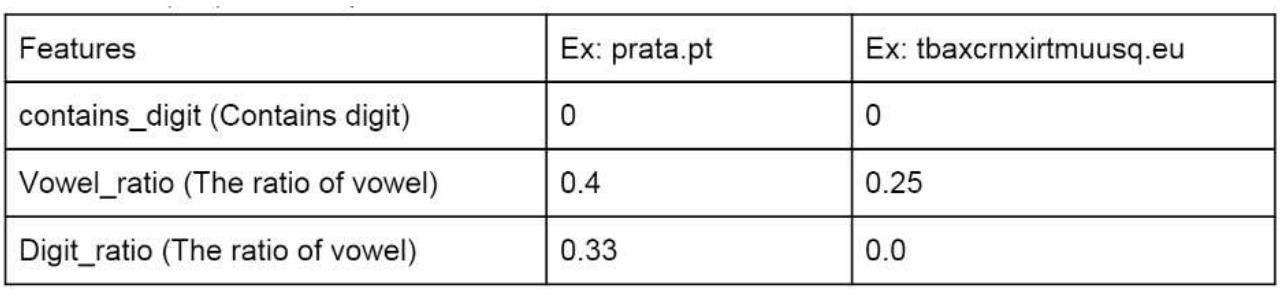
Table 4.1 Structural features.



Features	Ex: prata.pt	Ex: tbaxcrnxirtmuusq.eu
DNL (Domain Name Length)	8	19
NoS (Number of Subdomains)	1	1
SLM (Subdomain Length Mean)	5.0	16.0
HwP (Has www Prefix)	0	0
HVTLD (Has a Valid Top Level Domain)	1	1
CSCS (Contains Single-Character Subdomain)	0	0
CTS (Contains Top Level Domain as Subdomain)	0	0
UR (Underscore Ratio)	0.0	0.0
CIPA (Contains IP Address)	0	0

From Table 4.1, nine structural features are generated. For example, prata.pt, DNL (The length of the domain name) is 8. It only has 1 subdomain, so its NoS value is 1. The length of the subdomain (SLM) is the length of ‘prata’, which equals 5.0. It does not have www Prefix, so its Hwp value is 0. ‘.pt’ is a valid top-level domain, so its HVTLD domain is 1. It does not contain a single-character sub-domain, so the CSCS value is 0. So does the CTS. The ratio of underscore (UR) for example is 0 also. And it does not have an IP address.

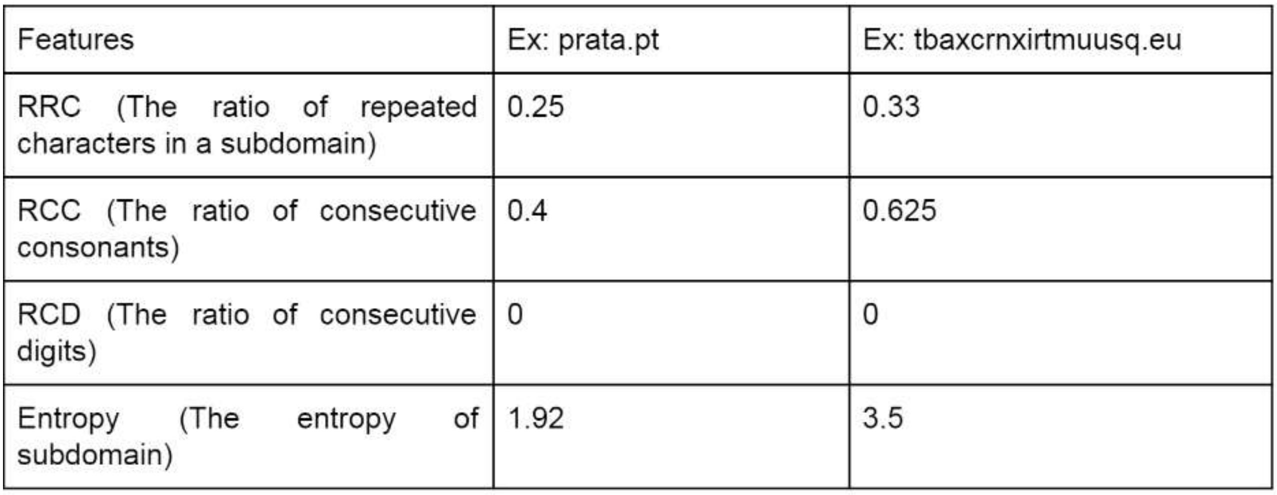
Table 4.2 Linguistic features.



Features	Ex: prata.pt	Ex: tbaxcrnxirtmuusq.eu
contains_digit (Contains digit)	0	0
Vowel_ratio (The ratio of vowel)	0.4	0.25
Digit_ratio (The ratio of vowel)	0.33	0.0

Based on linguistic analysis, three linguistic features are generated from the domain. Whether a domain contains a digit (contains-digit), the ratio of the vowel in a domain and the ratio of the digit. The value of these linguistic features can be known from Table 4.2

4.3 Statistical features.



Features	Ex: prata.pt	Ex: tbaxcrnxirtmuusq.eu
RRC (The ratio of repeated characters in a subdomain)	0.25	0.33
RCC (The ratio of consecutive consonants)	0.4	0.625
RCD (The ratio of consecutive digits)	0	0
Entropy (The entropy of subdomain)	1.92	3.5

There are also 4 statistical features that will be generated. From Table 3. RRC represents the ratio of repeated characters in a subdomain. RCC represents the ratio of consecutive consonants, RCD represents the ratio of consecutive digits and Entropy means the entropy of a sub-domain.

MODEL BUILDING:

The data is trained using Random Forest, Logistic Regression and LSTM and detects the DGA domain. Here before building the model the data is split into 80% train data and 20% test data. The models are built on train data. Once the models are built the predicted values are found by using those values.

ALGORITHMS

RANDOM FOREST(RF):

Random forest is a bunch of decision trees. It can be seen as an ensemble model. A random forest model will take all predicting results from its inner decision trees as a vote. Random-Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of ensemble learning, which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model.

It is an ensemble method which is better than a single decision tree because it reduces the over-fitting by averaging the result.

- It overcomes the problem of overfitting by averaging or combining the results of different decision trees.
- Random forests work better for a larger range of data items than a single decision tree does.
- Random forest has less variance then single decision tree.
- Random forests are very flexible and possess very high accuracy.
- Scaling of data does not require a random forest algorithm. It maintains good accuracy even after providing data without scaling.
- Random Forest algorithms maintain good accuracy even if a large proportion of the data is missing.

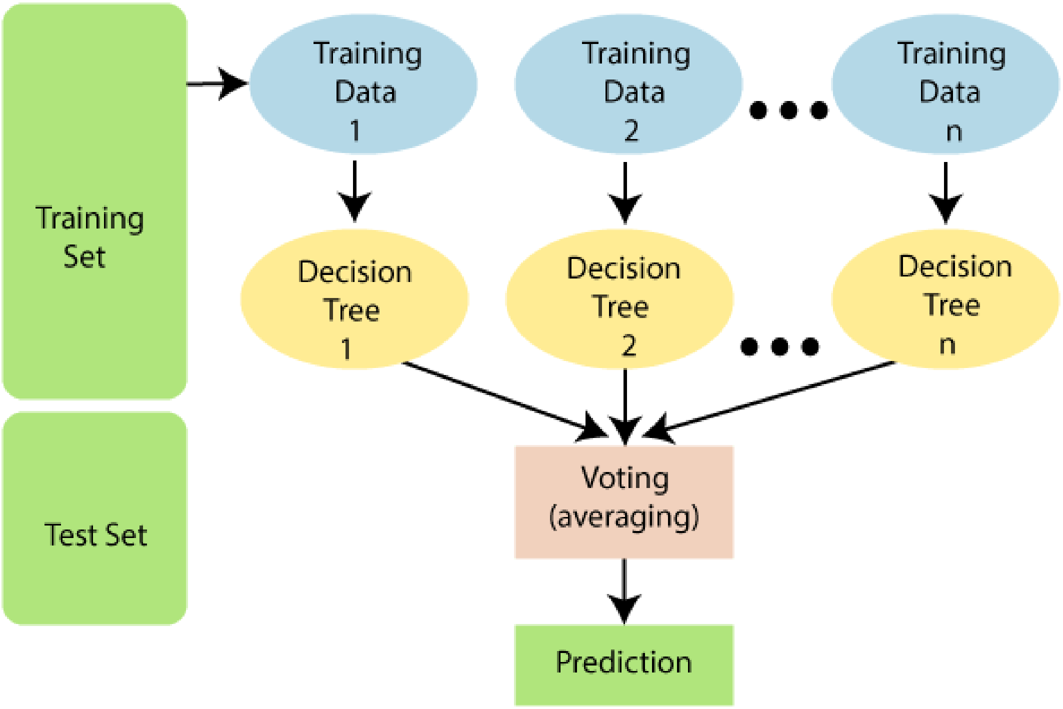


Figure 4.1.2 Random Forest

- There should be some actual values in the feature variable of the dataset so that the classifier can predict accurate results rather than a guessed result.

- The predictions from each tree must have very low correlations.

Working of Random Forest Model:

The Working process can be explained in the below steps:

Step-1: Select random K data points from the training set.

Step-2: Build the decision trees associated with the selected data points (Subsets).

Step-3: Choose the number N for decision trees that you want to build.

Step-4: Repeat Step 1 & 2.

Step-5: For new data points, find the predictions of each decision tree, and assign the new data points to the category that wins the majority votes.

Implementation Steps are given below:

- Data preprocessing step
- Fitting the Random-Forest algorithm to the Training set
- Predicting the test result
- Test accuracy of the result (Creation of Confusion matrix)
- Visualizing the test set result.

LSTM(Long Short-Term Memory Neural Network):

Long short-term memory (LSTM) units are units of a recurrent neural network (RNN). An RNN composed of LSTM units is often called an LSTM network. Long short-term memory (LSTM) is an artificial recurrent neural network (RNN) architecture used in the field of deep learning. It was proposed in 1997 by Sepp Hochreiter and Jurgen schmidhuber. Unlike standard feed-forward neural networks, LSTM has feedback connections. It can process not only single data points (such as images) but also entire sequences of data (such as speech or video).

For example, LSTM is an application to tasks such as unsegmented, connected handwriting recognition, or speech recognition. A general LSTM unit is composed of a cell, an input gate, an output gate, and a forget gate. The cell remembers values over arbitrary time intervals, and three gates regulate the flow of information into and out of the cell. LSTM is well-suited to classify, process, and predict the time series given of unknown duration. Feed Forward means that they always tend to move forward. They do not remember any previous information. If you want to add any new piece of data, it will overwrite the existing data. RNNs have something called Short Term Memory.

This short-term memory prevents them from storing data. In addition, RNNs cannot differentiate between important and less useful information.

This is different in LSTM. They have certain cell states within them. The information, which we give, passes through these states. These cell states help to separate out useful and non-useful information. This means that LSTM can remember or forget things. There are also three dependencies in these cells:

- Cell State (previous)
- Hidden State (previous)
- Current Time-Step

These are the states, which help LSTM to remember and make decisions.

LOGISTIC REGRESSION

Logistic regression is a statistical **model** that in its basic form uses a **logistic** function to **model** a binary dependent variable, although many more complex extensions exist.

Decision Tree is a Supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome. In a Decision tree, there are two nodes, which are the Decision Node and Leaf Node. Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches. The decisions or the test are performed on the basis of features of the given dataset.

It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions. It is called a decision tree because, similar to a tree, it starts with the root node, which expands on further branches and constructs a tree-like structure. In-order to build a tree, we use the CART algorithm, which stands for Classification and Regression Tree algorithm. A decision tree simply asks a question, and based on the answer (Yes/No), it further splits the tree into subtrees.

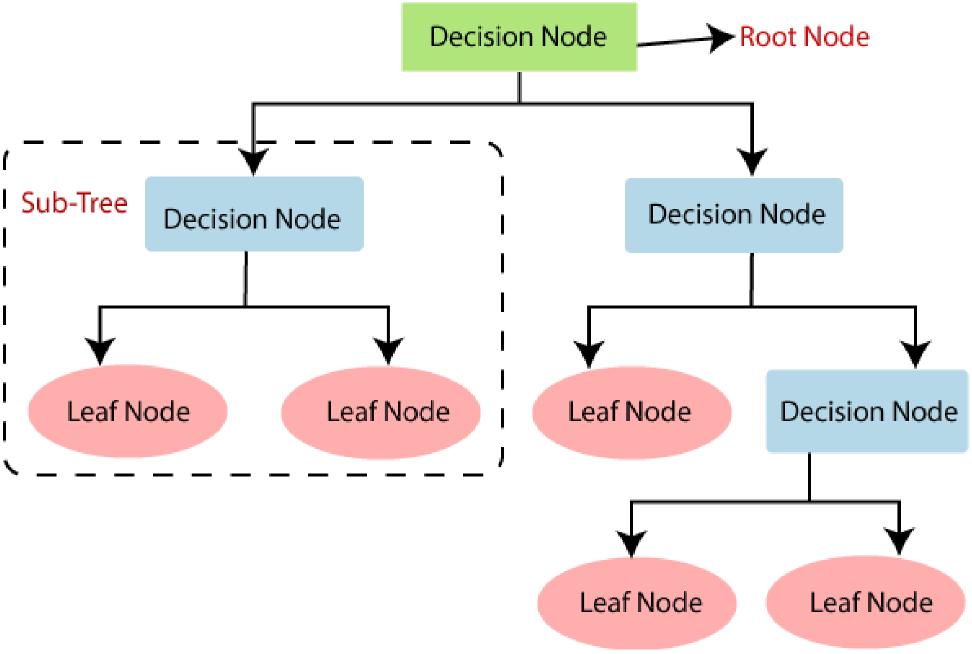
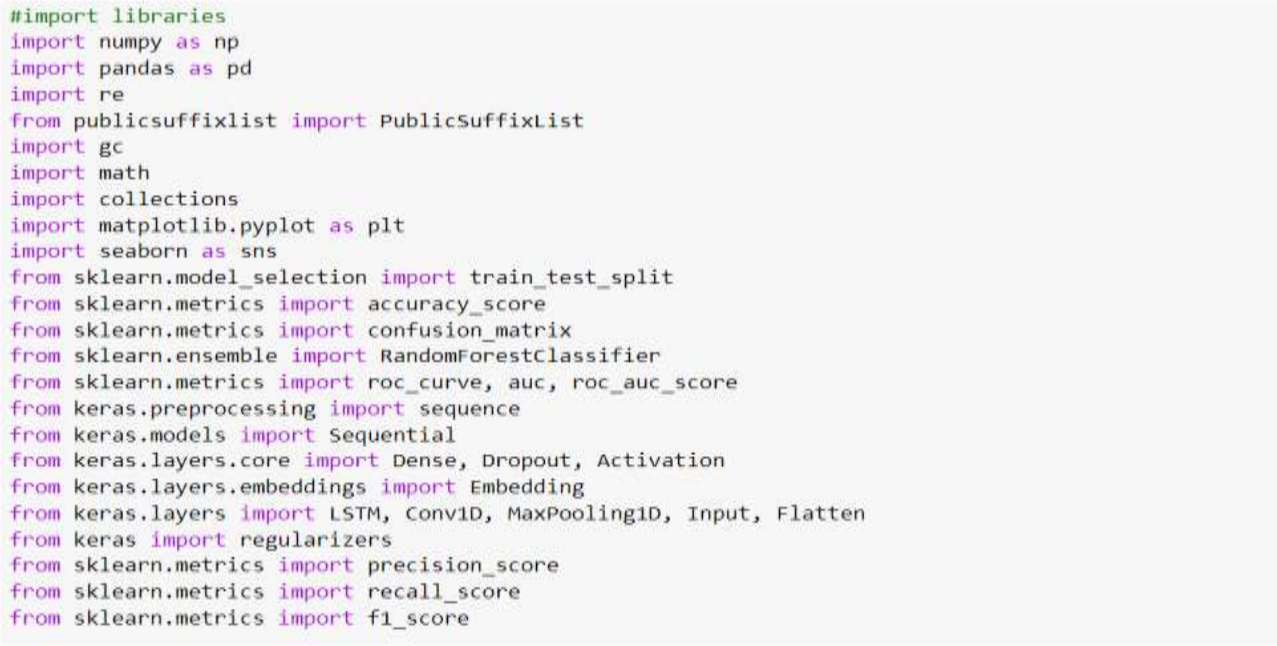


Figure 4.1.3 Logistic Regression

5.IMPLEMENTATION

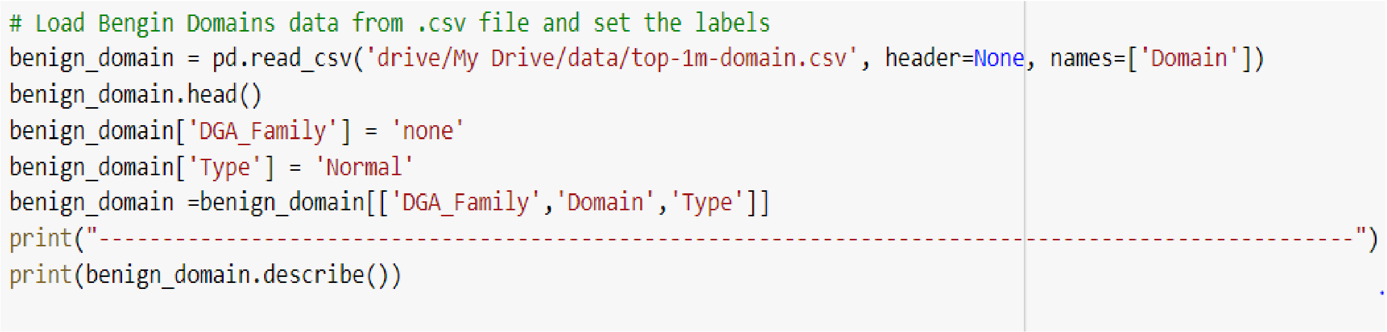
Figure 5.1 represents the collections, methods and functions that are used for implementing the project.



```
#import libraries
import numpy as np
import pandas as pd
import re
from publicsuffixlist import PublicSuffixList
import gc
import math
import collections
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_curve, auc, roc_auc_score
from keras.preprocessing import sequence
from keras.models import Sequential
from keras.layers.core import Dense, Dropout, Activation
from keras.layers.embeddings import Embedding
from keras.layers import LSTM, Conv1D, MaxPooling1D, Input, Flatten
from keras import regularizers
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
```

Figure 5.1 Import collections for classification

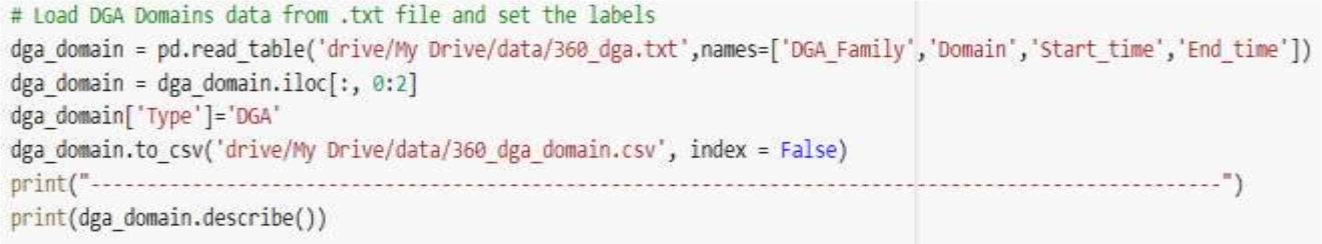
Figure 5.2 represents the loading of a benign domain’s dataset from the drive.



```
# Load Bengin Domains data from .csv file and set the labels
benign_domain = pd.read_csv('drive/My Drive/data/top-1m-domain.csv', header=None, names=['Domain'])
benign_domain.head()
benign_domain['DGA_Family'] = 'none'
benign_domain['Type'] = 'Normal'
benign_domain =benign_domain[['DGA_Family','Domain','Type']]
print("-----------------------------------------------------------------------------------------------------------")
print(benign_domain.describe())
```

Figure 5.2 Loading of Benign domains dataset

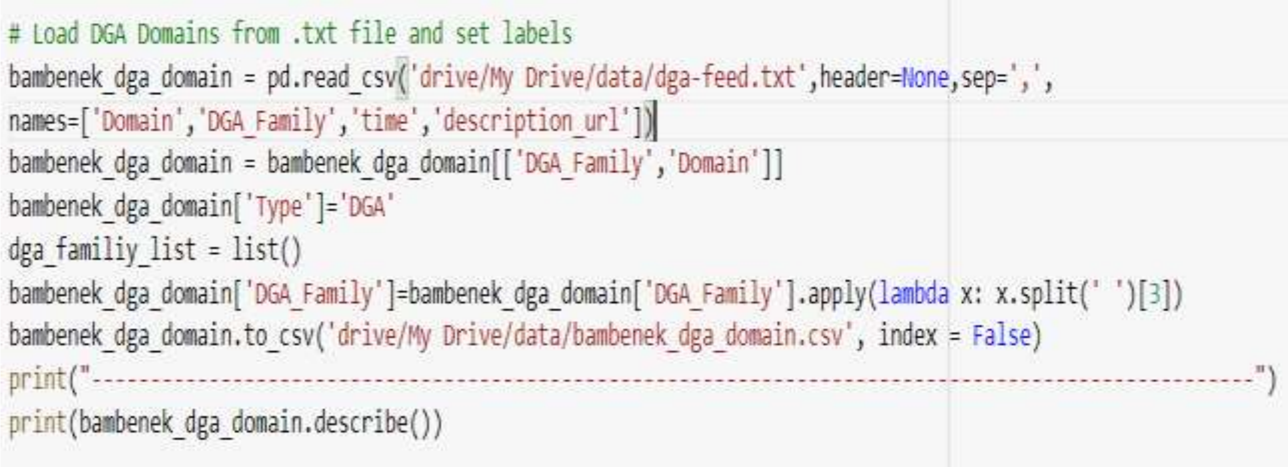
Figure 5.3 represents the loading of a DGA domains dataset from the drive.



```
# Load DGA Domains data from .txt file and set the labels
dga_domain = pd.read_table('drive/My Drive/data/360_dga.txt',names=['DGA_Family','Domain','Start_time','End_time'])
dga_domain = dga_domain.iloc[:, 0:2]
dga_domain['Type']='DGA'
dga_domain.to_csv('drive/My Drive/data/360_dga_domain.csv', index = False)
print("----------------------------------------------------------------------------------------------------------")
print(dga_domain.describe())
```

Figure 5.3 Loading of DGA domains dataset

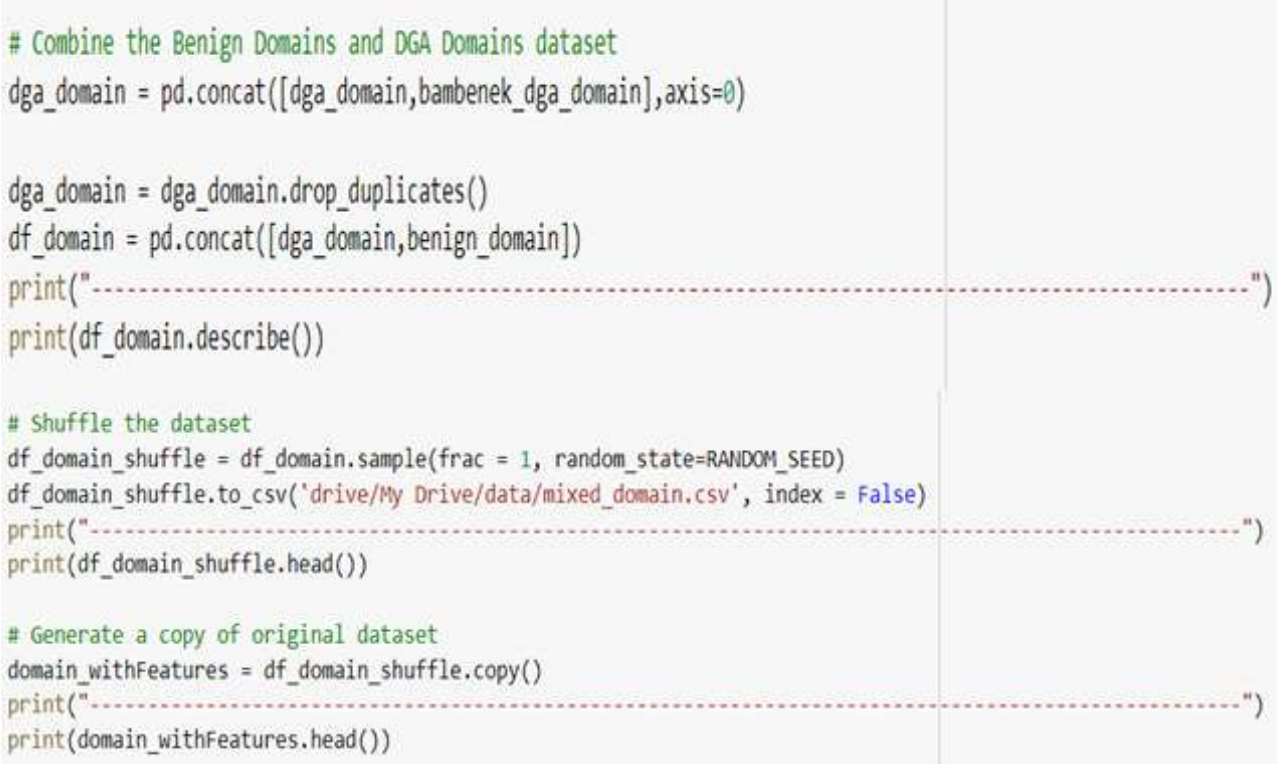
Figure 5.4 represents the loading of a DGA domains dataset from the drive.



```
# Load DGA Domains from .txt file and set labels
bambenek_dga_domain = pd.read_csv('drive/My Drive/data/dga-feed.txt',header=None,sep=',',
names=['Domain','DGA_Family','time','description_url'])
bambenek_dga_domain = bambenek_dga_domain[['DGA_Family','Domain']]
bambenek_dga_domain['Type']='DGA'
dga_familiy_list = list()
bambenek_dga_domain['DGA_Family']=bambenek_dga_domain['DGA_Family'].apply(lambda x: x.split(' ')[3])
bambenek_dga_domain.to_csv('drive/My Drive/data/bambenek_dga_domain.csv', index = False)
print("----------------------------------------------------------------------------------------------")
print(bambenek_dga_domain.describe())
```

Figure 5.4 Loading of DGA domains dataset

Figure 5.5 represents the concatenating of benign domain dataset and DGA domains. Initially it combines DGA domains datasets and in the next process it concatenates with normal domains dataset. Shuffle the combined dataset and creates a copy to the original dataset.

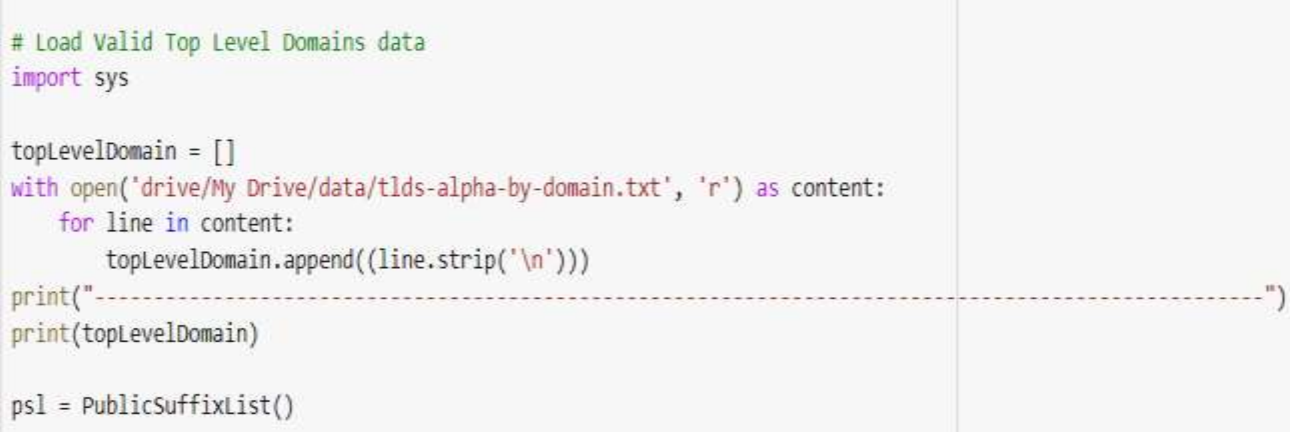


```
# Combine the Benign Domains and DGA Domains dataset
dga_domain = pd.concat([dga_domain,bambenek_dga_domain],axis=0)

dga_domain = dga_domain.drop_duplicates()
df_domain = pd.concat([dga_domain,benign_domain])
print("----------------------------------------------------------------------------------------------")
print(df_domain.describe())

# Shuffle the dataset
df_domain_shuffle = df_domain.sample(frac = 1, random_state=RANDOM_SEED)
df_domain_shuffle.to_csv('drive/My Drive/data/mixed_domain.csv', index = False)
print("----------------------------------------------------------------------------------------------")
print(df_domain_shuffle.head())

# Generate a copy of original dataset
domain_withFeatures = df_domain_shuffle.copy()
print("----------------------------------------------------------------------------------------------")
print(domain_withFeatures.head())
```

Figure 5.5 Concatenate benign dataset and DGA domains dataset and Shuffle the dataset

Figure 5.6 represents the Loading of Valid Top Level Domains data to ensure the suffix.

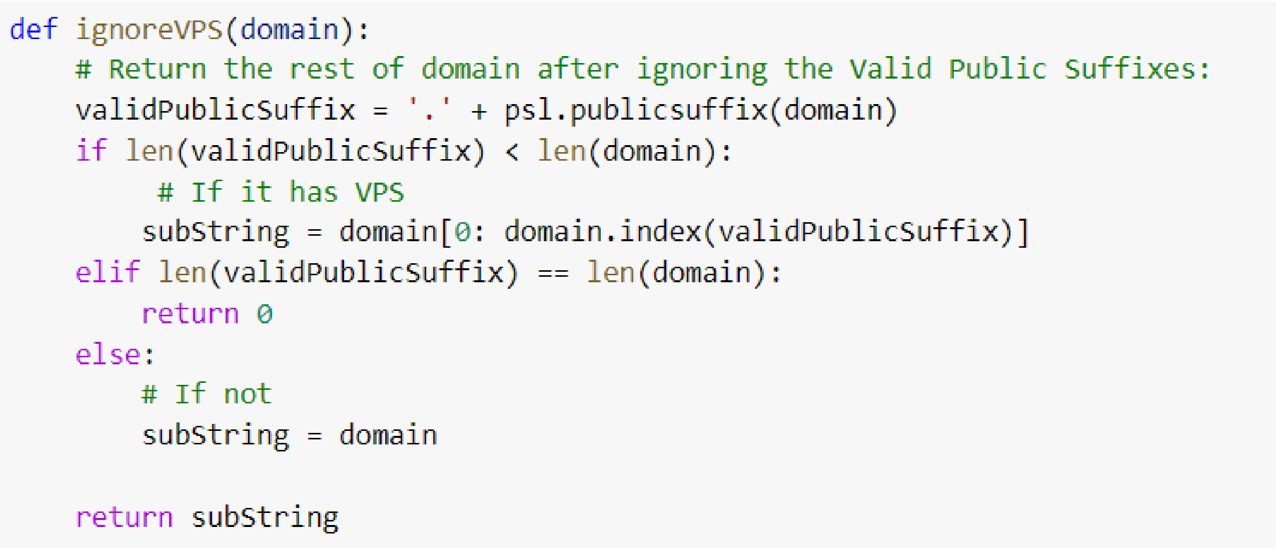
The given domain is divided into individual strings using split function.



```
# Load Valid Top Level Domains data
import sys

topLevelDomain = []
with open('drive/My Drive/data/tlds-alpha-by-domain.txt', 'r') as content:
    for line in content:
        topLevelDomain.append((line.strip('\n')))
print("---------------------------------------------------------------------------------------------------------------")
print(topLevelDomain)

psl = PublicSuffixList()
```

Figure 5.6 Loading Valid Top Level Domains data

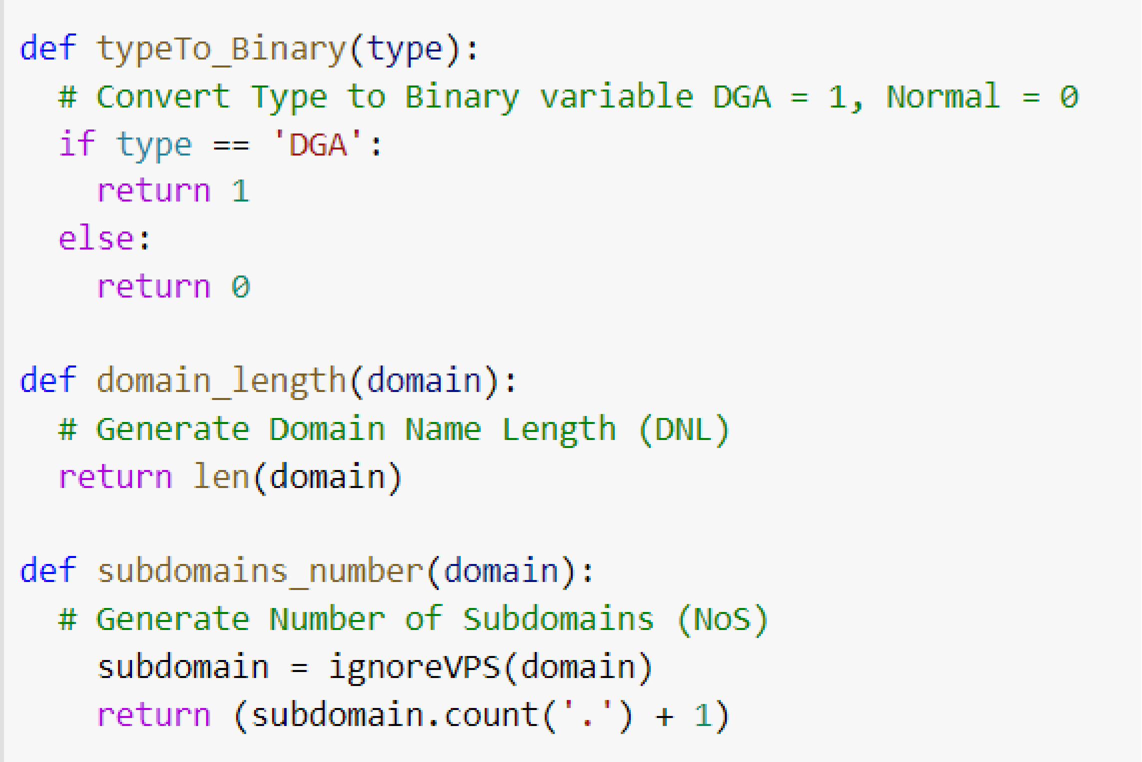
Figure 5.7 represents the feature extraction function which returns the rest of the domain after ignoring the valid public suffixes if it has VPS return 0 else it Returns 1.



```
def ignoreVPS(domain):
    # Return the rest of domain after ignoring the Valid Public Suffixes:
    validPublicSuffix = '.' + psl.publicsuffix(domain)
    if len(validPublicSuffix) < len(domain):
         # If it has VPS
        subString = domain[0: domain.index(validPublicSuffix)]
    elif len(validPublicSuffix) == len(domain):
        return 0
    else:
        # If not
        subString = domain

    return subString
```

Figure 5.7 ignore VPS

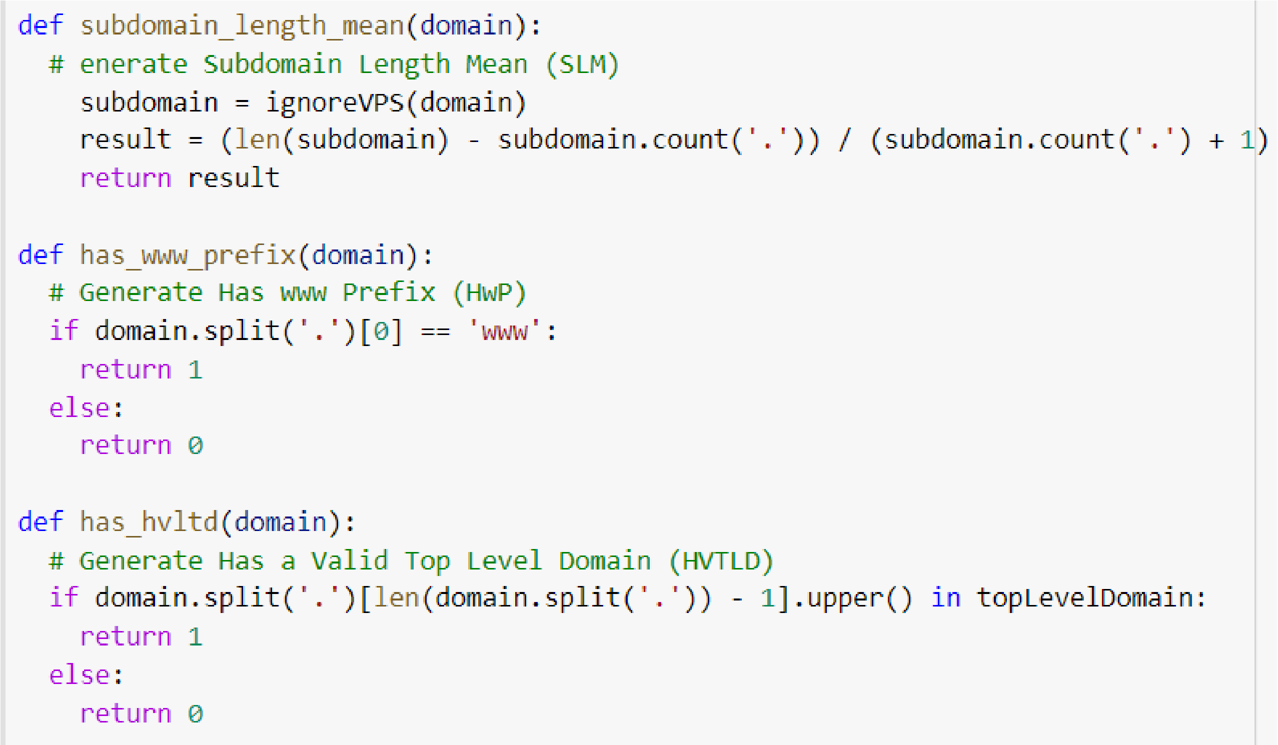
Figure 5.8 represents the type to Binary function which assigns DGA equal to 1 and Benign equal to zero, domain length function Returns length of the domain and subdomain number function returns count of subdomain.



```
def typeTo_Binary(type):
  # Convert Type to Binary variable DGA = 1, Normal = 0
  if type == 'DGA':
    return 1
  else:
    return 0

def domain_length(domain):
  # Generate Domain Name Length (DNL)
  return len(domain)

def subdomains_number(domain):
  # Generate Number of Subdomains (NoS)
    subdomain = ignoreVPS(domain)
    return (subdomain.count('.') + 1)
```

Figure 5.8 Domain length function

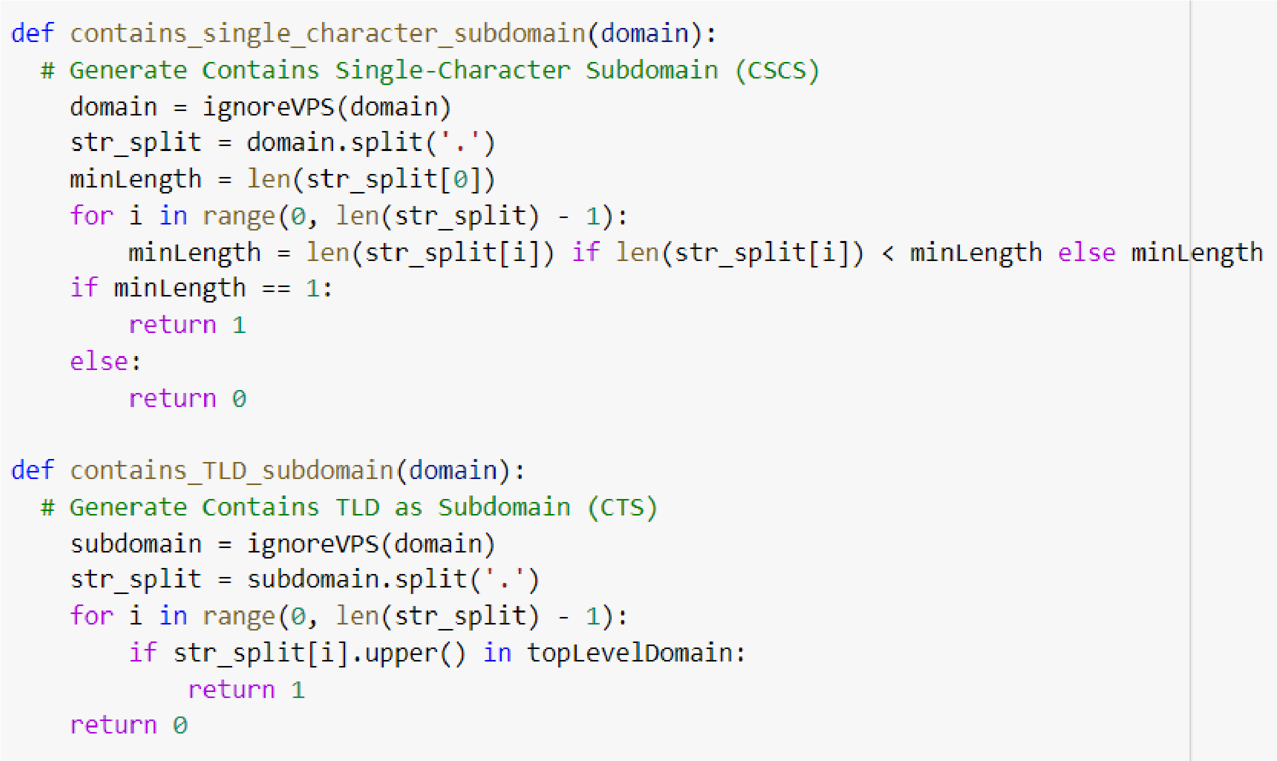
Figure 5.9 represents the subdomain length mean function which Generates the subdomain length mean and has www prefix function returns 1 if function has www else return 0, has_HVTLD function Generate has a valid top-level domain and it Returns 1 if it contains value top load domain else return 0.



```
def subdomain_length_mean(domain):
  # enerate Subdomain Length Mean (SLM)
    subdomain = ignoreVPS(domain)
    result = (len(subdomain) - subdomain.count('.')) / (subdomain.count('.') + 1)
    return result

def has_www_prefix(domain):
  # Generate Has www Prefix (HwP)
  if domain.split('.')[0] == 'www':
    return 1
  else:
    return 0

def has_hvltd(domain):
  # Generate Has a Valid Top Level Domain (HVTLD)
  if domain.split('.')[len(domain.split('.')) - 1].upper() in topLevelDomain:
    return 1
  else:
    return 0
```

Figure 5.9 Subdomain length mean.

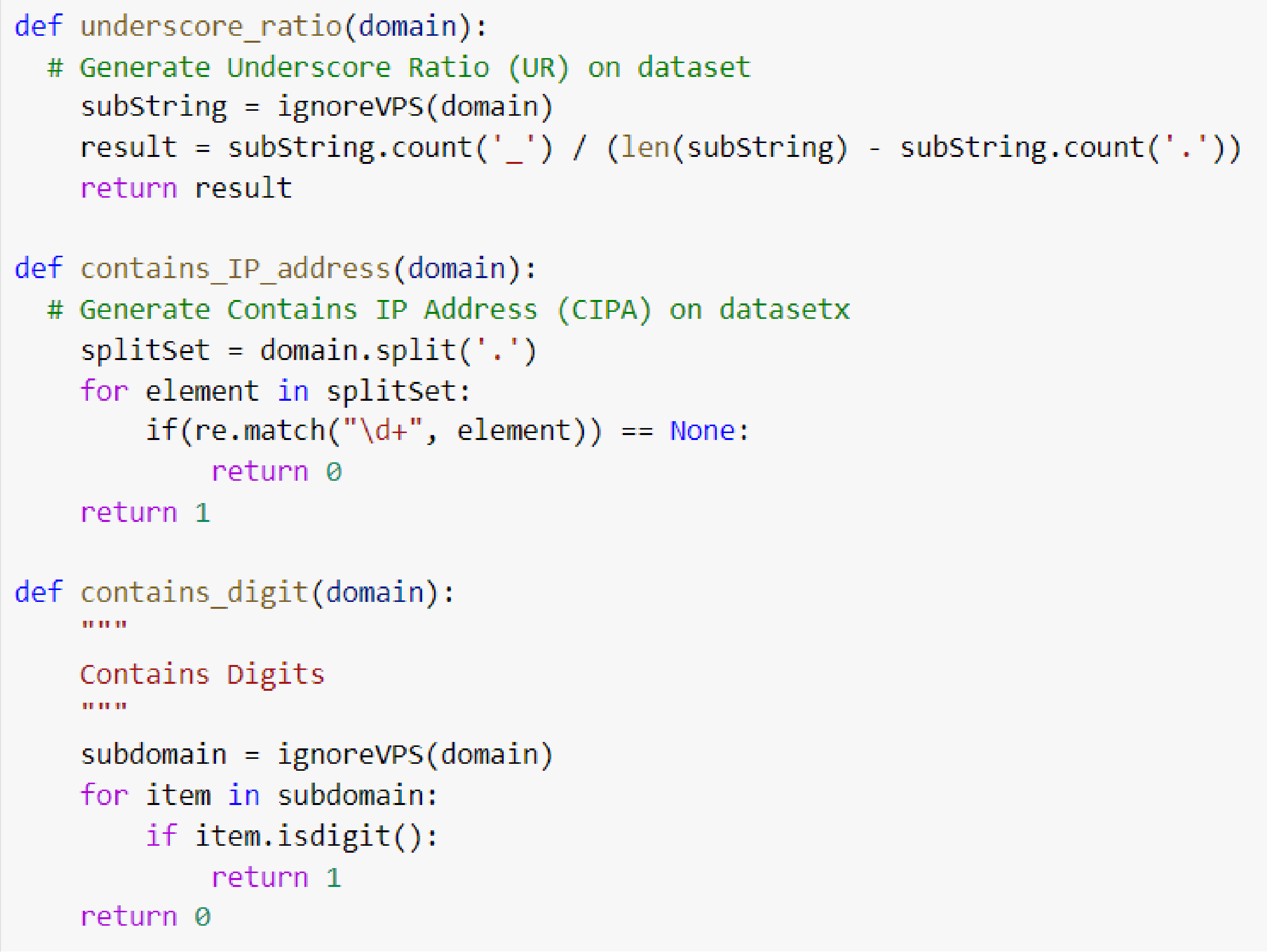
Figure 5.10 represents the contains single characters subdomain function Which returns value 1 if a minimum length equal to 1 and contains TLD subdomain function Returns 1 if it contains TLD as a sub-domain.



```python
def contains_single_character_subdomain(domain):
  # Generate Contains Single-Character Subdomain (CSCS)
    domain = ignoreVPS(domain)
    str_split = domain.split('.')
    minLength = len(str_split[0])
    for i in range(0, len(str_split) - 1):
        minLength = len(str_split[i]) if len(str_split[i]) < minLength else minLength
    if minLength == 1:
        return 1
    else:
        return 0

def contains_TLD_subdomain(domain):
  # Generate Contains TLD as Subdomain (CTS)
    subdomain = ignoreVPS(domain)
    str_split = subdomain.split('.')
    for i in range(0, len(str_split) - 1):
        if str_split[i].upper() in topLevelDomain:
            return 1
    return 0
```

Figure 5.10 Contains single characters sub-domain.

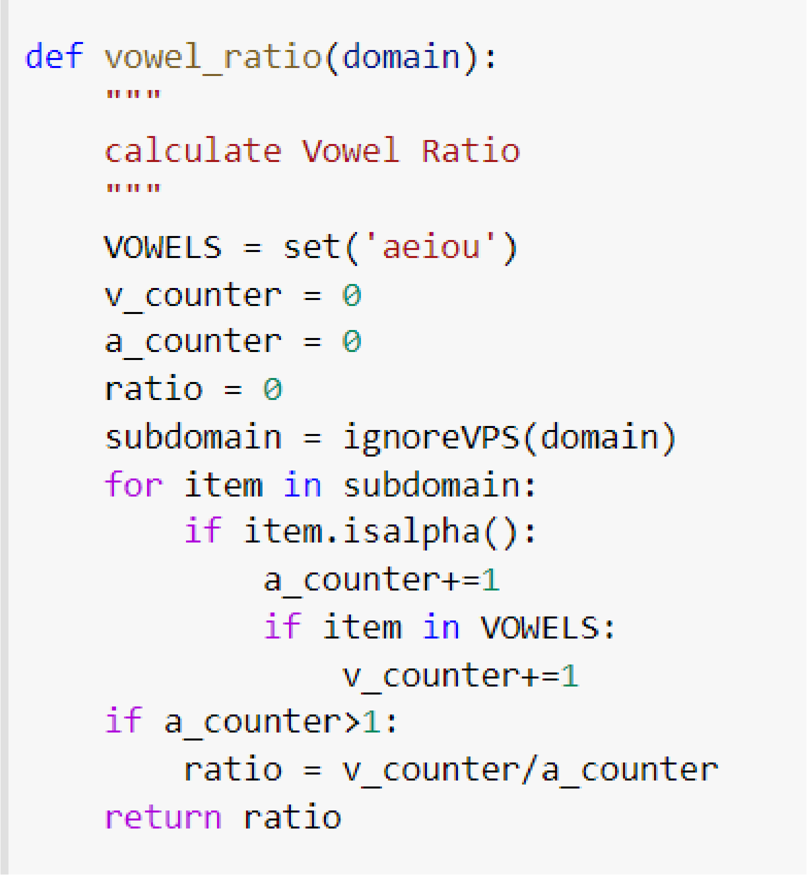
figure 5.11 represents the underscore ratio function, contains IP address function and contains digit function returns 1 if it contains digit else return 0.



```
def underscore_ratio(domain):
  # Generate Underscore Ratio (UR) on dataset
    subString = ignoreVPS(domain)
    result = subString.count('_') / (len(subString) - subString.count('.'))
    return result

def contains_IP_address(domain):
  # Generate Contains IP Address (CIPA) on datasetx
    splitSet = domain.split('.')
    for element in splitSet:
        if(re.match("\d+", element)) == None:
            return 0
    return 1

def contains_digit(domain):
    """
    Contains Digits
    """
    subdomain = ignoreVPS(domain)
    for item in subdomain:
        if item.isdigit():
            return 1
    return 0
```

Figure 5.11 Underscore ratio

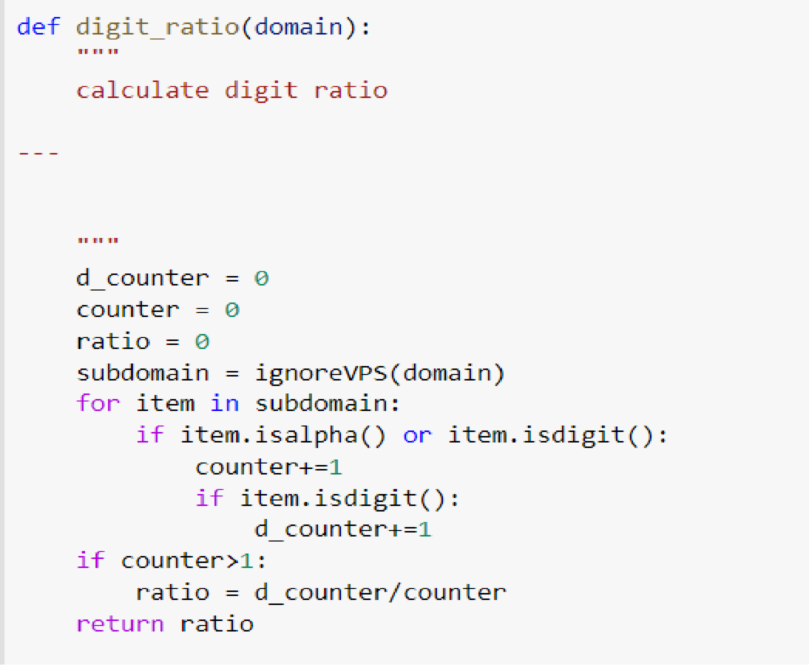
Figure 5.12 represents the vowel ratio function; it is the ratio between alphabets and vowels.



```python
def vowel_ratio(domain):
    """
    calculate Vowel Ratio
    """
    VOWELS = set('aeiou')
    v_counter = 0
    a_counter = 0
    ratio = 0
    subdomain = ignoreVPS(domain)
    for item in subdomain:
        if item.isalpha():
            a_counter+=1
            if item in VOWELS:
                v_counter+=1
    if a_counter>1:
        ratio = v_counter/a_counter
    return ratio
```

Figure 5.12 Vowel ratio

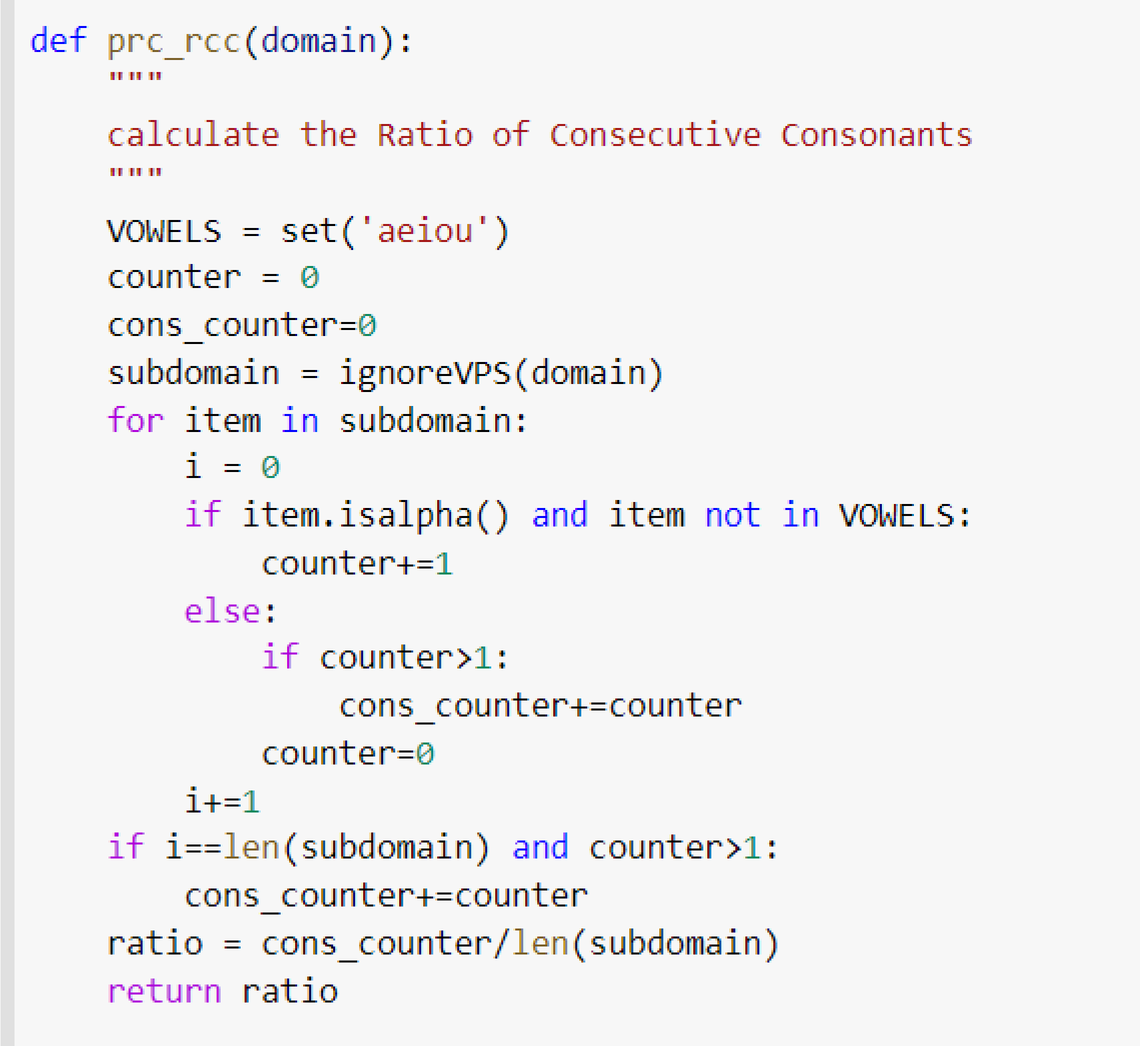
Figure 5.13 represent the digit ratio function which carat leads the ratio between alphabets and digits



```python
def digit_ratio(domain):
    """
    calculate digit ratio

---


    """
    d_counter = 0
    counter = 0
    ratio = 0
    subdomain = ignoreVPS(domain)
    for item in subdomain:
        if item.isalpha() or item.isdigit():
            counter+=1
            if item.isdigit():
                d_counter+=1
    if counter>1:
        ratio = d_counter/counter
    return ratio
```

Figure 5.13: Digit ratio

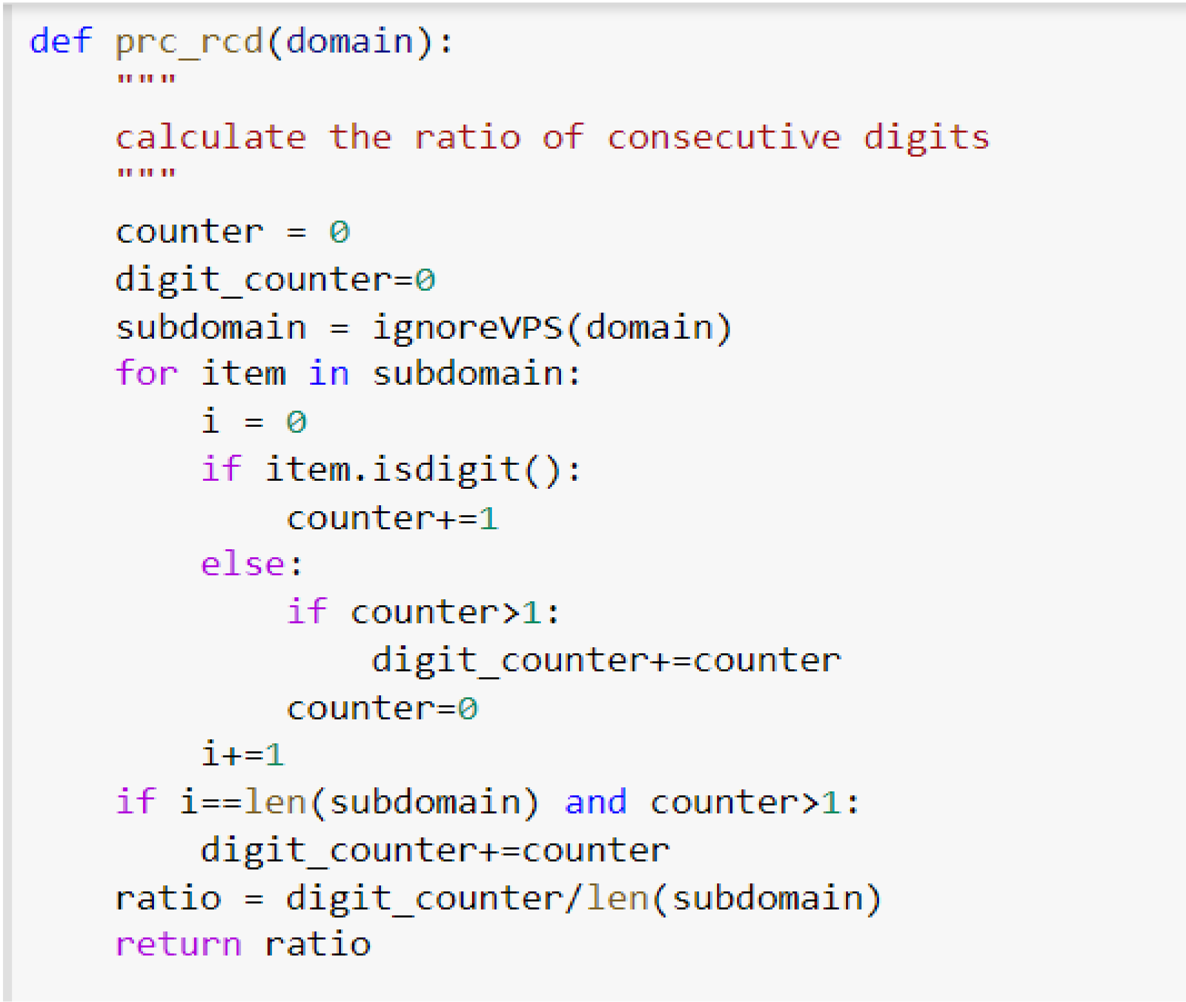
Figure 5.14 represents the prc_rcc function which calculates the ratio of consecutive consonants.



```python
def prc_rcc(domain):
    """
    calculate the Ratio of Consecutive Consonants
    """
    VOWELS = set('aeiou')
    counter = 0
    cons_counter=0
    subdomain = ignoreVPS(domain)
    for item in subdomain:
        i = 0
        if item.isalpha() and item not in VOWELS:
            counter+=1
        else:
            if counter>1:
                cons_counter+=counter
            counter=0
        i+=1
    if i==len(subdomain) and counter>1:
        cons_counter+=counter
    ratio = cons_counter/len(subdomain)
    return ratio
```

Figure 5.14 prc_rcc function

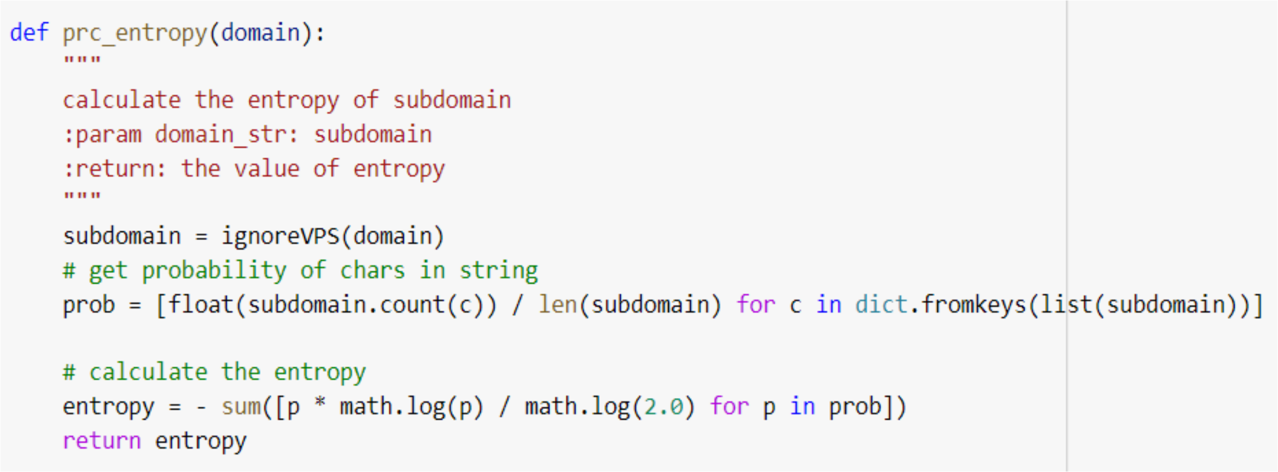
Figure 5.15 represents the prc_rcd function which represents the ratio of consecutive digits.



```
def prc_rcd(domain):
    """
    calculate the ratio of consecutive digits
    """
    counter = 0
    digit_counter=0
    subdomain = ignoreVPS(domain)
    for item in subdomain:
        i = 0
        if item.isdigit():
            counter+=1
        else:
            if counter>1:
                digit_counter+=counter
            counter=0
        i+=1
    if i==len(subdomain) and counter>1:
        digit_counter+=counter
    ratio = digit_counter/len(subdomain)
    return ratio
```

Figure 5.15 prc_rcd function

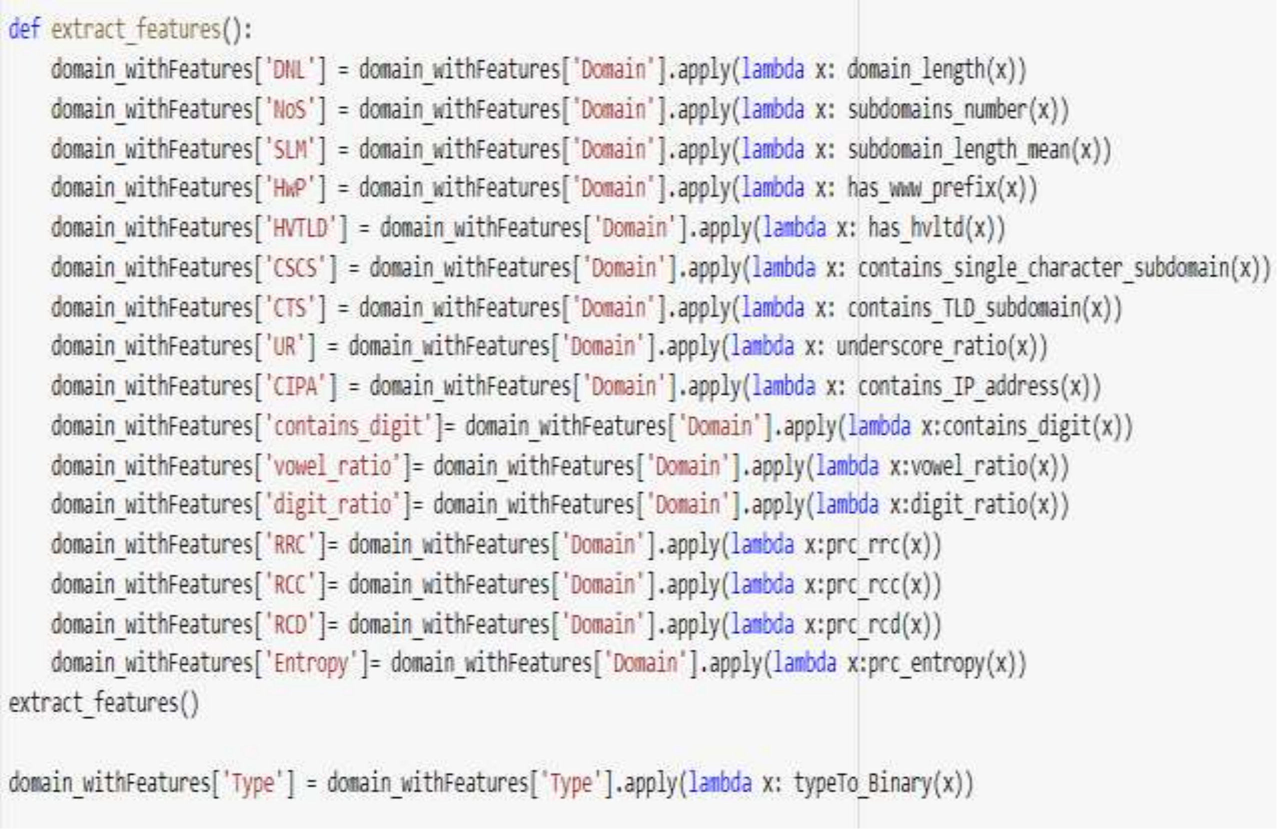
Figure 5.16 represent prc_entropy function which calculates and returns the entropy value of a subdomain.



```
def prc_entropy(domain):
    """
    calculate the entropy of subdomain
    :param domain_str: subdomain
    :return: the value of entropy
    """
    subdomain = ignoreVPS(domain)
    # get probability of chars in string
    prob = [float(subdomain.count(c)) / len(subdomain) for c in dict.fromkeys(list(subdomain))]

    # calculate the entropy
    entropy = - sum([p * math.log(p) / math.log(2.0) for p in prob])
    return entropy
```

Figure 5.16 prc_ entropy function

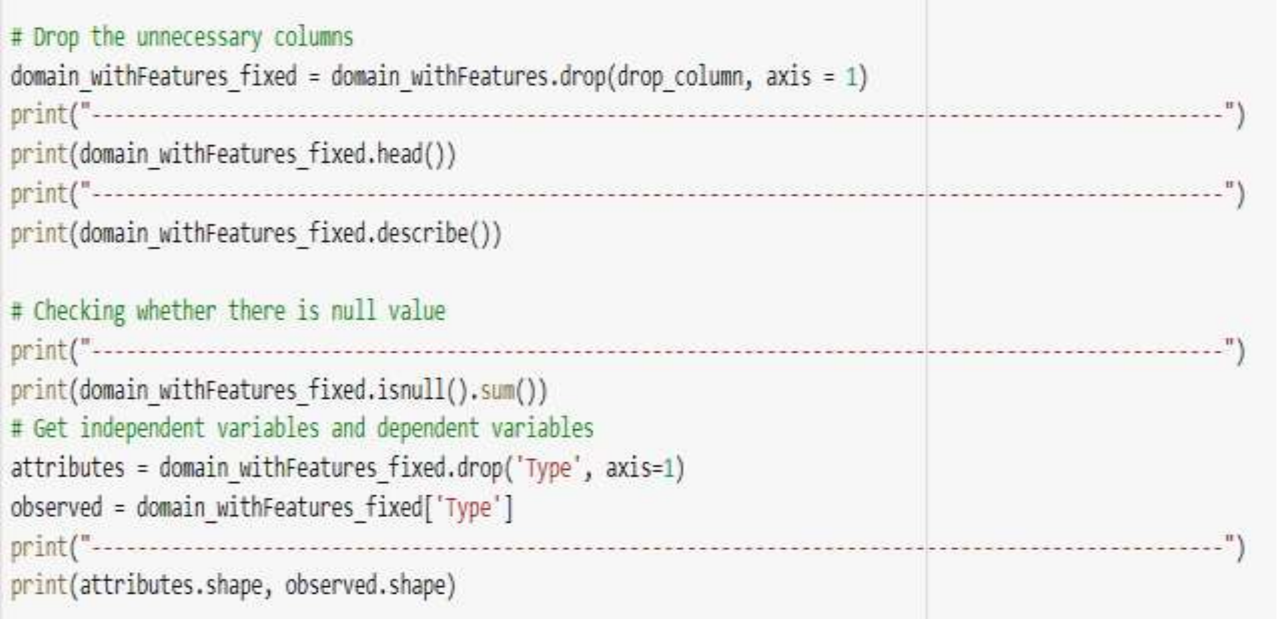
Figure 5.17 represents the declaration of all the extracted feature's and we use the python Lambda function to take the input continuously.



```
def extract_features():
    domain_withFeatures['DNL'] = domain_withFeatures['Domain'].apply(lambda x: domain_length(x))
    domain_withFeatures['NoS'] = domain_withFeatures['Domain'].apply(lambda x: subdomains_number(x))
    domain_withFeatures['SLM'] = domain_withFeatures['Domain'].apply(lambda x: subdomain_length_mean(x))
    domain_withFeatures['HwP'] = domain_withFeatures['Domain'].apply(lambda x: has_www_prefix(x))
    domain_withFeatures['HVTLD'] = domain_withFeatures['Domain'].apply(lambda x: has_hvltd(x))
    domain_withFeatures['CSCS'] = domain_withFeatures['Domain'].apply(lambda x: contains_single_character_subdomain(x))
    domain_withFeatures['CTS'] = domain_withFeatures['Domain'].apply(lambda x: contains_TLD_subdomain(x))
    domain_withFeatures['UR'] = domain_withFeatures['Domain'].apply(lambda x: underscore_ratio(x))
    domain_withFeatures['CIPA'] = domain_withFeatures['Domain'].apply(lambda x: contains_IP_address(x))
    domain_withFeatures['contains_digit']= domain_withFeatures['Domain'].apply(lambda x:contains_digit(x))
    domain_withFeatures['vowel_ratio']= domain_withFeatures['Domain'].apply(lambda x:vowel_ratio(x))
    domain_withFeatures['digit_ratio']= domain_withFeatures['Domain'].apply(lambda x:digit_ratio(x))
    domain_withFeatures['RRC']= domain_withFeatures['Domain'].apply(lambda x:prc_rrc(x))
    domain_withFeatures['RCC']= domain_withFeatures['Domain'].apply(lambda x:prc_rcc(x))
    domain_withFeatures['RCD']= domain_withFeatures['Domain'].apply(lambda x:prc_rcd(x))
    domain_withFeatures['Entropy']= domain_withFeatures['Domain'].apply(lambda x:prc_entropy(x))
extract_features()

domain_withFeatures['Type'] = domain_withFeatures['Type'].apply(lambda x: typeTo_Binary(x))
```

Figure 5.17 Extracted features

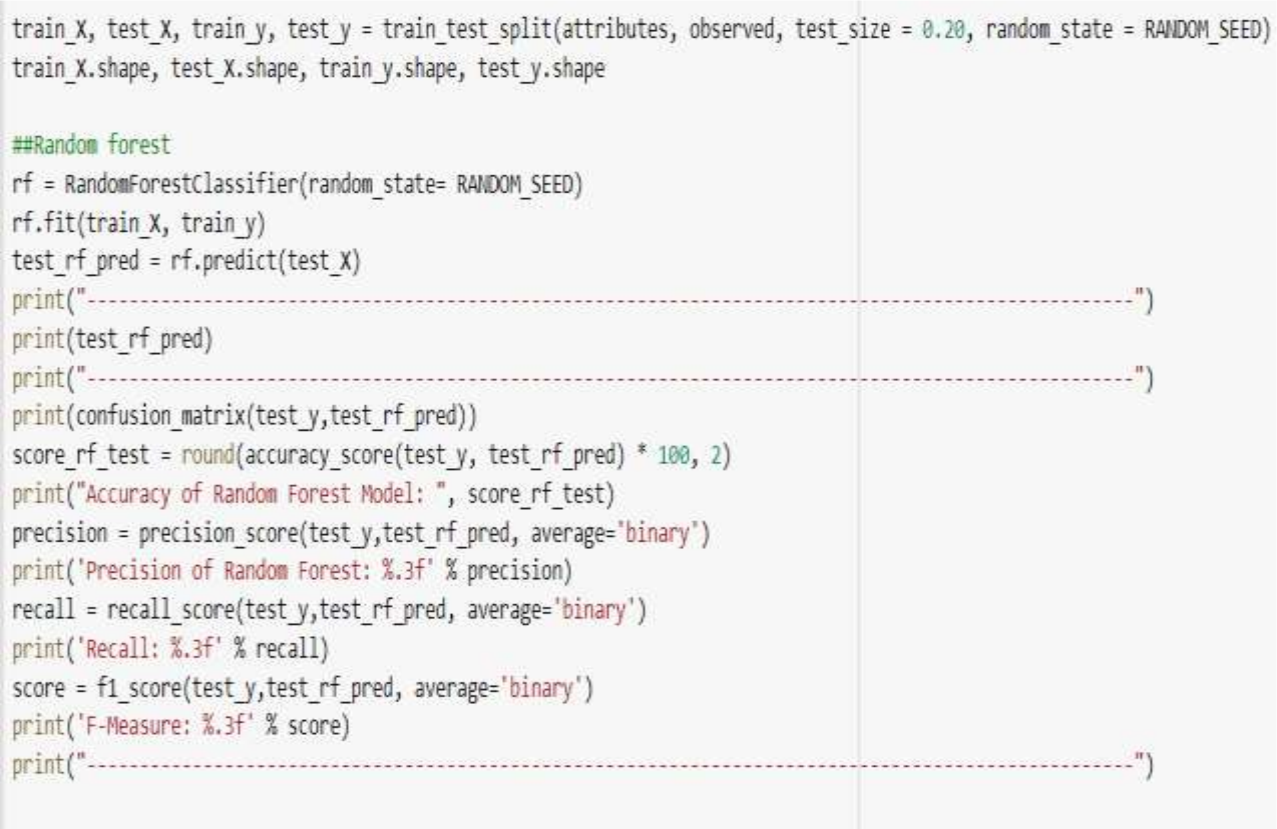
Figure 5.18 represents the dropping of unnecessary columns and checking whether there is any null value and get the independent variables and dependent variables from the data the model.



```
# Drop the unnecessary columns
domain_withFeatures_fixed = domain_withFeatures.drop(drop_column, axis = 1)
print("--------------------------------------------------------------------------------------------------")
print(domain_withFeatures_fixed.head())
print("--------------------------------------------------------------------------------------------------")
print(domain_withFeatures_fixed.describe())

# Checking whether there is null value
print("--------------------------------------------------------------------------------------------------")
print(domain_withFeatures_fixed.isnull().sum())
# Get independent variables and dependent variables
attributes = domain_withFeatures_fixed.drop('Type', axis=1)
observed = domain_withFeatures_fixed['Type']
print("--------------------------------------------------------------------------------------------------")
print(attributes.shape, observed.shape)
```

Figure 5.18 Dropping of unnecessary column

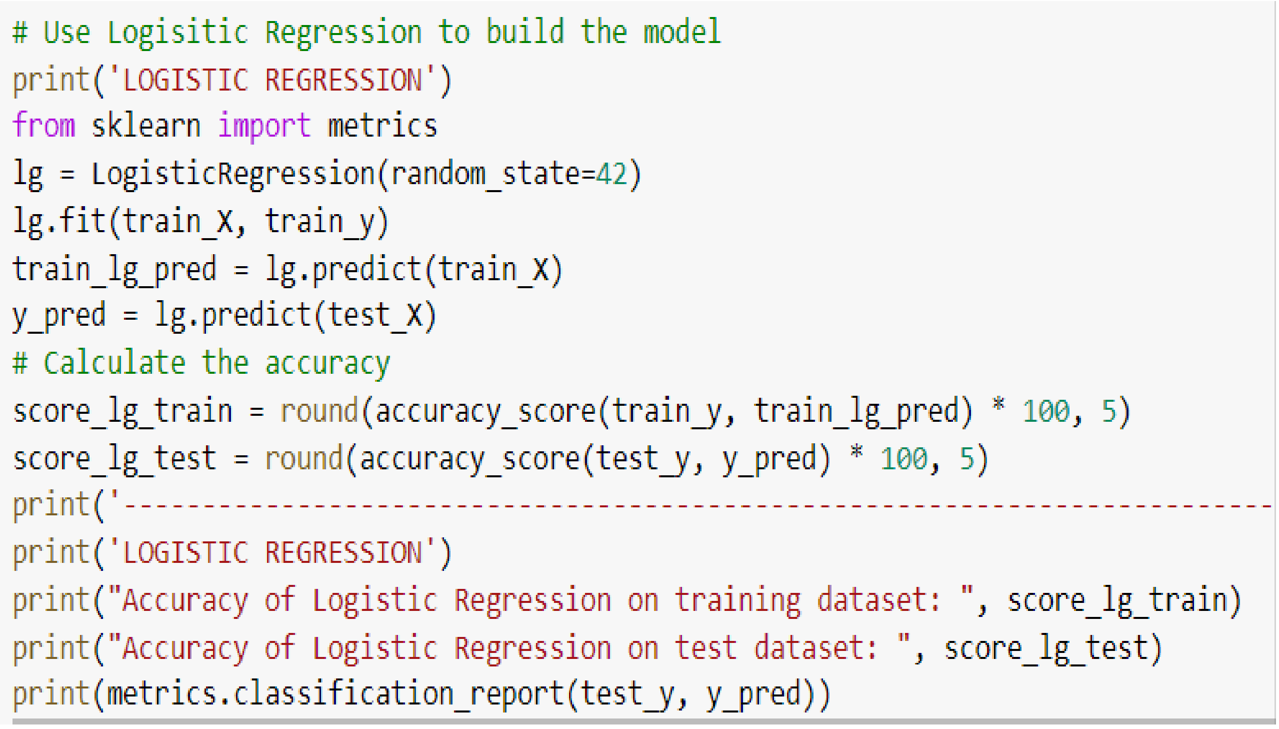
Figure 5.19 represents the random forest algorithm to build the model, initially split the data into training data set and testing dataset by using Train test split function ,train_X and train_Y parameters passed to the random forest model ,the model returns the accuracy, Precision, recall and F-measure.



```
train_X, test_X, train_y, test_y = train_test_split(attributes, observed, test_size = 0.20, random_state = RANDOM_SEED)
train_X.shape, test_X.shape, train_y.shape, test_y.shape

##Random forest
rf = RandomForestClassifier(random_state= RANDOM_SEED)
rf.fit(train_X, train_y)
test_rf_pred = rf.predict(test_X)
print("--------------------------------------------------------------------------------------")
print(test_rf_pred)
print("--------------------------------------------------------------------------------------")
print(confusion_matrix(test_y,test_rf_pred))
score_rf_test = round(accuracy_score(test_y, test_rf_pred) * 100, 2)
print("Accuracy of Random Forest Model: ", score_rf_test)
precision = precision_score(test_y,test_rf_pred, average='binary')
print('Precision of Random Forest: %.3f' % precision)
recall = recall_score(test_y,test_rf_pred, average='binary')
print('Recall: %.3f' % recall)
score = f1_score(test_y,test_rf_pred, average='binary')
print('F-Measure: %.3f' % score)
print("--------------------------------------------------------------------------------------")
```

Figure 5. 19 Represents the random forest algorithm.

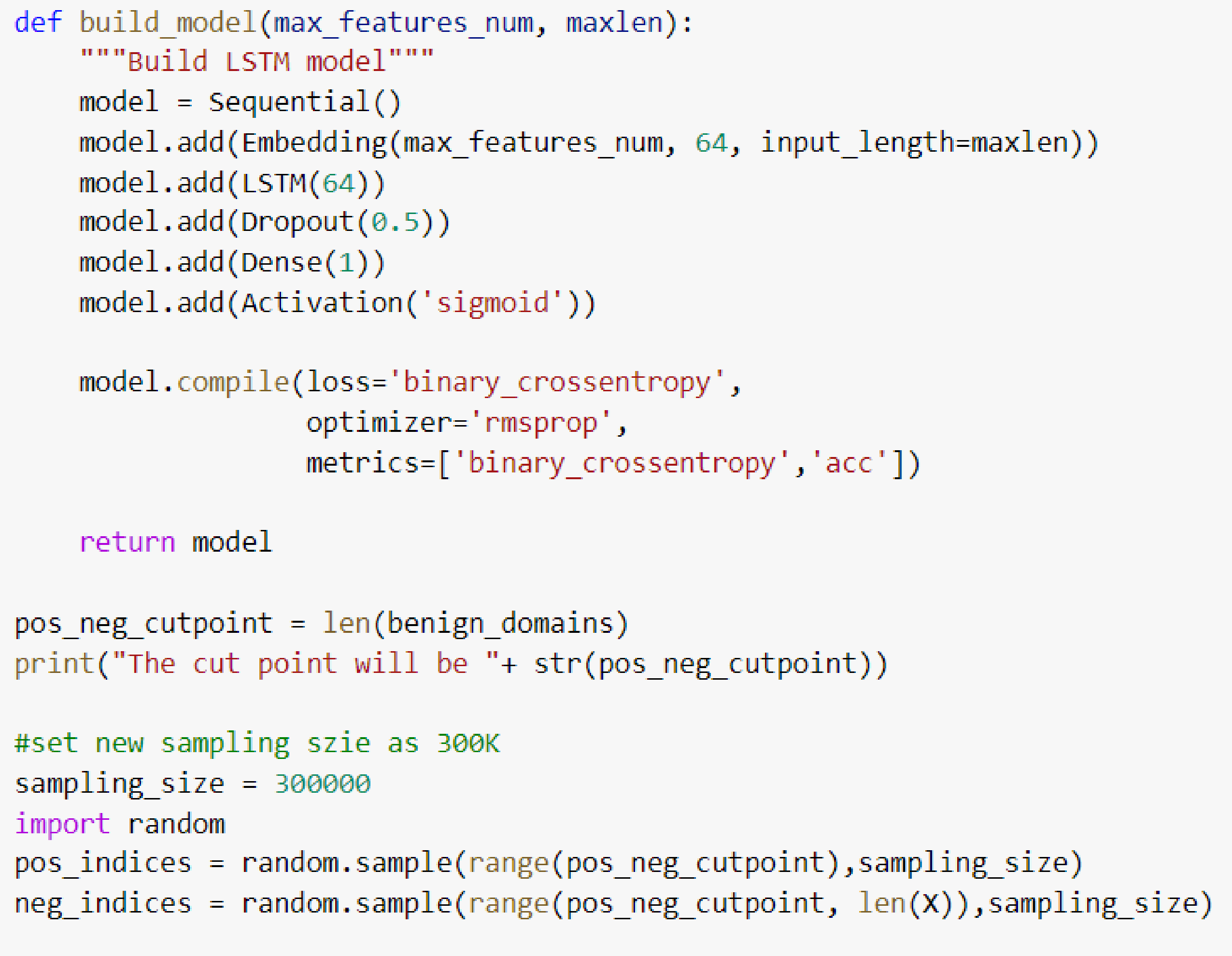
Figure 5.20 represents the Logistic regression algorithm to built the model, initially split the data into training data set and testing dataset by using Train_test_split function, train_X and train_Y parameters passed to the Logistic regression model, and it returns the accuracy of testing and training date set.



```
# Use Logisitic Regression to build the model
print('LOGISTIC REGRESSION')
from sklearn import metrics
lg = LogisticRegression(random_state=42)
lg.fit(train_X, train_y)
train_lg_pred = lg.predict(train_X)
y_pred = lg.predict(test_X)
# Calculate the accuracy
score_lg_train = round(accuracy_score(train_y, train_lg_pred) * 100, 5)
score_lg_test = round(accuracy_score(test_y, y_pred) * 100, 5)
print('-------------------------------------------------------------------------
print('LOGISTIC REGRESSION')
print("Accuracy of Logistic Regression on training dataset: ", score_lg_train)
print("Accuracy of Logistic Regression on test dataset: ", score_lg_test)
print(metrics.classification_report(test_y, y_pred))
```

Figure 5.20 Logistic regression algorithm

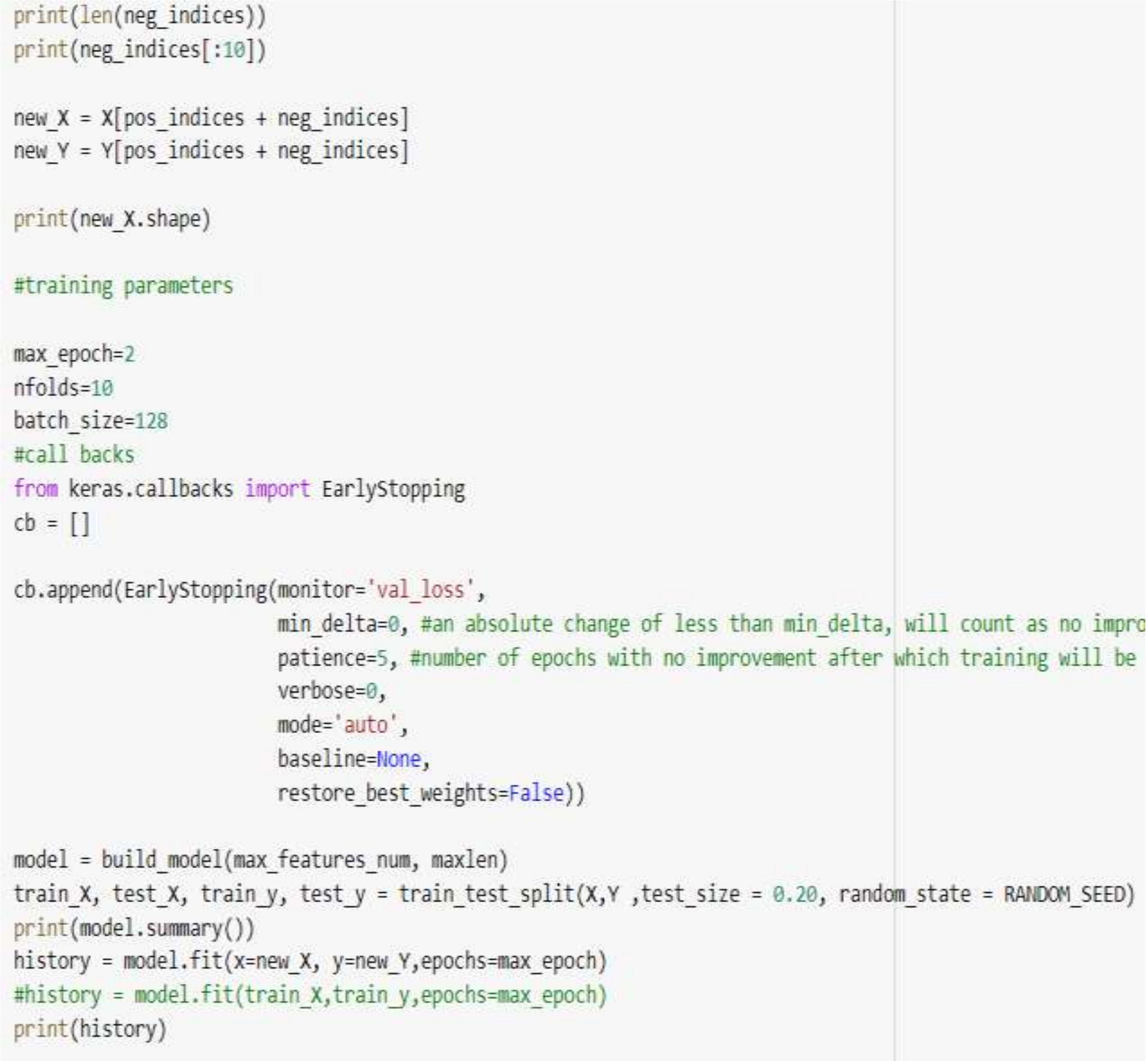
Figure 5.21 shows the CNN model which defines the required layers which are implemented for accurate results.



```
def build_model(max_features_num, maxlen):
    """Build LSTM model"""
    model = Sequential()
    model.add(Embedding(max_features_num, 64, input_length=maxlen))
    model.add(LSTM(64))
    model.add(Dropout(0.5))
    model.add(Dense(1))
    model.add(Activation('sigmoid'))

    model.compile(loss='binary_crossentropy',
                  optimizer='rmsprop',
                  metrics=['binary_crossentropy','acc'])

    return model

pos_neg_cutpoint = len(benign_domains)
print("The cut point will be "+ str(pos_neg_cutpoint))

#set new sampling szie as 300K
sampling_size = 300000
import random
pos_indices = random.sample(range(pos_neg_cutpoint),sampling_size)
neg_indices = random.sample(range(pos_neg_cutpoint, len(X)),sampling_size)
```

Figure 5. 21 Model Initialization

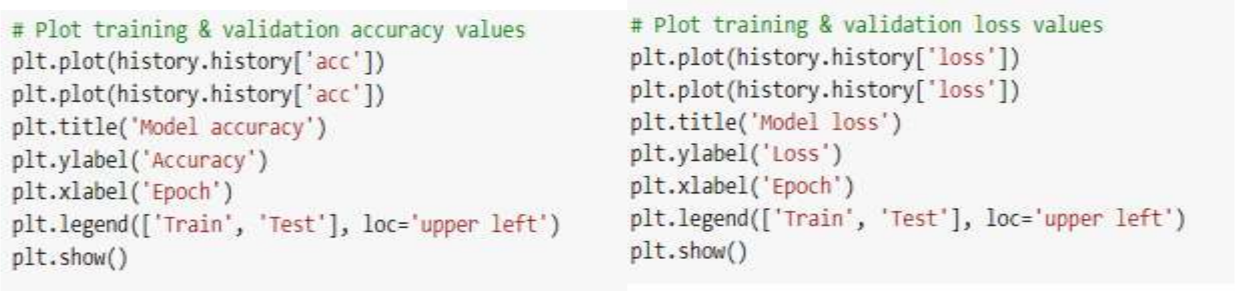
Figure 5.22 represents model fit which is for running the model and fitting into the requirements.



```
print(len(neg_indices))
print(neg_indices[:10])

new_X = X[pos_indices + neg_indices]
new_Y = Y[pos_indices + neg_indices]

print(new_X.shape)

#training parameters

max_epoch=2
nfolds=10
batch_size=128
#call backs
from keras.callbacks import EarlyStopping
cb = []

cb.append(EarlyStopping(monitor='val_loss',
                        min_delta=0, #an absolute change of less than min_delta, will count as no impro
                        patience=5, #number of epochs with no improvement after which training will be
                        verbose=0,
                        mode='auto',
                        baseline=None,
                        restore_best_weights=False))

model = build_model(max_features_num, maxlen)
train_X, test_X, train_y, test_y = train_test_split(X,Y ,test_size = 0.20, random_state = RANDOM_SEED)
print(model.summary())
history = model.fit(x=new_X, y=new_Y,epochs=max_epoch)
#history = model.fit(train_X,train_y,epochs=max_epoch)
print(history)
```

Figure 5. 22 Model fit functionality

Figure 5.23 represents a graph which tells about the performance of the model which depicts a plot between epochs and accuracy.

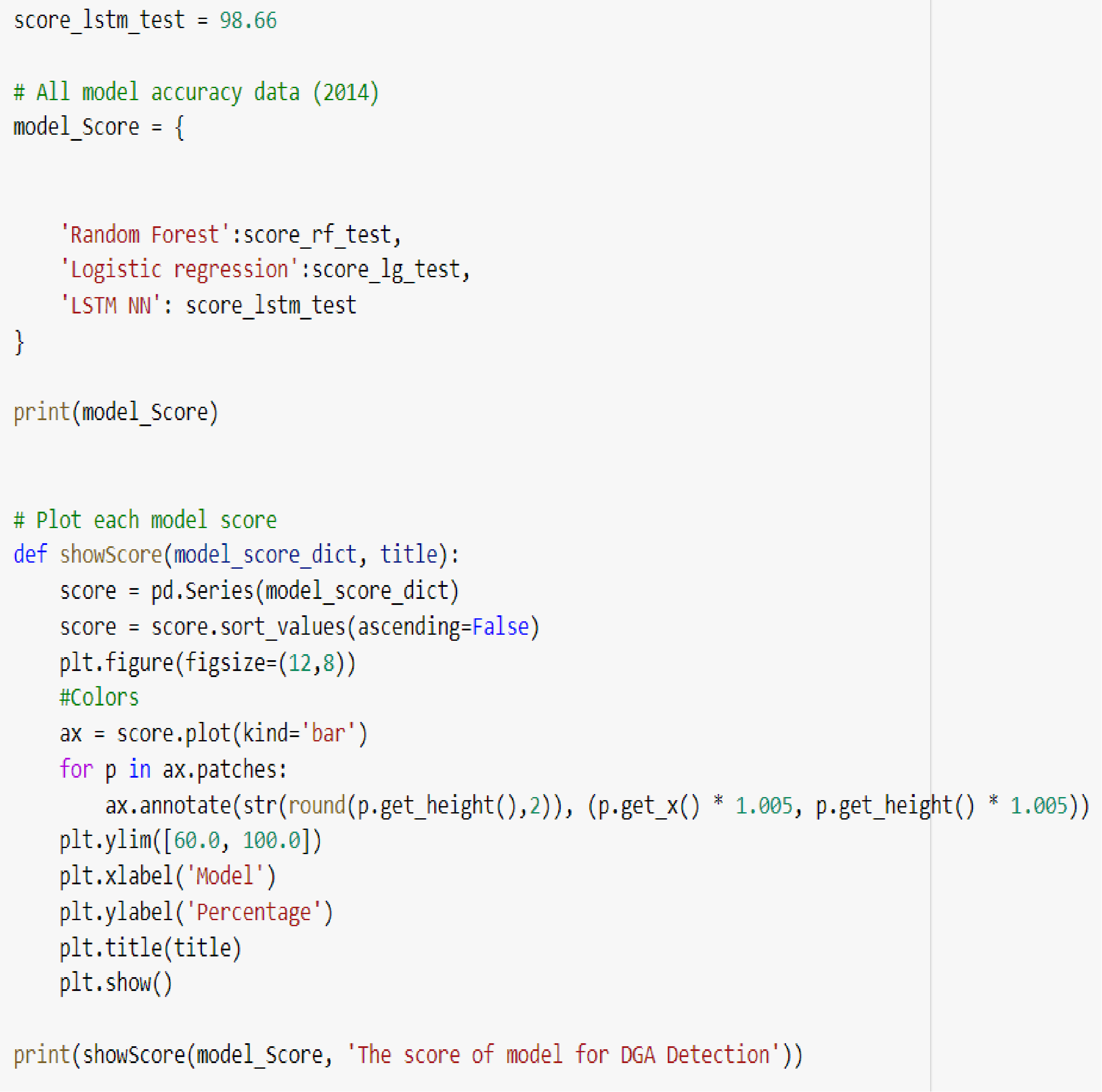


```
# Plot training & validation accuracy values
plt.plot(history.history['acc'])
plt.plot(history.history['acc'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```

```
# Plot training & validation loss values
plt.plot(history.history['loss'])
plt.plot(history.history['loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```

Figure 5.23 Graphs for performance

Figure 5.24 represents the comparative graphs of all the algorithms that are used in building models, and the graph clearly represents that LSTM shows higher accuracy than the random forest algorithm and logistic regression.

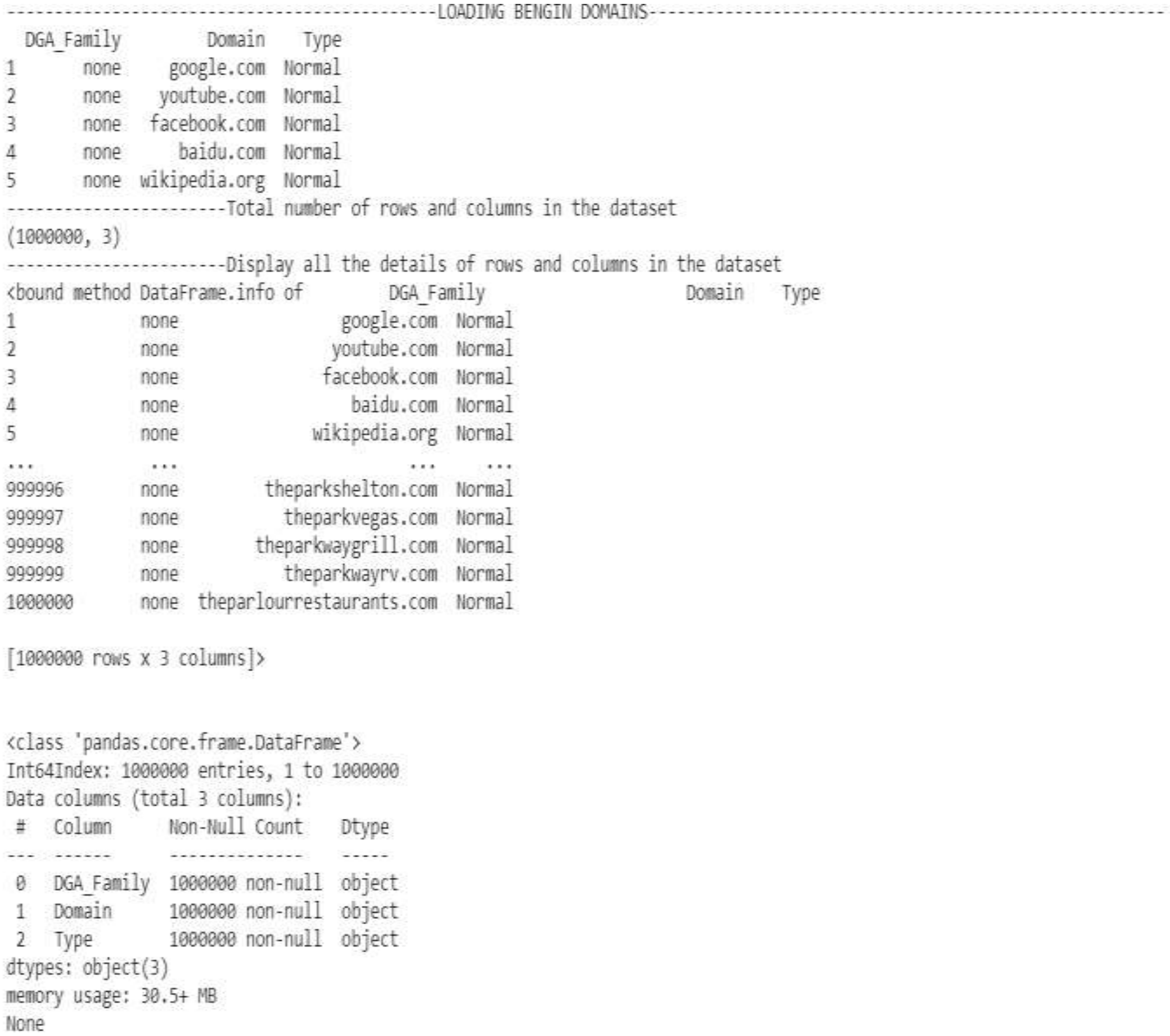


```
score_lstm_test = 98.66

# All model accuracy data (2014)
model_Score = {


    'Random Forest':score_rf_test,
    'Logistic regression':score_lg_test,
    'LSTM NN': score_lstm_test
}

print(model_Score)



# Plot each model score
def showScore(model_score_dict, title):
    score = pd.Series(model_score_dict)
    score = score.sort_values(ascending=False)
    plt.figure(figsize=(12,8))
    #Colors
    ax = score.plot(kind='bar')
    for p in ax.patches:
        ax.annotate(str(round(p.get_height(),2)), (p.get_x() * 1.005, p.get_height() * 1.005))
    plt.ylim([60.0, 100.0])
    plt.xlabel('Model')
    plt.ylabel('Percentage')
    plt.title(title)
    plt.show()

print(showScore(model_Score, 'The score of model for DGA Detection'))
```

Figure 5.24 Graph for Algorithms

6.RESULTS

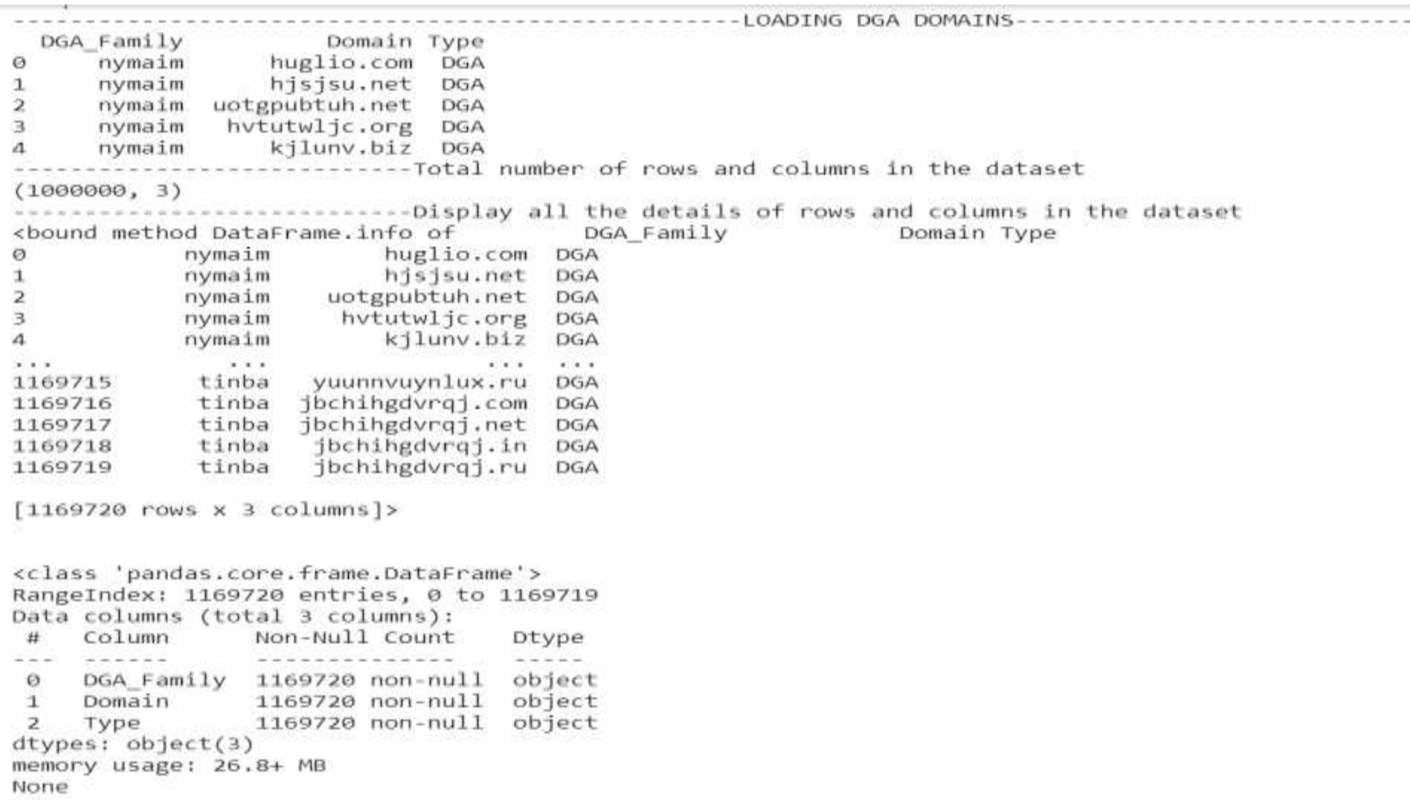
Figure 6.1 represents the Loading of Benign domains dataset from the drive to google collab which are text files initially and converted into csv files.



```
--------------------------------------------LOADING BENGIN DOMAINS--------------------------------------------------
  DGA_Family        Domain    Type
1       none     google.com  Normal
2       none    youtube.com  Normal
3       none   facebook.com  Normal
4       none      baidu.com  Normal
5       none  wikipedia.org  Normal
----------------------Total number of rows and columns in the dataset
(1000000, 3)
----------------------Display all the details of rows and columns in the dataset
<bound method DataFrame.info of        DGA_Family                    Domain    Type
1             none               google.com  Normal
2             none              youtube.com  Normal
3             none             facebook.com  Normal
4             none                baidu.com  Normal
5             none            wikipedia.org  Normal
...            ...                      ...     ...
999996        none        theparkshelton.com  Normal
999997        none           theparkvegas.com  Normal
999998        none        theparkwaygrill.com  Normal
999999        none           theparkwayrv.com  Normal
1000000       none  theparlourrestaurants.com  Normal

[1000000 rows x 3 columns]>


<class 'pandas.core.frame.DataFrame'>
Int64Index: 1000000 entries, 1 to 1000000
Data columns (total 3 columns):
 #   Column      Non-Null Count    Dtype
---  ------      --------------    -----
 0   DGA_Family  1000000 non-null  object
 1   Domain      1000000 non-null  object
 2   Type        1000000 non-null  object
dtypes: object(3)
memory usage: 30.5+ MB
None
```

Figure 6.1 Loading of Benign domains.

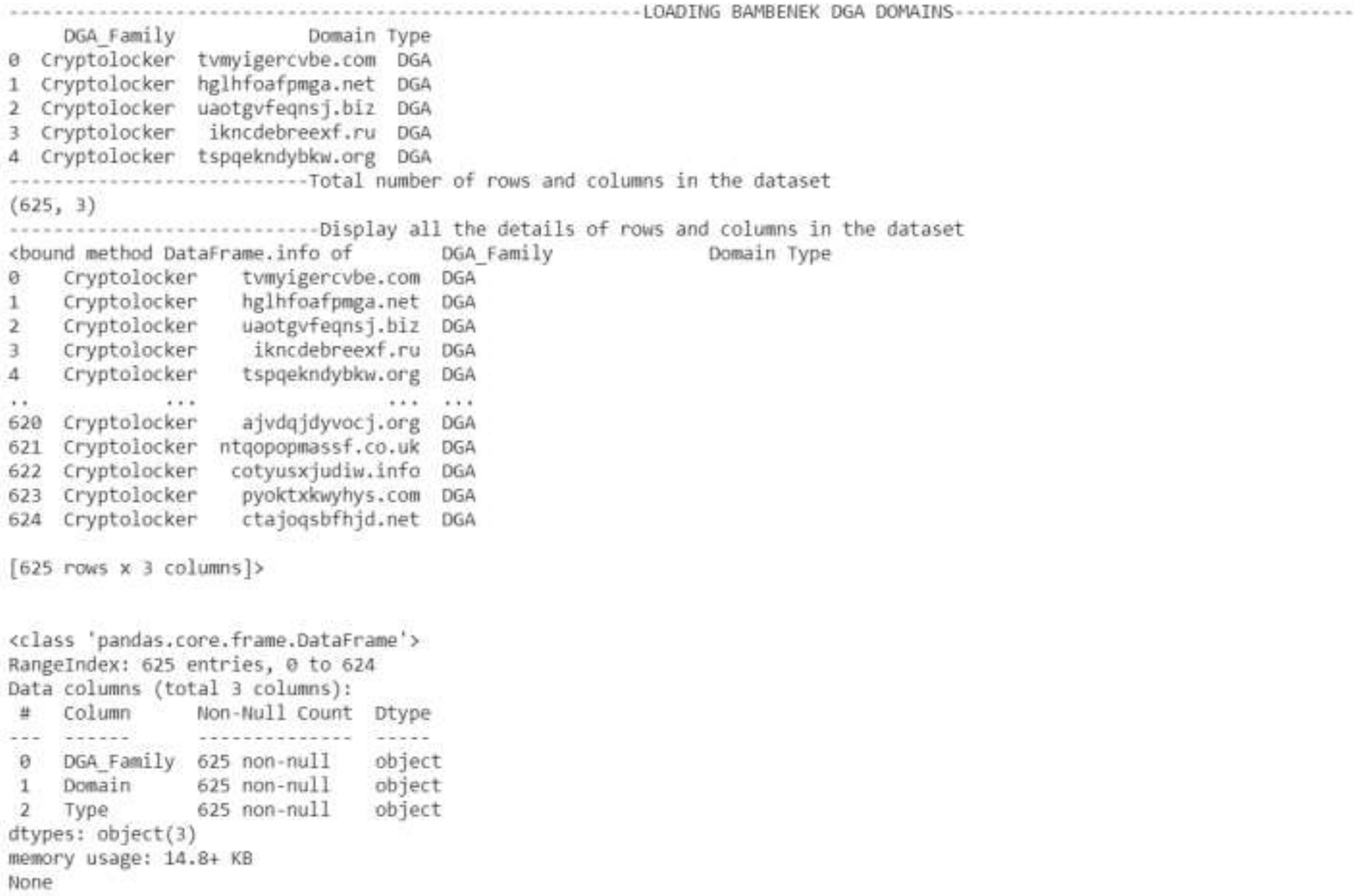
Figure 6.2 represents the Loading of DGA domains dataset from the drive to google collab which are text files initially and converted into csv files.



```
--------------------------------------------------LOADING DGA DOMAINS------------------------------
  DGA_Family           Domain Type
0     nymaim       huglio.com  DGA
1     nymaim       hjsjsu.net  DGA
2     nymaim   uotgpubtuh.net  DGA
3     nymaim   hvtutwljc.org  DGA
4     nymaim       kjlunv.biz  DGA
---------------------------Total number of rows and columns in the dataset
(1000000, 3)
---------------------------Display all the details of rows and columns in the dataset
<bound method DataFrame.info of         DGA_Family            Domain Type
0             nymaim        huglio.com  DGA
1             nymaim        hjsjsu.net  DGA
2             nymaim    uotgpubtuh.net  DGA
3             nymaim     hvtutwljc.org  DGA
4             nymaim        kjlunv.biz  DGA
...              ...               ...  ...
1169715        tinba   yuunnvuynlux.ru  DGA
1169716        tinba  jbchihgdvrqj.com  DGA
1169717        tinba  jbchihgdvrqj.net  DGA
1169718        tinba   jbchihgdvrqj.in  DGA
1169719        tinba   jbchihgdvrqj.ru  DGA

[1169720 rows x 3 columns]>


<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1169720 entries, 0 to 1169719
Data columns (total 3 columns):
 #   Column      Non-Null Count    Dtype
---  ------      --------------    -----
 0   DGA_Family  1169720 non-null  object
 1   Domain      1169720 non-null  object
 2   Type        1169720 non-null  object
dtypes: object(3)
memory usage: 26.8+ MB
None
```

Figure 6.2 Loading DGA domains.

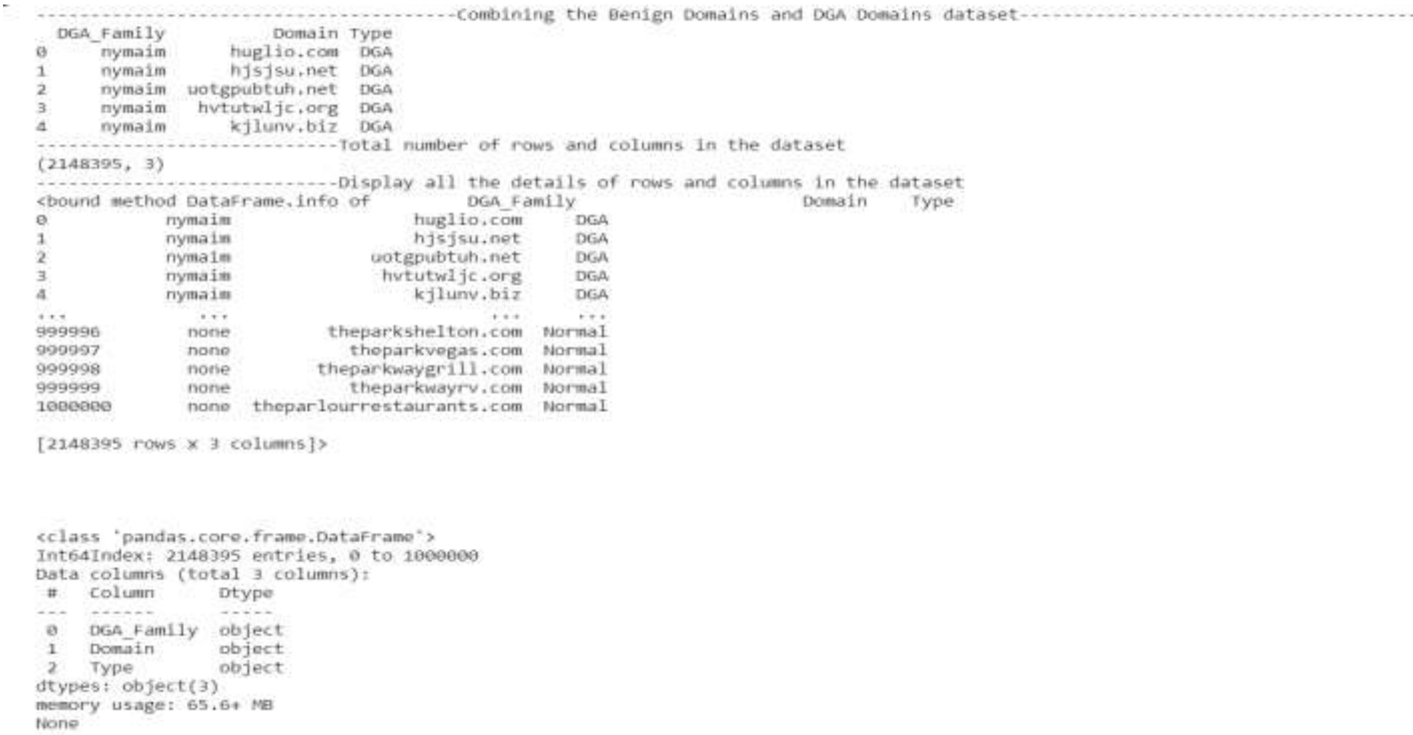
Figure 6.3 represents the Loading of Bambenek DGA domains dataset from the drive to google collab which are text files initially and converted into csv files.



```
-------------------------------------------------------LOADING BAMBENEK DGA DOMAINS----------------------------------
     DGA_Family            Domain Type
0  Cryptolocker  tvmyigercvbe.com  DGA
1  Cryptolocker  hglhfoafpmga.net  DGA
2  Cryptolocker  uaotgvfeqnsj.biz  DGA
3  Cryptolocker   ikncdebreexf.ru  DGA
4  Cryptolocker  tspqekndybkw.org  DGA
--------------------------Total number of rows and columns in the dataset
(625, 3)
---------------------------Display all the details of rows and columns in the dataset
<bound method DataFrame.info of        DGA_Family             Domain Type
0    Cryptolocker    tvmyigercvbe.com  DGA
1    Cryptolocker    hglhfoafpmga.net  DGA
2    Cryptolocker    uaotgvfeqnsj.biz  DGA
3    Cryptolocker     ikncdebreexf.ru  DGA
4    Cryptolocker    tspqekndybkw.org  DGA
..            ...                 ...  ...
620  Cryptolocker    ajvdqjdyvocj.org  DGA
621  Cryptolocker  ntqopopmassf.co.uk  DGA
622  Cryptolocker   cotyusxjudiw.info  DGA
623  Cryptolocker    pyoktxkwyhys.com  DGA
624  Cryptolocker    ctajoqsbfhjd.net  DGA

[625 rows x 3 columns]>


<class 'pandas.core.frame.DataFrame'>
RangeIndex: 625 entries, 0 to 624
Data columns (total 3 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   DGA_Family  625 non-null    object
 1   Domain      625 non-null    object
 2   Type        625 non-null    object
dtypes: object(3)
memory usage: 14.8+ KB
None
```

Figure 6.3 Loading Bambenek DGA domains

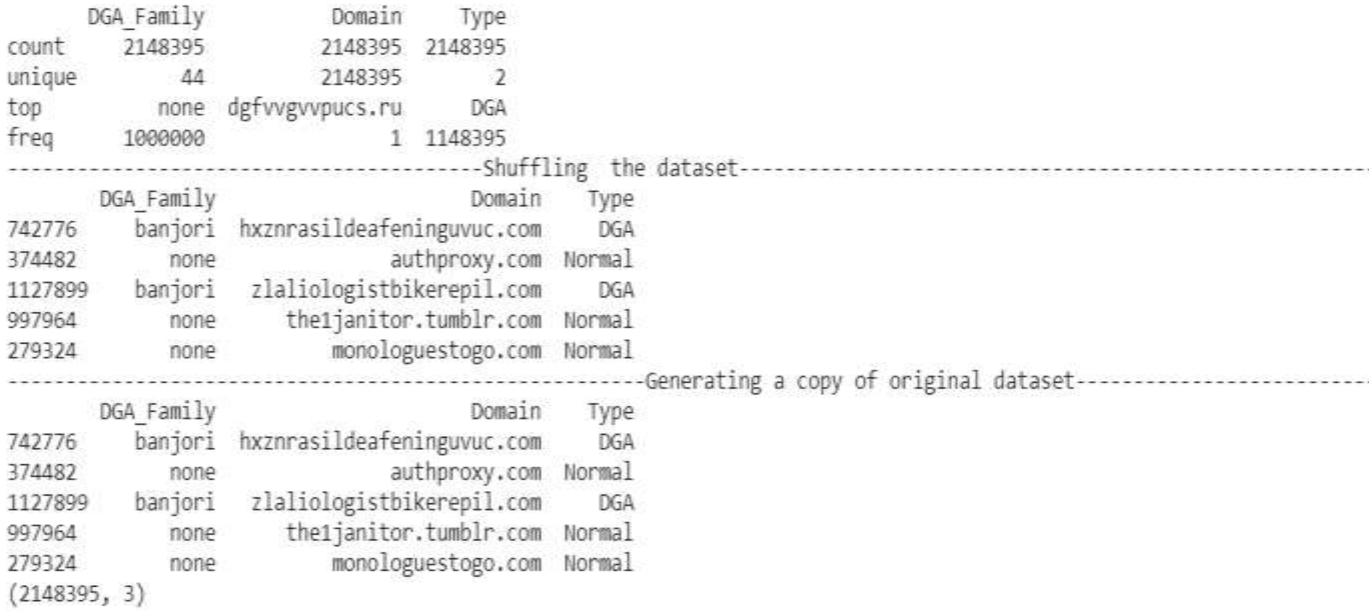
Figure 6.4 represents the results after the Concatenation of DGA domain dataset and BENIGN domains dataset and their total number of rows and columns.



```
----------------------------------------Combining the Benign Domains and DGA Domains dataset----------------------------------------
  DGA_Family          Domain Type
0     nymaim      huglio.com  DGA
1     nymaim      hjsjsu.net  DGA
2     nymaim  uotgpubtuh.net  DGA
3     nymaim   hvtutwljc.org  DGA
4     nymaim      kjlunv.biz  DGA
----------------------------Total number of rows and columns in the dataset
(2148395, 3)
----------------------------Display all the details of rows and columns in the dataset
<bound method DataFrame.info of         DGA_Family                      Domain    Type
0            nymaim                  huglio.com     DGA
1            nymaim                  hjsjsu.net     DGA
2            nymaim              uotgpubtuh.net     DGA
3            nymaim               hvtutwljc.org     DGA
4            nymaim                  kjlunv.biz     DGA
...             ...                         ...     ...
999996         none          theparkshelton.com  Normal
999997         none            theparkvegas.com  Normal
999998         none         theparkwaygrill.com  Normal
999999         none            theparkwayrv.com  Normal
1000000        none  theparlourrestaurants.com  Normal

[2148395 rows x 3 columns]>



<class 'pandas.core.frame.DataFrame'>
Int64Index: 2148395 entries, 0 to 1000000
Data columns (total 3 columns):
 #   Column      Dtype 
---  ------      ----- 
 0   DGA_Family  object
 1   Domain      object
 2   Type        object
dtypes: object(3)
memory usage: 65.6+ MB
None
```

Figure 6.4 Concatenate DGA and BENIGN domains

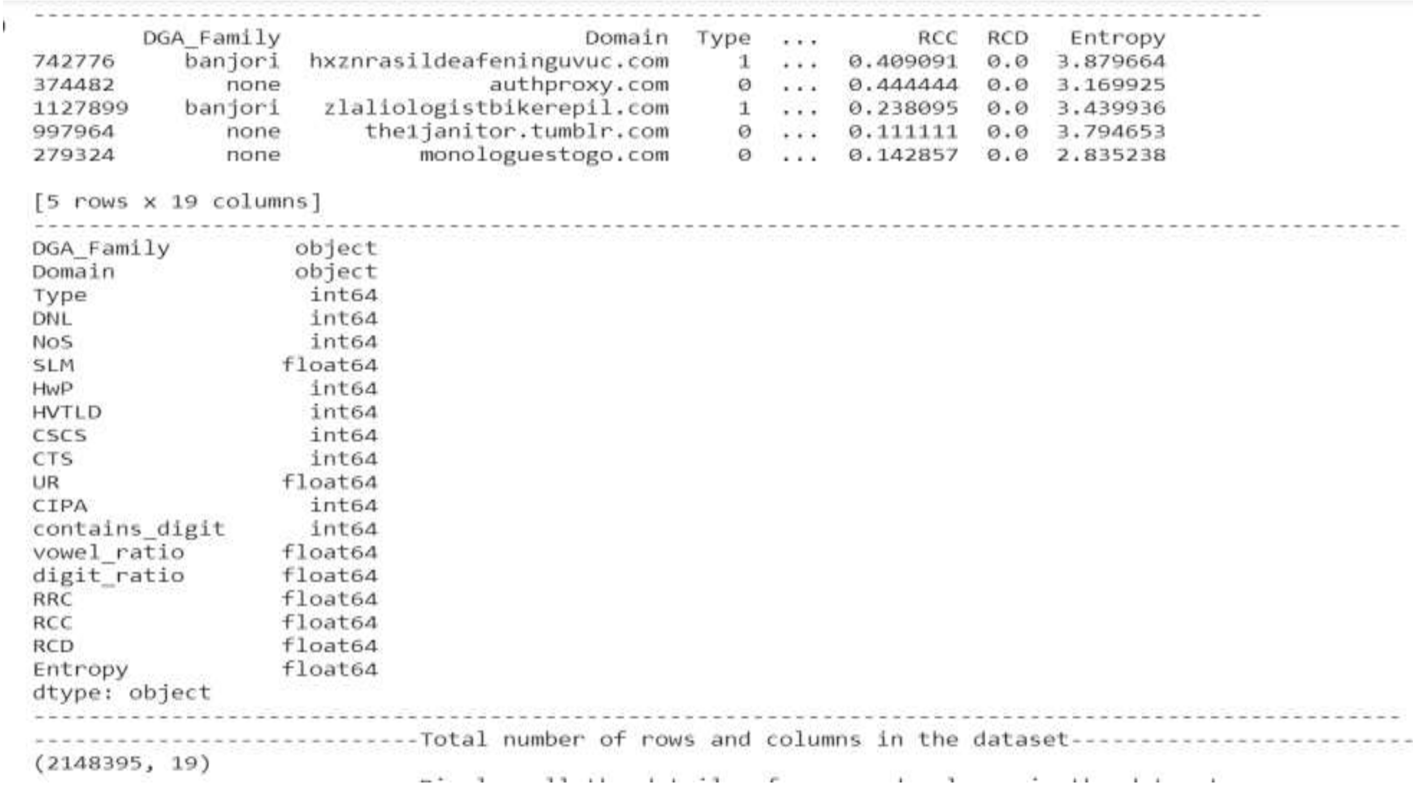
Figure 6.5 represents the data after concatenating, results of the shuffle data and generating the copy of data set to the original data set.



```
       DGA_Family          Domain     Type
count     2148395         2148395  2148395
unique         44         2148395        2
top          none  dgfvvgvvpucs.ru      DGA
freq      1000000               1  1148395
-----------------------------------------Shuffling  the dataset-----------------------------------------------------
        DGA_Family                       Domain    Type
742776     banjori  hxznrasildeafeninguvuc.com     DGA
374482        none               authproxy.com  Normal
1127899    banjori   zlaliologistbikerepil.com     DGA
997964        none        the1janitor.tumblr.com  Normal
279324        none           monologuestogo.com  Normal
-------------------------------------------------------Generating a copy of original dataset-------------------------
        DGA_Family                       Domain    Type
742776     banjori  hxznrasildeafeninguvuc.com     DGA
374482        none               authproxy.com  Normal
1127899    banjori   zlaliologistbikerepil.com     DGA
997964        none        the1janitor.tumblr.com  Normal
279324        none           monologuestogo.com  Normal
(2148395, 3)
```

Figure 6.5 After concatenating the shuffle data

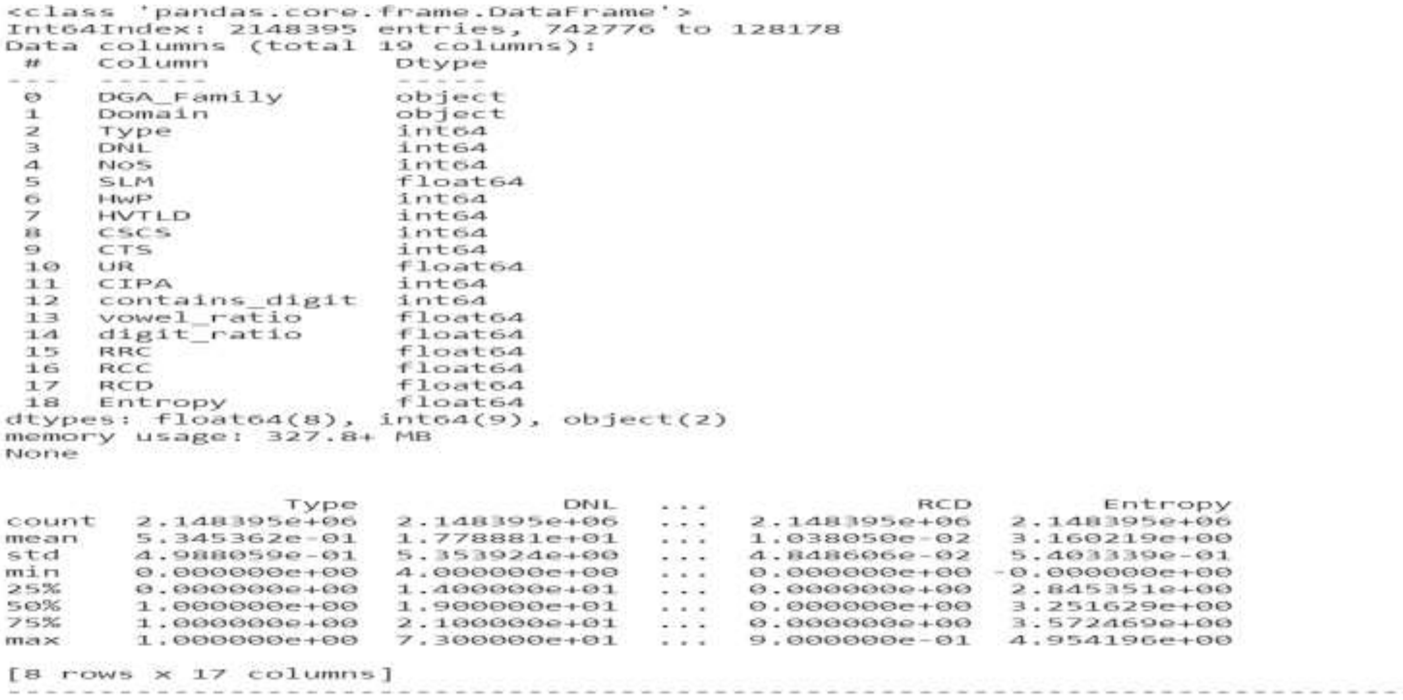
Figure 6.6 represents the Datatypes of extracted features and their total number of rows and columns available in the data.



```
 -----------------------------------------------------------------------------------------
         DGA_Family                        Domain  Type  ...       RCC  RCD   Entropy
742776      banjori  hxznrasildeafeninguvuc.com     1  ...  0.409091  0.0  3.879664
374482         none             authproxy.com     0  ...  0.444444  0.0  3.169925
1127899     banjori   zlaliologistbikerepil.com     1  ...  0.238095  0.0  3.439936
997964         none      the1janitor.tumblr.com     0  ...  0.111111  0.0  3.794653
279324         none         monologuestogo.com     0  ...  0.142857  0.0  2.835238

[5 rows x 19 columns]
---------------------------------------------------------------------------------------------------
DGA_Family          object
Domain              object
Type                 int64
DNL                  int64
NoS                  int64
SLM                float64
HwP                  int64
HVTLD                int64
CSCS                 int64
CTS                  int64
UR                 float64
CIPA                 int64
contains_digit       int64
vowel_ratio        float64
digit_ratio        float64
RRC                float64
RCC                float64
RCD                float64
Entropy            float64
dtype: object
---------------------------------------------------------------------------------------------------
---------------------------Total number of rows and columns in the dataset--------------------------
(2148395, 19)
```

Figure 6.6 Datatype of extracted features

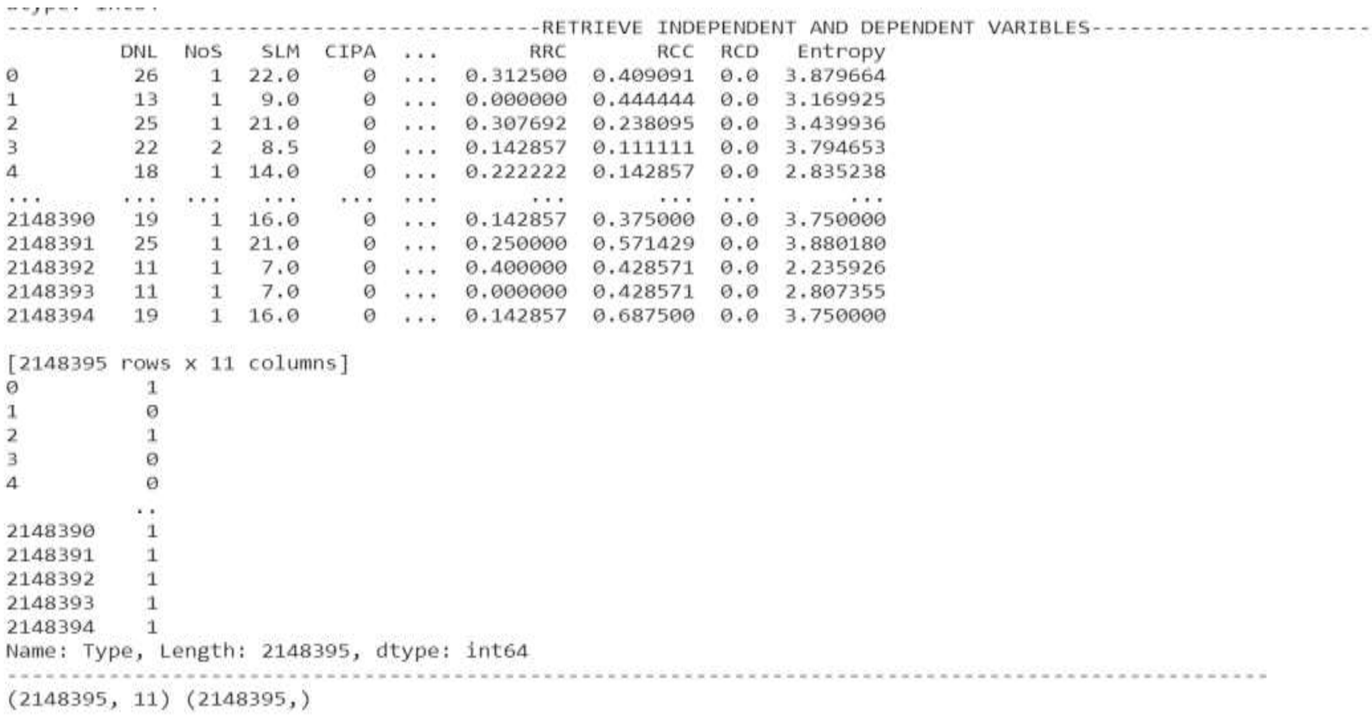
Figure 6.7 represents the Summary statistics of the extracted features from the given data.



```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2148395 entries, 742776 to 128178
Data columns (total 19 columns):
 #   Column          Dtype
---  ------          -----
 0   DGA_Family      object
 1   Domain          object
 2   Type            int64
 3   DNL             int64
 4   NoS             int64
 5   SLM             float64
 6   HwP             int64
 7   HVTLD           int64
 8   CSCS            int64
 9   CTS             int64
 10  UR              float64
 11  CIPA            int64
 12  contains_digit  int64
 13  vowel_ratio     float64
 14  digit_ratio     float64
 15  RRC             float64
 16  RCC             float64
 17  RCD             float64
 18  Entropy         float64
dtypes: float64(8), int64(9), object(2)
memory usage: 327.8+ MB
None

               Type           DNL  ...           RCD       Entropy
count  2.148395e+06  2.148395e+06  ...  2.148395e+06  2.148395e+06
mean   5.345362e-01  1.778881e+01  ...  1.038050e-02  3.160219e+00
std    4.988059e-01  5.353924e+00  ...  4.848606e-02  5.403339e-01
min    0.000000e+00  4.000000e+00  ...  0.000000e+00 -0.000000e+00
25%    0.000000e+00  1.400000e+01  ...  0.000000e+00  2.845351e+00
50%    1.000000e+00  1.900000e+01  ...  0.000000e+00  3.251629e+00
75%    1.000000e+00  2.100000e+01  ...  0.000000e+00  3.572469e+00
max    1.000000e+00  7.300000e+01  ...  9.000000e-01  4.954196e+00

[8 rows x 17 columns]
---------------------------------------------------------------------------------------------------
```

Figure 6.7 Summary statistics of the Domains

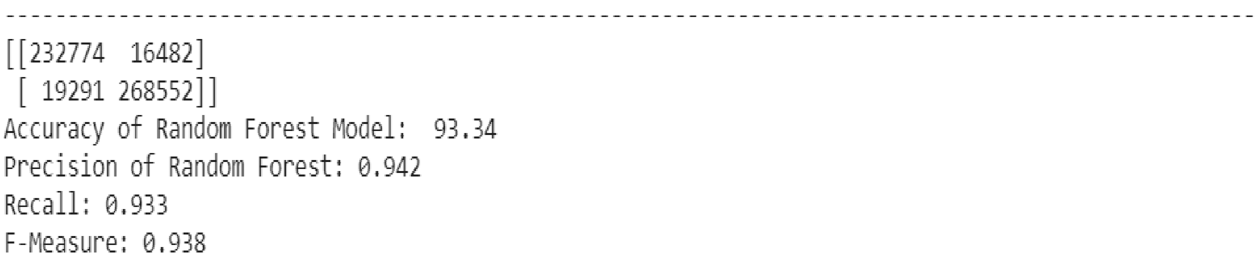
Figure 6.8 represents the Segregation of dependent and independent variables from the extracted features which in the algorithms used to build the models.



```
----------------------------------------RETRIEVE INDEPENDENT AND DEPENDENT VARIBLES----------------------
             DNL   NoS    SLM  CIPA  ...       RRC       RCC  RCD   Entropy
0             26     1   22.0     0  ...  0.312500  0.409091  0.0  3.879664
1             13     1    9.0     0  ...  0.000000  0.444444  0.0  3.169925
2             25     1   21.0     0  ...  0.307692  0.238095  0.0  3.439936
3             22     2    8.5     0  ...  0.142857  0.111111  0.0  3.794653
4             18     1   14.0     0  ...  0.222222  0.142857  0.0  2.835238
...          ...   ...    ...   ...  ...       ...       ...  ...       ...
2148390       19     1   16.0     0  ...  0.142857  0.375000  0.0  3.750000
2148391       25     1   21.0     0  ...  0.250000  0.571429  0.0  3.880180
2148392       11     1    7.0     0  ...  0.400000  0.428571  0.0  2.235926
2148393       11     1    7.0     0  ...  0.000000  0.428571  0.0  2.807355
2148394       19     1   16.0     0  ...  0.142857  0.687500  0.0  3.750000

[2148395 rows x 11 columns]
0          1
1          0
2          1
3          0
4          0
          ..
2148390    1
2148391    1
2148392    1
2148393    1
2148394    1
Name: Type, Length: 2148395, dtype: int64
--------------------------------------------------------------------------------------------------------
(2148395, 11) (2148395,)
```

Figure 6.8 Segregation of dependent and independent variables

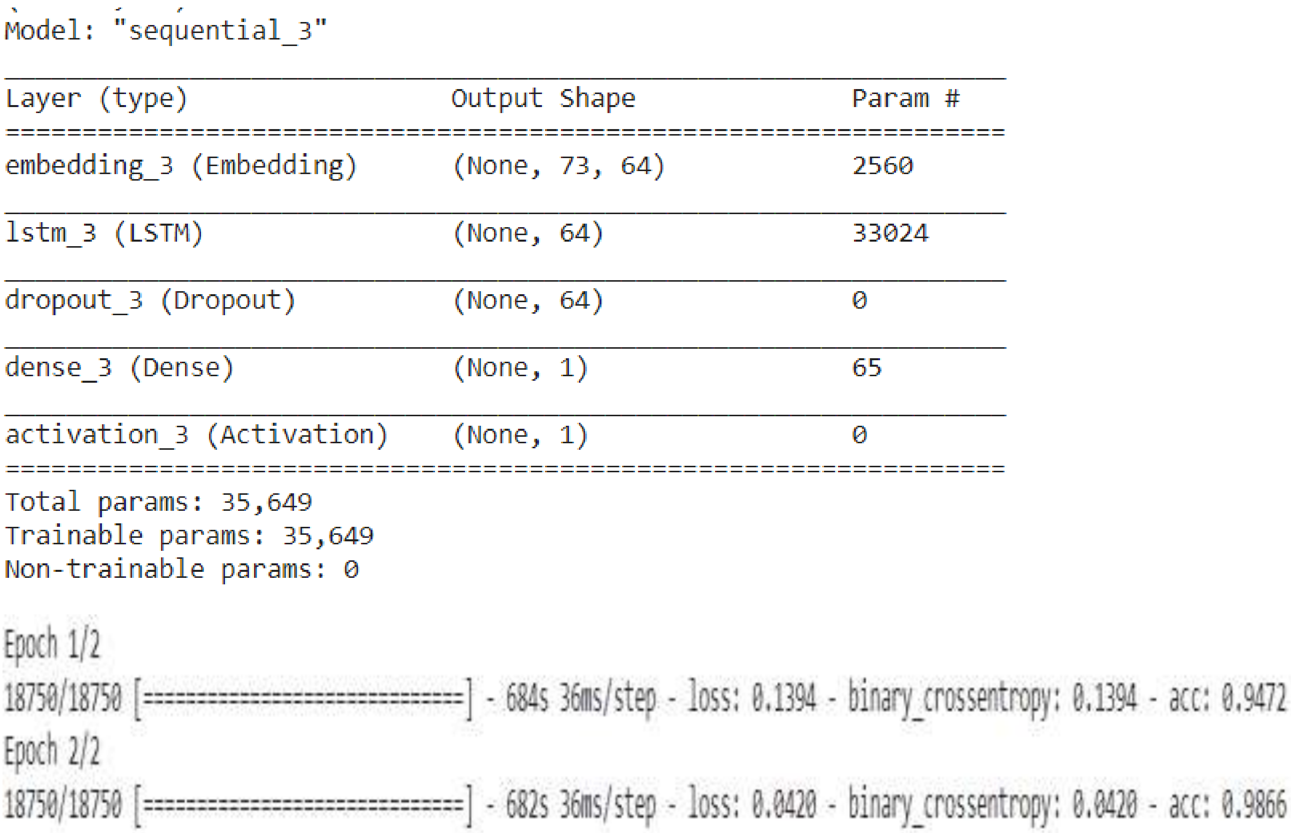
Figure 6.9 represents the output of the Random forest algorithm which gives the accuracy ,recall, precision, f1-measure for the testing data.



```
--------------------------------------------------------------------------------------------------
[[232774  16482]
 [ 19291 268552]]
Accuracy of Random Forest Model:  93.34
Precision of Random Forest: 0.942
Recall: 0.933
F-Measure: 0.938
```

Figure 6.9 Random Forest accuracy

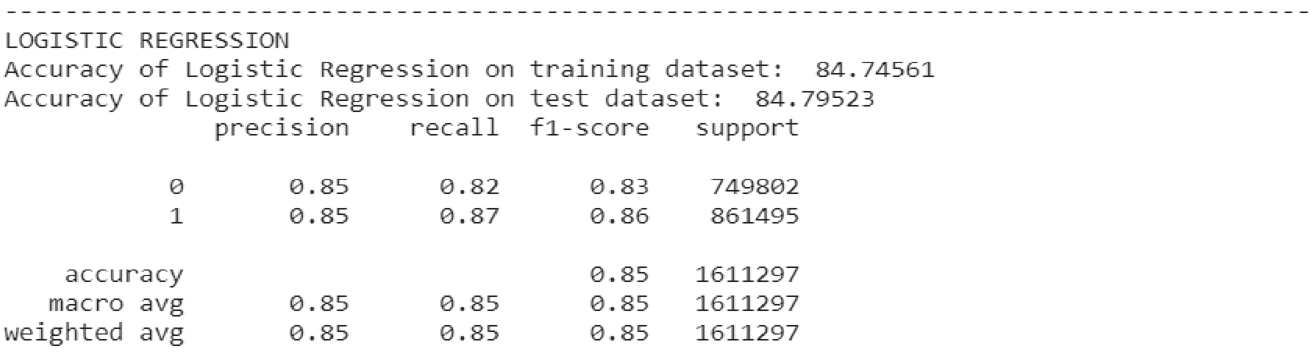
Figure 6.10 represents the overall summary of our model such as layers we used, Number of parameters etc.and model fitting using 2 epochs to get accuracy, loss, binary_ross entropy for testing data.



```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_3 (Embedding)      (None, 73, 64)            2560      
_________________________________________________________________
lstm_3 (LSTM)                (None, 64)                33024     
_________________________________________________________________
dropout_3 (Dropout)          (None, 64)                0         
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 65        
_________________________________________________________________
activation_3 (Activation)    (None, 1)                 0         
=================================================================
Total params: 35,649
Trainable params: 35,649
Non-trainable params: 0

Epoch 1/2
18750/18750 [==============================] - 684s 36ms/step - loss: 0.1394 - binary_crossentropy: 0.1394 - acc: 0.9472
Epoch 2/2
18750/18750 [==============================] - 682s 36ms/step - loss: 0.0420 - binary_crossentropy: 0.0420 - acc: 0.9866
```

Figure 6.10 LSTM accuracy

Figure 6.11 represents the output of the Logistic regression algorithm which gives the accuracy, recall, precision, f1-score and support for the training and testing data.



```
--------------------------------------------------------------------------------------
LOGISTIC REGRESSION
Accuracy of Logistic Regression on training dataset:  84.74561
Accuracy of Logistic Regression on test dataset:  84.79523
              precision    recall  f1-score   support

           0       0.85      0.82      0.83    749802
           1       0.85      0.87      0.86    861495

    accuracy                           0.85   1611297
   macro avg       0.85      0.85      0.85   1611297
weighted avg       0.85      0.85      0.85   1611297
```

Figure 6.11 Logistic Regression accuracy

Figure 6.12 represents the correlation between the individual features which were extracted from the data.

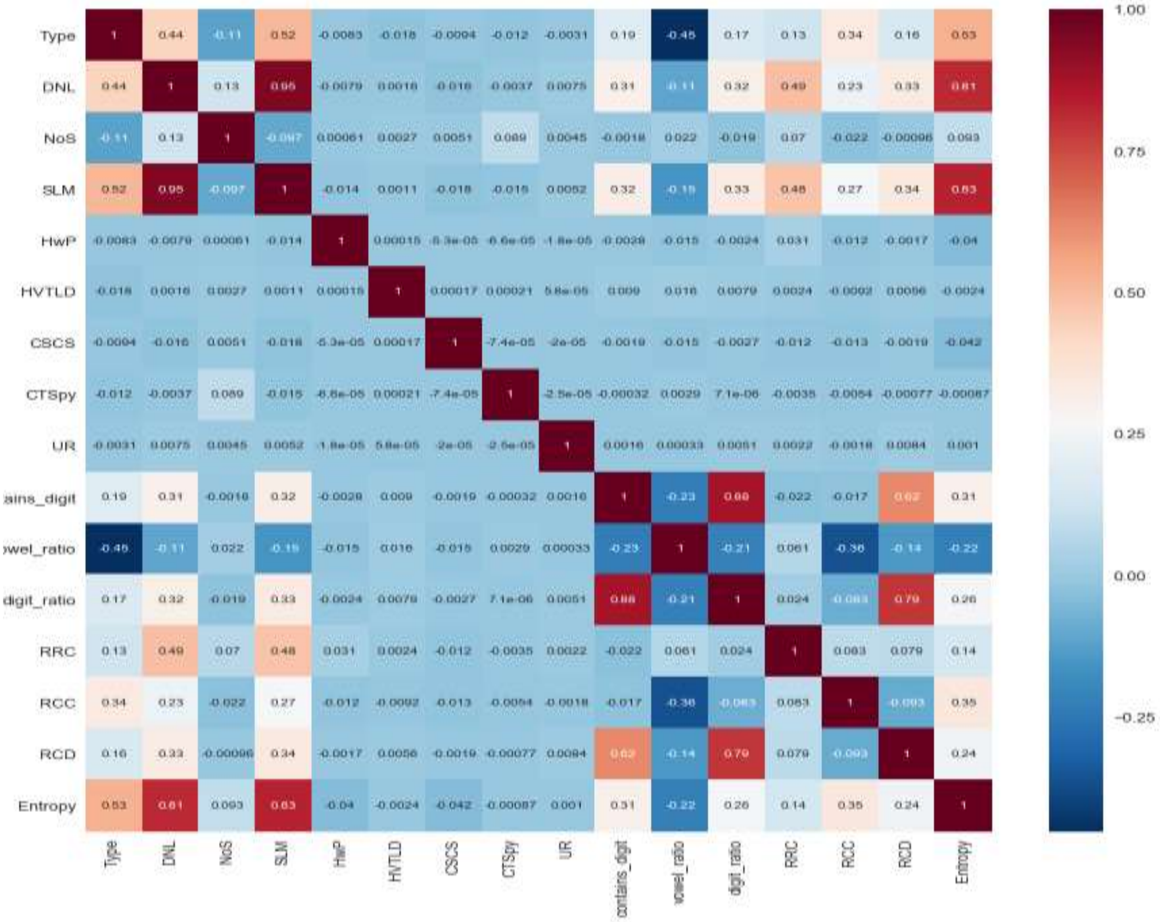


Figure 6.12 Correlation between variables

Figure 6.13 represents the importance of the individual features which were extracted from the data.

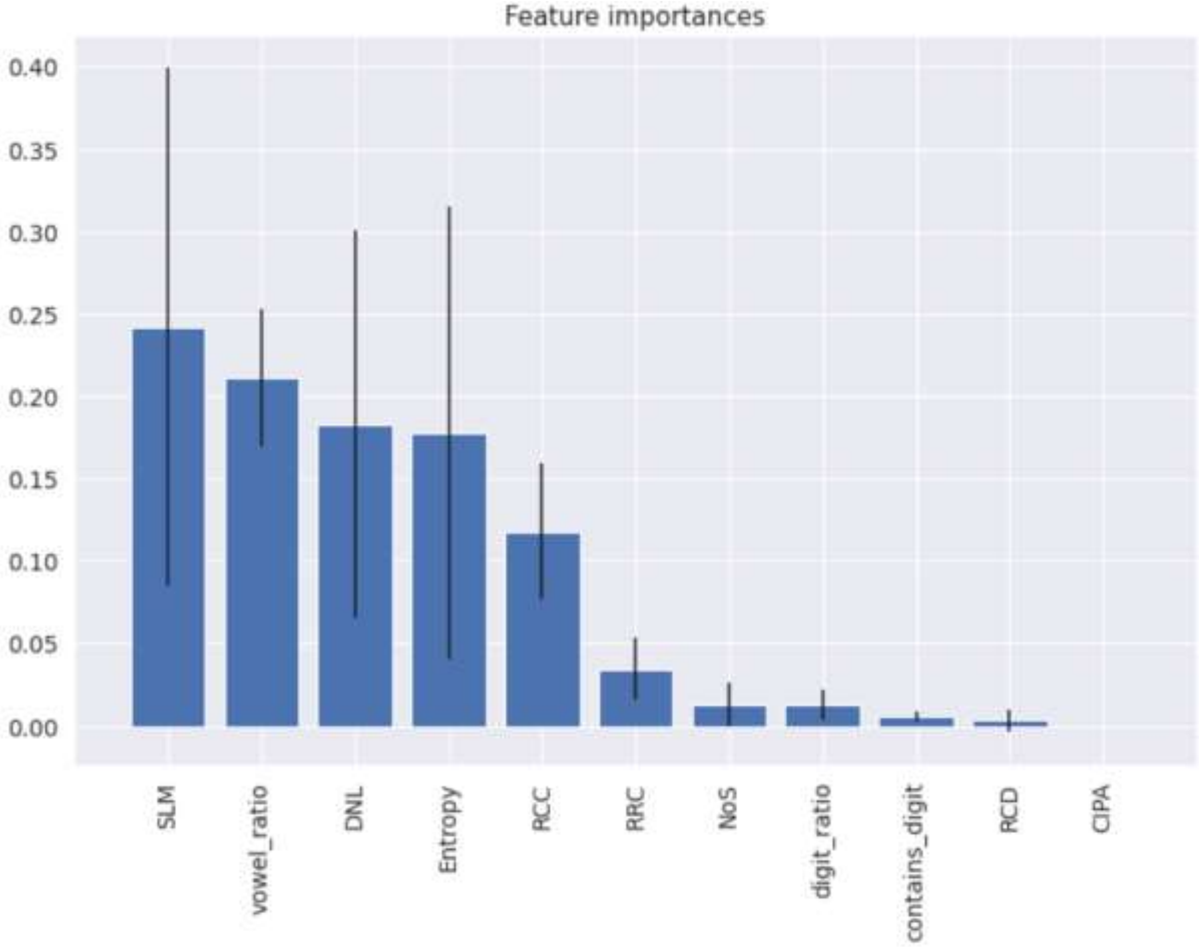


Figure 6.13 Feature Importance

Figure 6.14 represents a graph which tells about the performance of the model which depicts a plot between epoch and Accuracy.

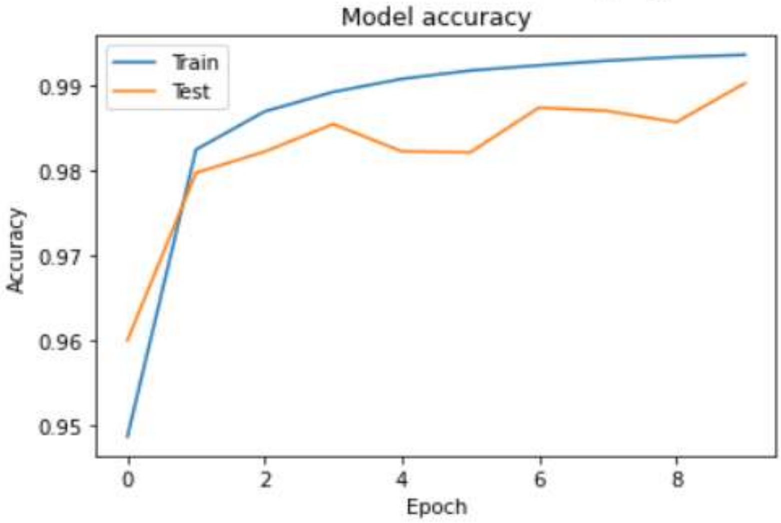


Figure 6.14 Plot training & validation accuracy values

Figure 6.15 represents a graph which tells about the performance of the model which depicts a plot between epoch and Loss.

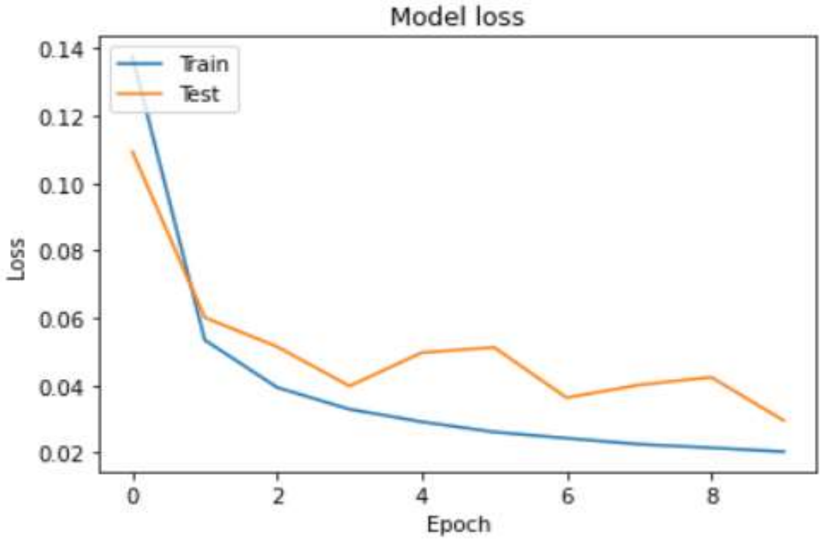
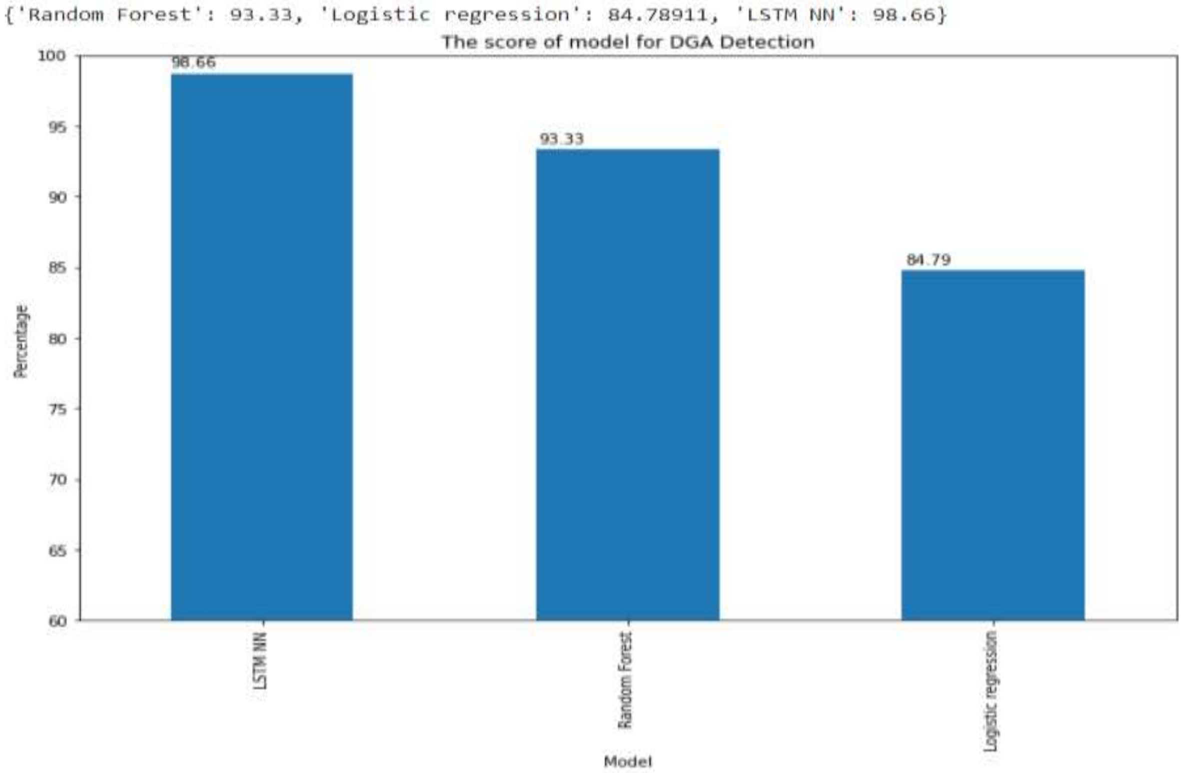


Figure 6.15 Plot training & validation loss values

Figure 6.16 addresses the accuracy of the different algorithms. LSTM has the highest accuracy comparatively random forest and Logistic regression.



```
{'Random Forest': 93.33, 'Logistic regression': 84.78911, 'LSTM NN': 98.66}
```

Figure 6.16 Graph for used algorithm's

7.CONCLUSION AND FUTURE SCOPE

Detecting DGAs is a grand challenge in security areas. Blacklisting is good for handling static methods. However, DGAs are usually used by an attacker to communicate with a variety of servers. They are dynamic, so simply using the blacklisting is not sufficient for detecting a DGA. In this research, we have proposed the machine learning framework with the development of a deep learning model to handle DGA threats. The proposed machine learning framework consists of a feature extractor, and a machine learning model for classification and prediction.The machine learning algorithms were used to get the final output.

Our future research will work towards implementing this model for holding large sets of data. As the size of the data we collected becomes larger and larger, the machine learning model cannot give accurate accuracy, to resolve this , we have to built a deep learning model to perform the classification, which has a better performance than the machine learning algorithms. Based on our extensive experiments on the real-world feed, we have shown that the proposed framework can effectively extract domain name features as well as classify and detect domain names where it belongs.

BIBLIOGRAPHY

[1] YI LI1, Kaiqi Xiong 1, (Senior Member, IEEE), Tommy chin 2, (Member, IEEE), and Chengbin hu1, A Machine Learning Framework for Domain Generation Algorithm-Based Malware Detection

[2] Xingguo Li 1, Junfeng Wang 2 and Xiaosong Zhang 3 Botnet Detection Technology Based on DNS

[3] Dingkui Yan, Huilin Zhang, Yipeng Wang, Tianning Zang, Xiaolin Xu, Yuwei Zeng, Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China, Pontus: A Linguistics-based DGA Detection System

[4] L. I. U. Wu, L. I. U. Ke, R. E. N. Ping, and D. Hai-Xin, "Study and Detection of malware based on behavior," no. 60203044, pp. 39–42, 2011

[5] J. Demme et al., "On quality counter on online malware detection feasibility," Proc. Int.Int Symp. Symp. Software. Computer. Architecture, pp. 559–570, 2013

[6] D. R. Ellis, K. S. Attwood, J. G. Aiken, and S. D. Tenaglia, "A worm identification strategy," WORM' 04-Proc. 2004 Activities of the ACM. Rapid Malcode, pp. 43–53,
2004.

[7] M. I. Gorelik, N. Abu-Ghazaleh, and D. Ozsoy, C. Donovick. Ponomarev, "Malware Adware processors: an active online malware detection system," IEEE 21st Int 2015. Symp.Symp. Strong Perform. Software. Computer. Archit. 2015 HPCA, pp. 651–661, 2015.

[8] R. Sharp, "Malware Introduction," Netw. Safe. Secure. Laboratory check, pp. 331–363,2015.

[9] J. Blount, Tauritz, and Tauritz, and S. A. Mulder, "Adaptive rule-based malware detection using training classification systems: concept proof," Proc. Int. Int. Software.Computer. Softw. Appl. For instance, pp. 110–115, 2011.