TECH MAHINDRA

Tech Mahindra, Hyderabad, Telangana, India

A REPORT ON

NETWORK ANOMALY DETECTION USING DATA SCIENCE

BY

VNS Keerthana Sudina

(179303166)

Department of Computer and Communication Engineering

MANIPAL UNIVERSITY JAIPUR

ACKNOWLEDGEMENTS

The technical internship opportunity I had at EMEA Oneness Lab at Tech Mahindra Ltd was a great chance of learning and professional development. This Internship has helped me gain a real-time experience of working in an office. I perceive this opportunity as a big milestone in my career development. I will strive to use gained skills and knowledge in the best possible way, and I will continue to work on their improvement, to attain desired career objectives.

I would like to express my deepest gratitude to my industry mentor, Mr. Srikanth Rao Amarachinta, Project Manager, for sharing his invaluable experience and guidance and also for allowing me to do an internship at their esteemed organization.

I am highly indebted to Technical Supervisors Mr. Vijay Kumar Botla, Mr. Banudeva Reddy Bandi and Ms. Arpitha Ravi for their guidance and constant supervision as well as for providing necessary information regarding the project & also for their support in completing the project.

I would also acknowledge my college for enabling us to go through an industrial internship to get hands-on experience of working in a professional environment

TABLE OF CONTENTS:

TABLE OF FIGURES

ABSTRACT

This report is related to the project being worked upon in the duration of the Technical Internship Program. The project is based on network anomaly detection. A dataset KDDCUP'99 is given by the organization, which is widely used as one of the few publicly available data sets for network-based anomaly detection systems.

This project is done by using Data Science with python along with libraries such as Sklearn, Numpy, and Pandas. We have to develop datasets for training and testing purposes. Analyze various learning techniques like classification, association, and clustering to build the model. The model is trained using a train dataset in such a way that it tries to predict further outcomes, and is tested with the testing part of the dataset. In this project, the dataset is analyzed and processed. Later on by using Logistic regression, Random forest, and Decision tree the model fit and used to predict if the network is being attacked or not to a desirable accuracy.

PROBLEM STATEMENT

To analyze the Network dataset and build a network anomaly detection system to detect anomalies and attacks in the Network by using Binomial Classification (Activity is normal or attack).
Note: Classifying different types of attack classes as attack.

INTRODUCTION

Company Profile

Tech Mahindra represents the connected world, offering innovative and customer-centric information technology experiences, enabling Enterprises, Associate and the Society to Rise™. **Tech Mahindra Limited** Indian multinational provider of Information Technology (IT), networking technology solutions, Integrated Engineering Solutions(IES) and Business Process Outsourcing (BPO) to various industry verticals and horizontals.

It is a subsidiary of the Mahindra Group; Tech Mahindra is a USD 4.9 billion company with 121,000+ professionals across 90 countries, helping 938 global customers including Fortune 500 companies. Tech Mahindra is the highest ranked Non-U.S. company in the Forbes Global Digital 100 list (2018) and in the Forbes Fab 50 companies in Asia (2018).

Network Anomaly Detection and attack classes

With the enormous growth of computer networks usage and the huge increase in the number of applications running on top of it, network security is becoming increasingly more important. All the computer systems suffer from security vulnerabilities which are both technically difficult and economically costly to be solved by the manufacturers. Therefore, the role of network anomaly detection, as special-purpose devices to detect anomalies and attacks in the network, is becoming more important.

Network behavior anomaly detection (NBAD) is the continuous monitoring of a proprietary network for unusual events or trends. NBAD is an integral part of network behavior analysis (NBA), which offers an additional layer of security to that provided by traditional anti-threat applications such as firewalls, antivirus software, and spyware detection software.

Types of attack classes that are to be detected by the model built:

- Denial of Service (DoS): An attacker tries to prevent legitimate users from using a service. For example: SYN flood, Smurf and teardrop.
- User to Root (U2R): An attacker has local access to the victim machine and tries to gain super-user privilege. For example: buffer overflow attacks.
- Remote to Local (R2L): An attacker tries to gain access to the victim machine without having an account on it. For example: password guessing attack.
- Probe: An attacker tries to gain information about the target host. For example: port-scan and ping-sweep.

Data Science

Data science is the study of data. It involves developing methods of recording, storing, and analyzing data to effectively extract useful information. The goal of data science is to gain insights and knowledge from any type of data (structured or unstructured). Data Science can also be defined as extraction, preparation, analysis, visualization, and maintenance of information. It is a cross-disciplinary field that uses scientific methods and processes to draw insights from data.

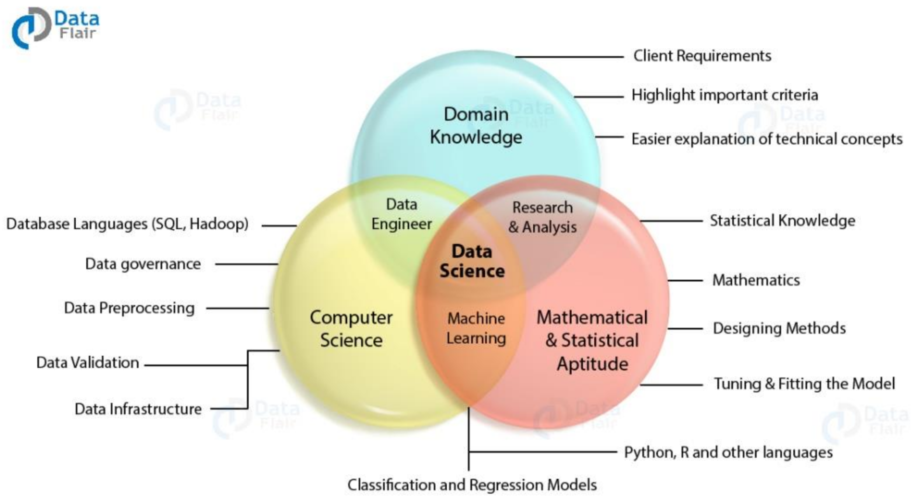


Figure 1: Data science

Python

Python is a high-level language used for general-purpose programming. It is a dynamic language that supports both structured programmings as well as object-oriented programming.

Unlike C and Java, Python focuses on readability. Hence it attracts lots of developers and also has a very large developer community which in turn brings out large support to everyone. Also, Python has a very large library, which eases lots of tasks.

Why python is used in Data Science?

The below points show how Python helps in each step of data analyses:

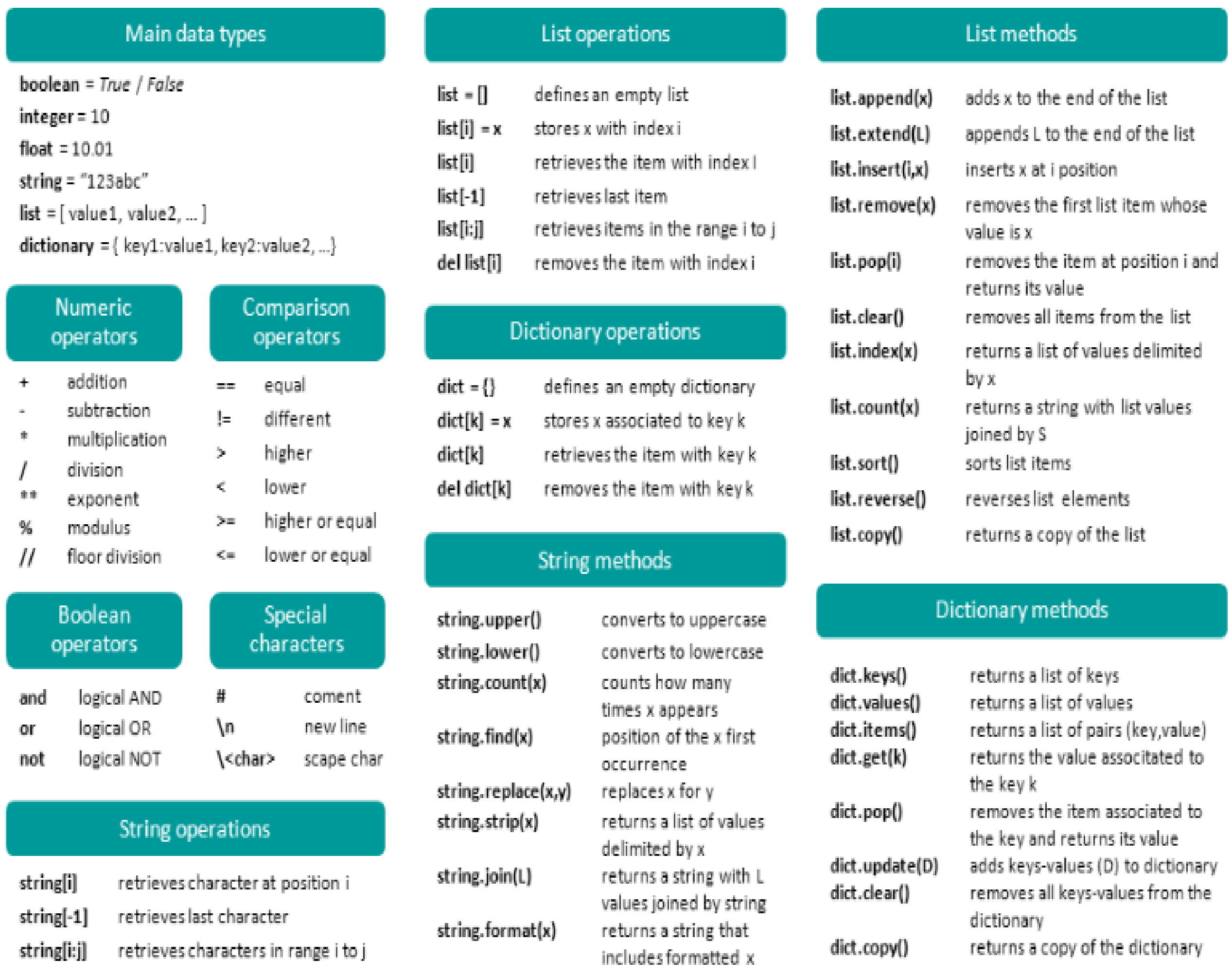
- Assume data as a very huge Excel sheet with a large number of rows and columns. From these large data, we derive insights by performing some operations and searching for a particular type of data in each column and row. Performing such high computational tasks can be cumbersome and extremely time-consuming. Hence Python provides libraries like **Numpy** and **Pandas** which eases this task by the use of *parallel processing*.

- Not always we have data readily available to us. Sometimes we need to scrap the data from the web. Here Python has libraries like **beautifulsoup** and **scrapy** which help extract data from the internet.
- Seeing so many numbers all over screen might turn out to be a big headache sometimes and difficult to derive insights. The only way of doing this is to represent the data in the form of figures like bar graphs, histograms, and pie-charts. Python has libraries for this too. For this, we use libraries like **Matplotlib** and **Seaborn**.
- Machine learning is an **incredibly high computational** technique that involves heavy mathematics like calculus, probability and matrix operations over thousands of rows and columns. All this becomes very simple and efficient with the help of **scikit-learn**, a machine learning library in python.

PROJECT METHODOLOGY

Understanding of Python

Data science with python involves learning of basics such as types, expressions, and variables, string operations, lists, tuples, sets, dictionaries, conditions and branching, loops, functions, objects and classes, reading and writing files.



Main data types

boolean = *True* / *False*
integer = 10
float = 10.01
string = "123abc"
list = [value1, value2, ...]
dictionary = { key1:value1, key2:value2, ...}

Numeric operators

+	addition
-	subtraction
*	multiplication
/	division
**	exponent
%	modulus
//	floor division

Comparison operators

==	equal
!=	different
>	higher
<	lower
>=	higher or equal
<=	lower or equal

Boolean operators

and	logical AND
or	logical OR
not	logical NOT

Special characters

#	coment
\n	new line
\<char>	scape char

String operations

string[i]	retrieves character at position i
string[-1]	retrieves last character
string[i:j]	retrieves characters in range i to j

List operations

list = []	defines an empty list
list[i] = x	stores x with index i
list[i]	retrieves the item with index I
list[-1]	retrieves last item
list[i:j]	retrieves items in the range i to j
del list[i]	removes the item with index i

Dictionary operations

dict = {}	defines an empty dictionary
dict[k] = x	stores x associated to key k
dict[k]	retrieves the item with key k
del dict[k]	removes the item with key k

String methods

string.upper()	converts to uppercase
string.lower()	converts to lowercase
string.count(x)	counts how many times x appears
string.find(x)	position of the x first occurrence
string.replace(x,y)	replaces x for y
string.strip(x)	returns a list of values delimited by x
string.join(L)	returns a string with L values joined by string
string.format(x)	returns a string that includes formatted x

List methods

list.append(x)	adds x to the end of the list
list.extend(L)	appends L to the end of the list
list.insert(i,x)	inserts x at i position
list.remove(x)	removes the first list item whose value is x
list.pop(i)	removes the item at position i and returns its value
list.clear()	removes all items from the list
list.index(x)	returns a list of values delimited by x
list.count(x)	returns a string with list values joined by S
list.sort()	sorts list items
list.reverse()	reverses list elements
list.copy()	returns a copy of the list

Dictionary methods

dict.keys()	returns a list of keys
dict.values()	returns a list of values
dict.items()	returns a list of pairs (key,value)
dict.get(k)	returns the value associtated to the key k
dict.pop()	removes the item associated to the key and returns its value
dict.update(D)	adds keys-values (D) to dictionary
dict.clear()	removes all keys-values from the dictionary
dict.copy()	returns a copy of the dictionary

Figure 2: Python basics

Libraries used

Pandas

Pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.

Features of Pandas

- **Handling of data:** The Pandas library provides a really fast and efficient way to manage and explore data. It does that by providing us with Series and Data Frames, which help us not only to represent data efficiently but also manipulate it in various ways.
- **Alignment and indexing:** Organization and labeling of data are perfectly taken care of by the intelligent methods of alignment and indexing, which can be found within Pandas.
- **Handling missing data:** One of the many problems associated with data is the occurrence of missing data or value as they might adulterate study results. Some Pandas features have you covered on this end because handling missing values is integrated within the library.
- **Merging and joining of datasets**: Pandas can help to merge various datasets, with extreme efficiency so that we don't face any problems while analyzing the data.
- **Visualize:** Visualizing is what makes the results of the study understandable by human eyes. Pandas have an in-built ability which helps in plotting data and see the various kinds of graphs formed.
- **Grouping:** With the help of the features of Pandas like GroupBy, we can split data into categories, according to the criteria set. The GroupBy function splits the data, implements a function and then combines the results.
- **Perform mathematical operations on the data**

Numpy

Numpy is a general-purpose array-processing package. It provides a high-performance multidimensional array object and tools for working with these arrays. It is the fundamental package for scientific computing with Python. Besides its obvious scientific uses, Numpy can also be used as an efficient multi-dimensional container of generic data.

In numpy, arrays allow a wide range of operations that can be performed on a particular array or a combination of Arrays. These operations include some basic Mathematical operation as well as Unary and Binary operations.

Every Numpy array is a table of elements (usually numbers), all of the same type, indexed by a tuple of positive integers. Every ndarray has an associated data type (dtype) object. This data type object (dtype) provides information about the layout of the array. The values of a ndarray are stored in a buffer that can be thought of as a contiguous block of memory bytes which can be interpreted by the dtype object. Numpy provides a large set of numeric data types that can be used to construct arrays. At the time of Array creation, Numpy tries to guess a data type, but functions that construct arrays usually also include an optional argument to explicitly specify the data type.

Matplotlib

Matplotlib is an amazing visualization library in Python for 2D plots of arrays. Matplotlib is a multi-platform data visualization library built on NumPy arrays and designed to work with the broader SciPy stack. One of the greatest benefits of visualization is that it allows us visual access to huge amounts of data in easily digestible visuals. Matplotlib consists of several plots like line, bar, scatter, histogram, etc.

Scikit learn

Scikit-learn is an open source Python library for machine learning. The library supports state-of-the-art algorithms such as KNN, XGBoost, random forest, SVM among others. It is built on top of Numpy. Scikit-learn is widely used in kaggle competition as well as prominent tech companies. Scikit-learn helps in preprocessing, dimensionality reduction (parameter selection), classification, regression, clustering, and model selection.

Seaborn

Seaborn is a library for making statistical graphics in Python. It is built on top of matplotlib and closely integrated with pandas data structures.

Some of the functionalities offered by this library are:

- A dataset-oriented API for examining relationships between multiple variables
- Specialized support for using categorical variables to show observations or aggregate statistics
- Options for visualizing univariate or bivariate distributions and for comparing them between subsets of data
- Automatic estimation and plotting of linear regression models for different kinds of dependent variables
- High-level abstractions for structuring multi-plot grids that let you easily build complex visualizations
- Concise control over matplotlib figure styling with several built-in themes
- Tools for choosing color palettes that faithfully reveal patterns in your data

It aims to make visualization a central part of exploring and understanding data. Its dataset-oriented plotting functions operate on data frames and arrays containing whole datasets and internally perform the necessary semantic mapping and statistical aggregation to produce informative plots.

Graphviz

Graphviz is a Python interface to the Graphviz graph layout and visualization package. With Graphviz you can create, edit, read, write, and draw graphs using Python to access the Graphviz graph data structure and layout algorithms. Graphviz is a package of open-source tools initiated by AT&T Labs Research for drawing graphs specified in DOT language scripts. It also provides libraries for software applications to use the tools.

Data Science Methodology

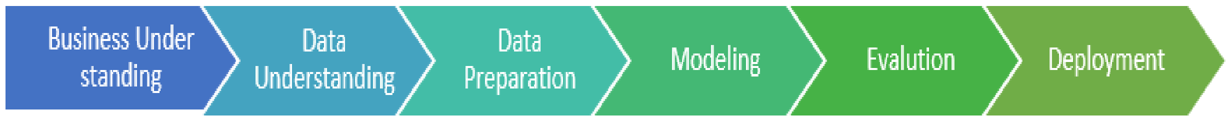


Figure 3: Phases of Data Science

a. Business Understanding

The very first step consists of business understanding. Whenever any requirement occurs, firstly we need to determine the business objective, assess the situation, determine data mining goals and then produce the project plan as per the requirement. Business objectives are defined in this phase.

b. Data Exploration

The second step consists of Data understanding. For further process, we need to gather initial data, describe and explore the data and verify data quality to ensure it contains the data we require. Data collected from the various sources is described in terms of its application and need for the project in this phase. This is also known as data exploration. This is necessary to verify the quality of data collected.

c. Data Preparation

Next, comes Data preparation. From the data collected in the last step, we need to select data as per the need, clean it, construct it to get useful information and then integrate it all. Finally, we need to format the data to get the appropriate data. Data is selected, cleaned, and integrated with the format finalized for the analysis in this phase.

d. Data Modeling

Once data is gathered, we need to do data modeling. For this, we need to select the modeling technique, generate test design, build a model and assess the model built. The data model is build to analyze relationships between various selected objects in the data, test cases are built for assessing the model and the model is tested and implemented on the data in this phase.

e. Data Evaluation

Next comes data evaluation where we evaluate the results generated in the last step, review the scope of error and determine the next steps that need to be performed. The results of the test cases are evaluated and reviewed for the scope of error in this phase.

f. Deployment

The final step in the analytic process is deployment. Here we need to plan the deployment and monitoring and maintenance, we need to produce the final reports and review the project. The results of the analysis are deployed in this phase. This is also known as reviewing the project.

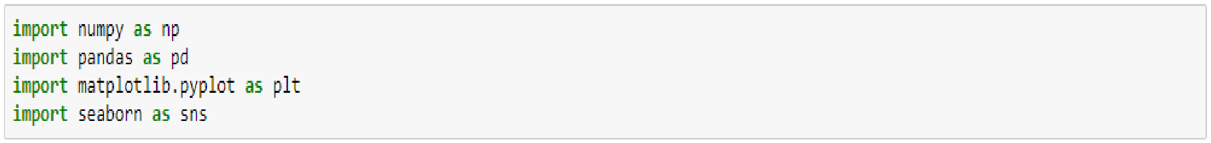
PROJECT IMPLEMENTATION

Source and Data collection

Data for project (network anomaly detection), the data dictionary, and the business problem statements were obtained from the organization. The data given by the organization had two datasets called **train** and **test** datasets. The train dataset consists of 125974 rows and 43 columns (features). The test dataset consists of 22545 rows and 43 columns (features).

Importing libraries

The necessary libraries are imported.



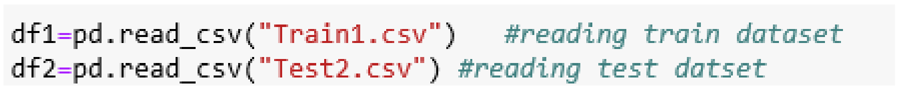
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Figure 4: Importing libraries

Reading the Data set

The dataset is read by using the read_csv function by pandas, the data is read into a data frame ‘df' in the notebook.

Pandas Data Frame is a two-dimensional size-mutable, potentially heterogeneous tabular data structure with labeled axes (rows and columns).



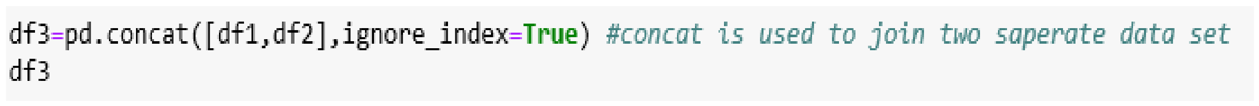
```
df1=pd.read_csv("Train1.csv")   #reading train dataset
df2=pd.read_csv("Test2.csv") #reading test datset
```

Figure 5: Reading the Train and Test dataset

In the figure mentioned above both the **Train** and **Test**, dataset are read into new data frames. df.head () prints only the first five rows along with all the columns.

Concatenation of Train and Test data

As the given data has two different datasets Train and Test, the concat function is used for merging the two different data frames.



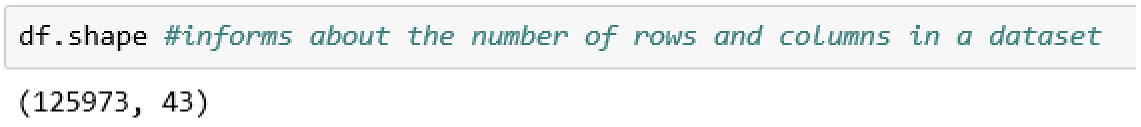
```
df3=pd.concat([df1,df2],ignore_index=True) #concat is used to join two saperate data set
df3
```

Figure 6: Concatenation of data

Understanding dataset

By using **the shape** function, the number of columns and rows of a dataset can be found out.

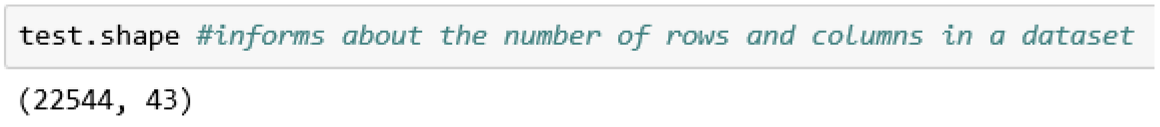
Originally the Train dataset consists of 125973 rows and 43 columns. The test consists of 22544 rows and 43 rows



```
df.shape #informs about the number of rows and columns in a dataset
```

```
(125973, 43)
```

Figure 7: Shape of the train dataset

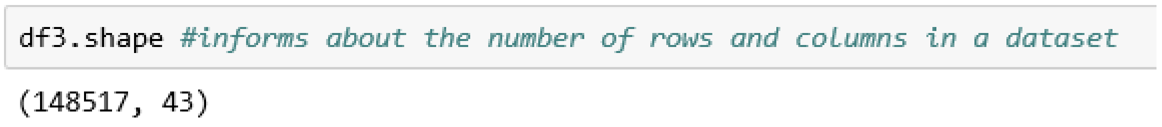


```
test.shape #informs about the number of rows and columns in a dataset
```

```
(22544, 43)
```

Figure 8: Shape of the test dataset

After concatenation, the dataset consists of 148517 and 43 columns.

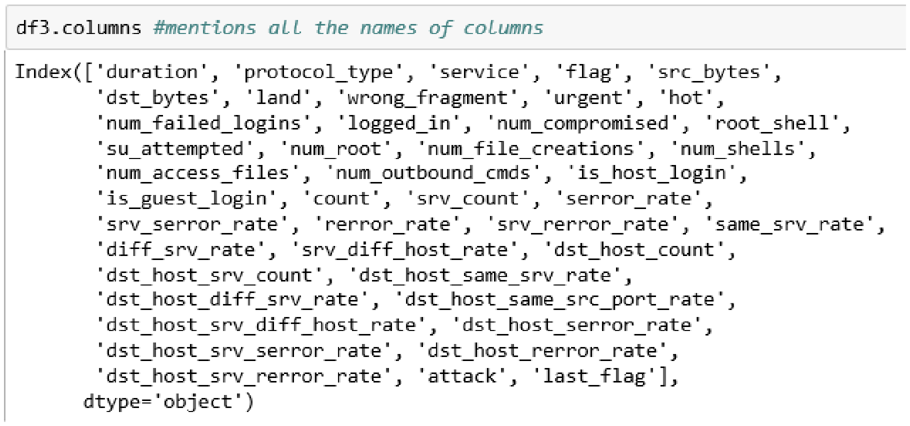


```
df3.shape #informs about the number of rows and columns in a dataset
```

```
(148517, 43)
```

Figure 9: Shape of concatenated data

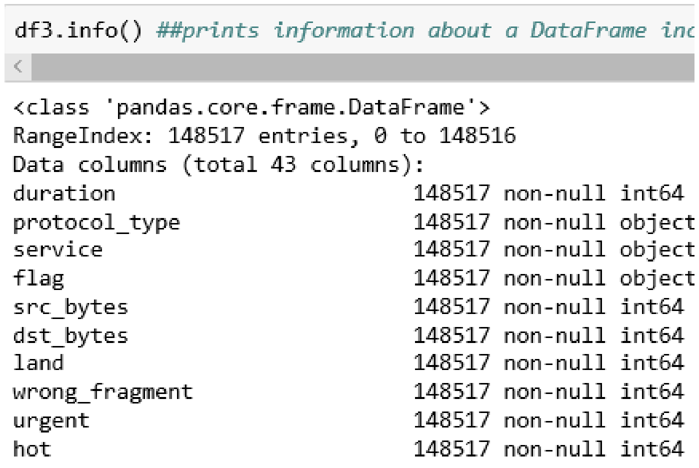
All the 43 features in the data are mentioned by using **columns function**.



```
df3.columns #mentions all the names of columns

Index(['duration', 'protocol_type', 'service', 'flag', 'src_bytes',
       'dst_bytes', 'land', 'wrong_fragment', 'urgent', 'hot',
       'num_failed_logins', 'logged_in', 'num_compromised', 'root_shell',
       'su_attempted', 'num_root', 'num_file_creations', 'num_shells',
       'num_access_files', 'num_outbound_cmds', 'is_host_login',
       'is_guest_login', 'count', 'srv_count', 'serror_rate',
       'srv_serror_rate', 'rerror_rate', 'srv_rerror_rate', 'same_srv_rate',
       'diff_srv_rate', 'srv_diff_host_rate', 'dst_host_count',
       'dst_host_srv_count', 'dst_host_same_srv_rate',
       'dst_host_diff_srv_rate', 'dst_host_same_src_port_rate',
       'dst_host_srv_diff_host_rate', 'dst_host_serror_rate',
       'dst_host_srv_serror_rate', 'dst_host_rerror_rate',
       'dst_host_srv_rerror_rate', 'attack', 'last_flag'],
      dtype='object')
```

Figure 10: Columns

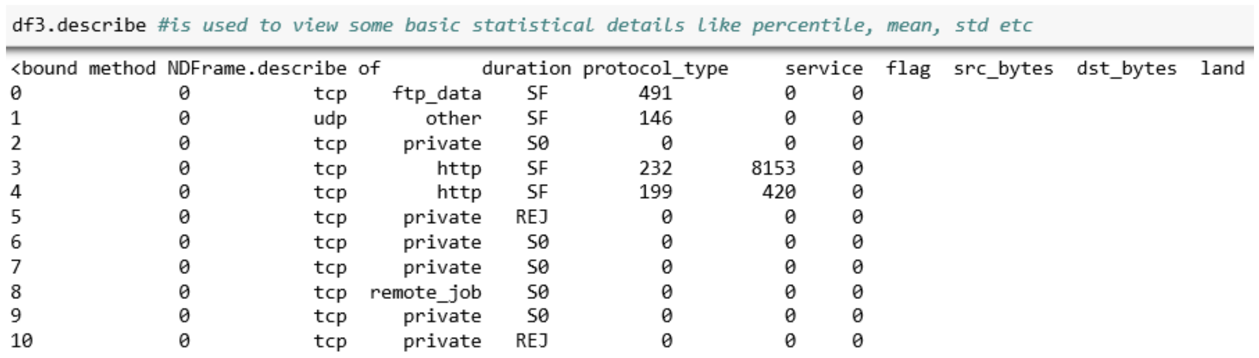
info() method gives us information about every feature in the dataset like type and number of missing values. There are 15 float columns, 4 object columns, and 24 integer columns in the data.



```
df3.info() ##prints information about a DataFrame inc

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 148517 entries, 0 to 148516
Data columns (total 43 columns):
duration                       148517 non-null int64
protocol_type                  148517 non-null object
service                        148517 non-null object
flag                           148517 non-null object
src_bytes                      148517 non-null int64
dst_bytes                      148517 non-null int64
land                           148517 non-null int64
wrong_fragment                 148517 non-null int64
urgent                         148517 non-null int64
hot                            148517 non-null int64
```

Figure 11: Info of the dataset

describe() method gives us mean, standard deviation, min, max, count and other features of every feature in the data set. It is shown below:

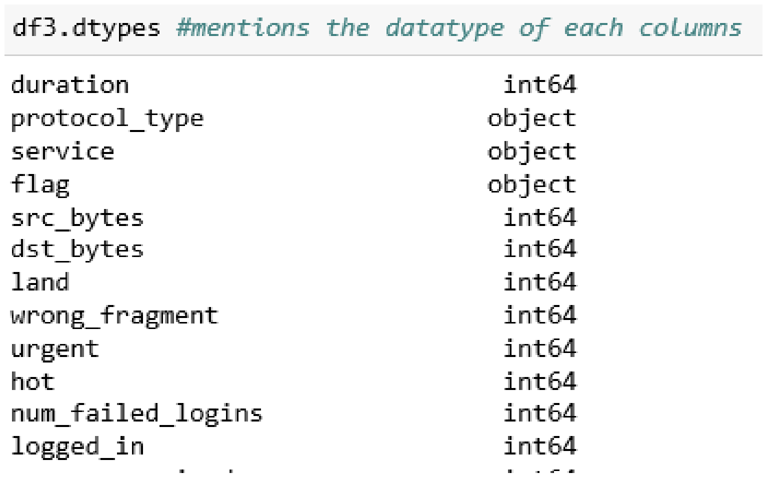


```
df3.describe #is used to view some basic statistical details like percentile, mean, std etc
```

```
<bound method NDFrame.describe of         duration protocol_type     service  flag  src_bytes  dst_bytes  land
0              0           tcp    ftp_data    SF        491          0     0
1              0           udp       other    SF        146          0     0
2              0           tcp     private    S0          0          0     0
3              0           tcp        http    SF        232       8153     0
4              0           tcp        http    SF        199        420     0
5              0           tcp     private   REJ          0          0     0
6              0           tcp     private    S0          0          0     0
7              0           tcp     private    S0          0          0     0
8              0           tcp  remote_job    S0          0          0     0
9              0           tcp     private    S0          0          0     0
10             0           tcp     private   REJ          0          0     0
```

Figure 12: Describe

dtypes mentions the data types of all the features present in the data.



```
df3.dtypes #mentions the datatype of each columns
```

```
duration                          int64
protocol_type                    object
service                          object
flag                             object
src_bytes                         int64
dst_bytes                         int64
land                              int64
wrong_fragment                    int64
urgent                            int64
hot                               int64
num_failed_logins                 int64
logged_in                         int64
```

Figure 13: Datatypes

IsNull finds out the present null values in the data. Since the data provided has no null values, handling of missing data need not be done.

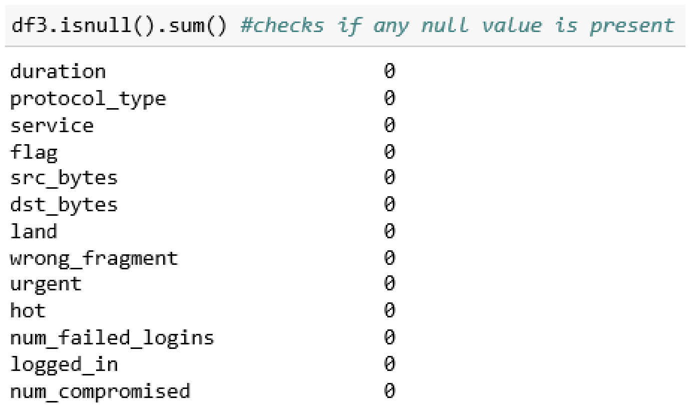

```
df3.isnull().sum() #checks if any null value is present

duration                 0
protocol_type            0
service                  0
flag                     0
src_bytes                0
dst_bytes                0
land                     0
wrong_fragment           0
urgent                   0
hot                      0
num_failed_logins        0
logged_in                0
num_compromised          0
```

Figure 14: Null value detection

Preparing the data

Numerical and categorical features

Categorical data represents characteristics. Therefore it can represent things like a person's gender, language, etc. Categorical data can also take on numerical values (Example: 1 for female and 0 for male).

Numerical data is the numerical measurement expressed not through a natural language description, but rather in terms of numbers. These data have the meaning as a measurement, such as a person's height, weight, or IQ; or they're a count, such as the number of stock shares a person owns, etc.

According to the dataset there are 34 **numerical** features. The names of the features are mentioned below:

"duration", "src_bytes", "dst_bytes", "land", "wrong_fragment", "urgent", "hot", "num_failed_logins", "num_compromised", "num_root", "num_file_creations", "num_shells", "num_access_files", "num_outbound_cmds", "count", "srv_count", "serror_rate", "srv_serror_rate", "rerror_rate", "srv_rerror_rate", "same_srv_rate", "diff_srv_rate", "srv_diff_host_rate", "dst_host_count", "dst_host_srv_count", "dst_host_same_srv_rate", "dst_host_diff_srv_rate", "dst_host_same_src_port_rate", "dst_host_srv_diff_host_rate", "dst_host_serror_rate", "dst_host_srv_serror_rate", "last_flag", "dst_host_rerror_rate", "dst_host_srv_rerror_rate"

There are 9 **categorical** features. The names of the features are mentioned below:

"protocol_type", "service", "flag", "root_shell", "su_attempted", "logged_in", "is_host_login", "is_guest_login", "attack"

Independent and Dependent variables

Independent variables *(also referred to as Features)* are the input for a process that is being analyzed. Dependent variables are the output of the process.

According to the problem statement provided by the organization "**attack**" feature is the dependent variable of the dataset. And other all the features are considered as independent variables.

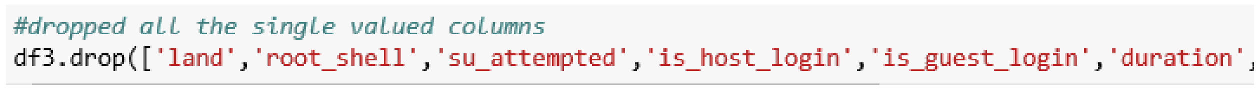
Single valued features

A Feature having a value that covers more than 90% of the column is called a single valued feature. Example: 'duration' column has 0 covering 92% of the column and 1 covering 8%

After understanding the data given, there were few features that were single valued. These features are mentioned below:

'land', 'root_shell', 'su_attempted', 'is_host_login', 'is_guest_login', 'duration', 'wrong_fragment', 'hot', 'urgent', 'num_failed_logins', 'num_compromised', 'num_root', 'num_file_creations', 'num_shells', 'num_access_files', 'num_outbound_cmds'

These singled valued features are removed from the data so that these features don't affect the model during model fitting.

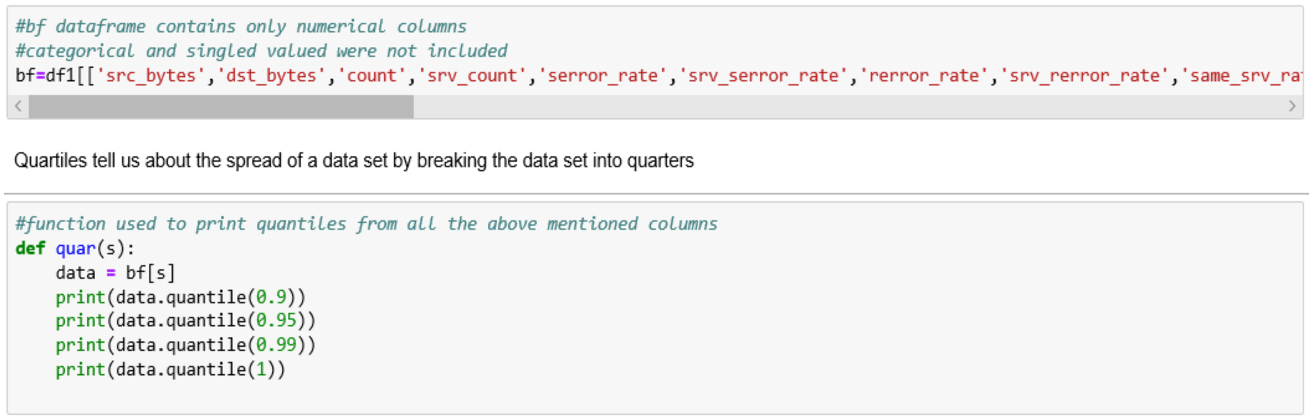


```
#dropped all the single valued columns
df3.drop(['land','root_shell','su_attempted','is_host_login','is_guest_login','duration',
```

Figure 15: Dropping single-valued features

Detection of outliers and clipping of data

Outliers are extreme values that deviate from other observations on data; they may indicate variability in measurement, experimental errors or a novelty. The detection of outliers is done by using **quantiles** on each column.



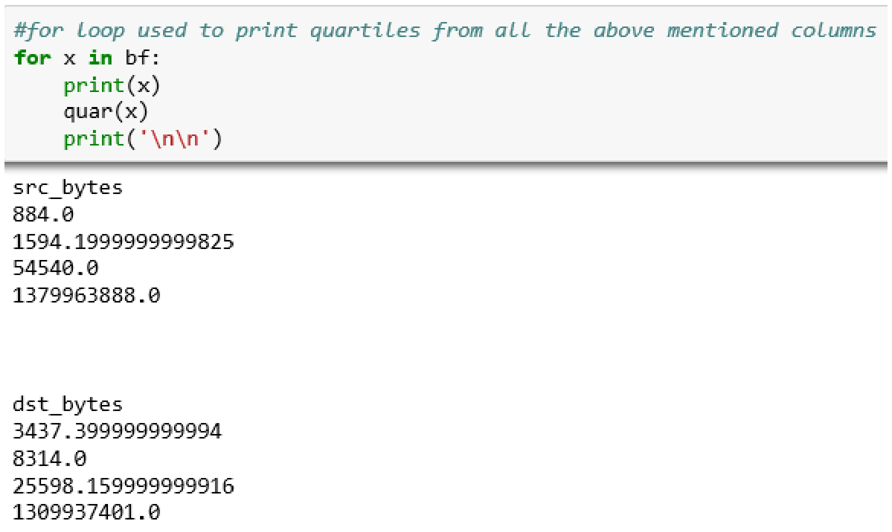
```
#bf dataframe contains only numerical columns
#categorical and singled valued were not included
bf=df1[['src_bytes','dst_bytes','count','srv_count','serror_rate','srv_serror_rate','rerror_rate','srv_rerror_rate','same_srv_ra
```

Quartiles tell us about the spread of a data set by breaking the data set into quarters

```
#function used to print quantiles from all the above mentioned columns
def quar(s):
    data = bf[s]
    print(data.quantile(0.9))
    print(data.quantile(0.95))
    print(data.quantile(0.99))
    print(data.quantile(1))
```

Figure 16: Outlier

In the above figure, quantiles of every column at 0.90, 0.95, 0.99 and 1 are found out.



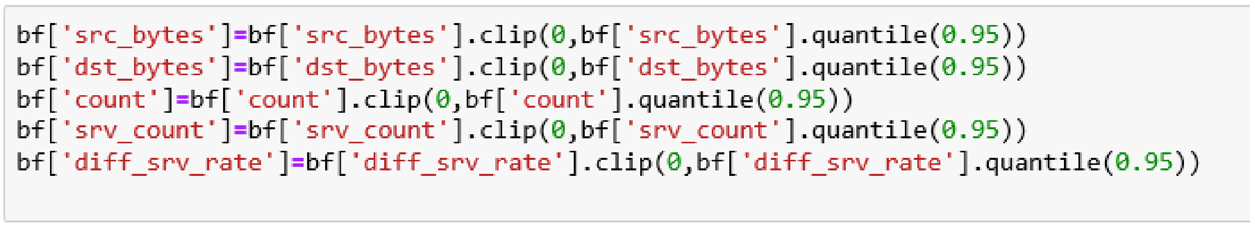
```
#for loop used to print quartiles from all the above mentioned columns
for x in bf:
    print(x)
    quar(x)
    print('\n\n')
```

```
src_bytes
884.0
1594.1999999999825
54540.0
1379963888.0



dst_bytes
3437.399999999994
8314.0
25598.159999999916
1309937401.0
```

Figure 17: Quantiles

According to the above figure, there is a sudden increase in value at a particular quantile. At these particular quantiles, clipping is needed to be done on the entire column.

The columns that are clipped are ‘src_bytes’, ‘dst_bytes’, ‘count’, ‘srv_count’, ‘diff_srv_rate’

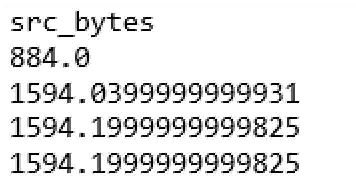


```
bf['src_bytes']=bf['src_bytes'].clip(0,bf['src_bytes'].quantile(0.95))
bf['dst_bytes']=bf['dst_bytes'].clip(0,bf['dst_bytes'].quantile(0.95))
bf['count']=bf['count'].clip(0,bf['count'].quantile(0.95))
bf['srv_count']=bf['srv_count'].clip(0,bf['srv_count'].quantile(0.95))
bf['diff_srv_rate']=bf['diff_srv_rate'].clip(0,bf['diff_srv_rate'].quantile(0.95))
```

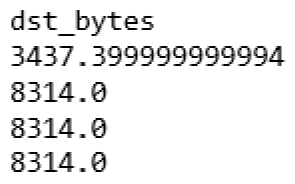
Figure 18: Clipping of data

For example: In figure 14, src_bytes has a sudden increase in the value at the quantile 0.95 so the src_byte is clipped at quantile 0.95 and so all the values after 0.95 also change. This change can be observed in figure 16.

Clipping is used to avoid using outliers in the calculations made later on.



```
src_bytes
884.0
1594.0399999999931
1594.1999999999825
1594.1999999999825
```

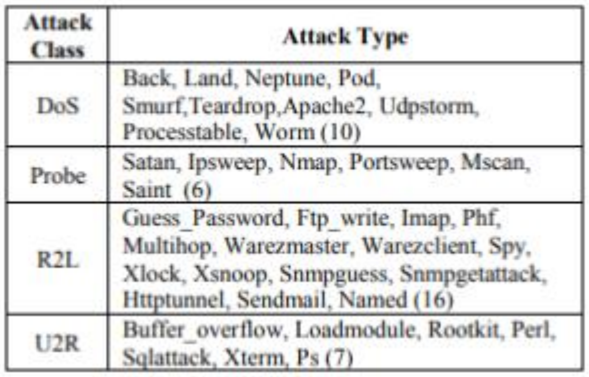


```
dst_bytes
3437.399999999994
8314.0
8314.0
8314.0
```

Figure 19: After Clipping of data

Changing all the different attack type to attack

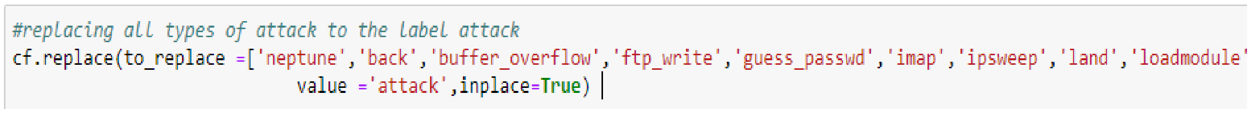
According to the problem statement, the project given needs to be done by using binomial classification i.e. the activity of the network can only be declared either as an **attack** or **normal** not by any other attack types.



Attack Class	Attack Type
DoS	Back, Land, Neptune, Pod, Smurf,Teardrop,Apache2, Udpstorm, Processtable, Worm (10)
Probe	Satan, Ipsweep, Nmap, Portsweep, Mscan, Saint (6)
R2L	Guess_Password, Ftp_write, Imap, Phf, Multihop, Warezmaster, Warezclient, Spy, Xlock, Xsnoop, Snmpguess, Snmpgetattack, Httptunnel, Sendmail, Named (16)
U2R	Buffer_overflow, Loadmodule, Rootkit, Perl, Sqlattack, Xterm, Ps (7)

Figure 20: Attack types

In order to apply binomial classification, all the different attack types such as neptune, back, buffer_overflow, ftp_write, guess_passwd, imap, ipsweep, land, loadmodule, multihop, nmap, perl, phf, pod, portsweep, rootkit, satan, smurf, spy, teardrop, warezclient, and warezmaster are replaced by the word **attack**.



```
#replacing all types of attack to the label attack
cf.replace(to_replace =['neptune','back','buffer_overflow','ftp_write','guess_passwd','imap','ipsweep','land','loadmodule'
                             value ='attack',inplace=True) |
```

Figure 21: Replacing attack types

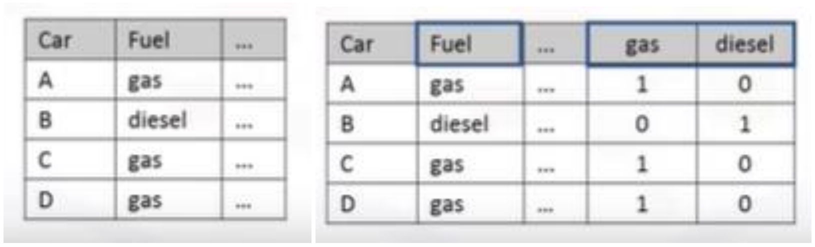
Now the attack column consists of only two types of variables:

- Attack
- Normal

Dummification applied on categorical features

A Dummy variable or Indicator Variable is an artificial variable created to represent an attribute with two or more distinct categories/levels. A Dummy variable or Indicator Variable is an artificial variable created to represent an attribute with two or more distinct categories/levels.

For example:



Car	Fuel	...
A	gas	...
B	diesel	...
C	gas	...
D	gas	...

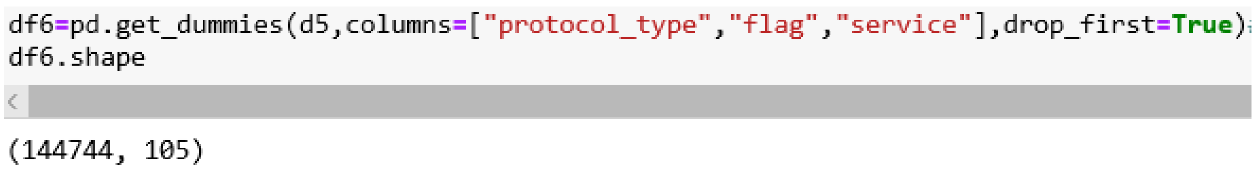
Car	Fuel	...	gas	diesel
A	gas	...	1	0
B	diesel	...	0	1
C	gas	...	1	0
D	gas	...	1	0

Figure 22: Dummification example

The remaining categorical features after removing all the single-valued features and except dependent variable (attack) are:

- protocol_type
- service
- flag

The pd.get_dummies function is used to get the dummies for the respective column.



```
df6=pd.get_dummies(d5,columns=["protocol_type","flag","service"],drop_first=True)
df6.shape
```

```
(144744, 105)
```

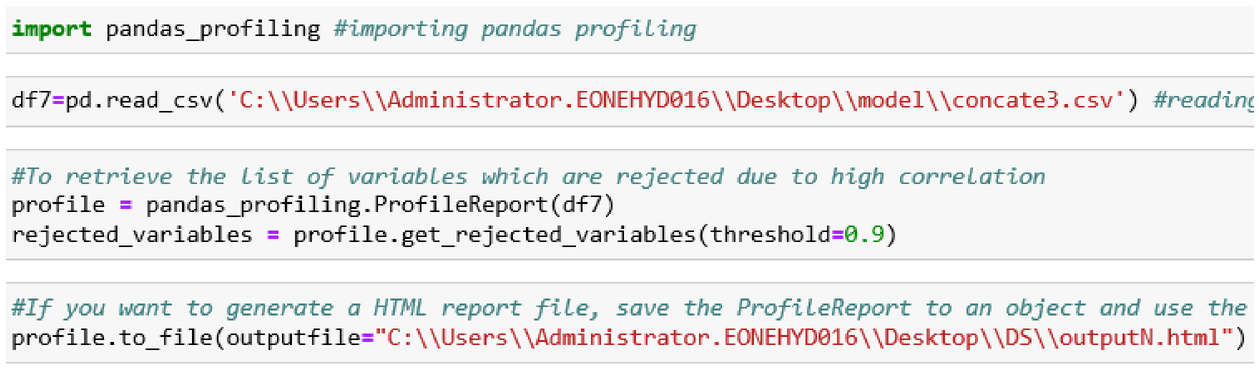
Figure 23: Dummification

After dummification, the number of columns has increased from 27 to 105.

Feature selection

Feature Selection is the process where you automatically or manually select those features which contribute most to your prediction variable or output in which you are interested. Having irrelevant features in your data can decrease the accuracy of the models and make your model learn based on irrelevant features. Pandas profiling is used in feature selection.

Pandas profiling is used for simple and fast exploratory data analysis of a Pandas Data frame. The first part of the HTML EDA report will contain an overview section providing you with basic information (number of observations, number of variables, etc.). It will also output a list of warnings telling you were to double-check the data and potentially focus your cleaning efforts on.



```
import pandas_profiling #importing pandas profiling
```

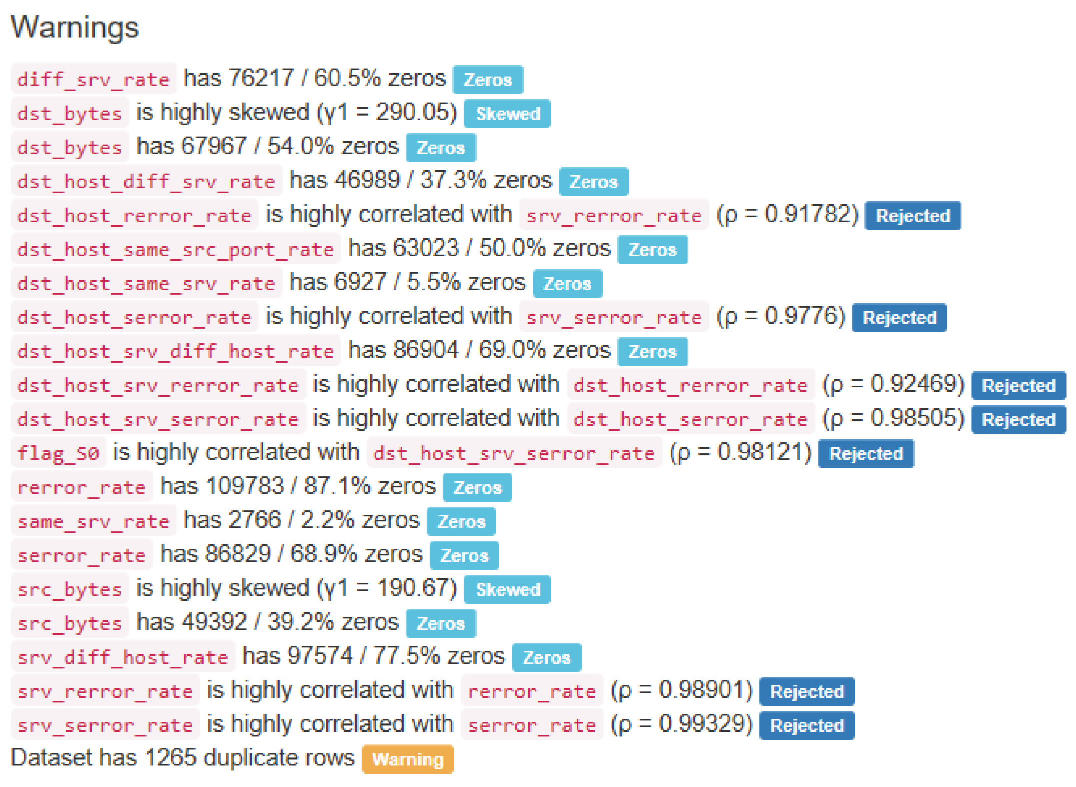
```
df7=pd.read_csv('C:\\Users\\Administrator.EONEHYD016\\Desktop\\model\\concate3.csv') #reading
```

```
#To retrieve the list of variables which are rejected due to high correlation
profile = pandas_profiling.ProfileReport(df7)
rejected_variables = profile.get_rejected_variables(threshold=0.9)
```

```
#If you want to generate a HTML report file, save the ProfileReport to an object and use the
profile.to_file(outputfile="C:\\Users\\Administrator.EONEHYD016\\Desktop\\DS\\outputN.html")
```

Figure 24: Implementation of Pandas Profiling

Using pandas profiling, the columns with high inter-correlation were recognized and dropped.



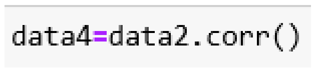
Warnings

```
diff_srv_rate has 76217 / 60.5% zeros Zeros
dst_bytes is highly skewed (γ1 = 290.05) Skewed
dst_bytes has 67967 / 54.0% zeros Zeros
dst_host_diff_srv_rate has 46989 / 37.3% zeros Zeros
dst_host_rerror_rate is highly correlated with srv_rerror_rate (ρ = 0.91782) Rejected
dst_host_same_src_port_rate has 63023 / 50.0% zeros Zeros
dst_host_same_srv_rate has 6927 / 5.5% zeros Zeros
dst_host_serror_rate is highly correlated with srv_serror_rate (ρ = 0.9776) Rejected
dst_host_srv_diff_host_rate has 86904 / 69.0% zeros Zeros
dst_host_srv_rerror_rate is highly correlated with dst_host_rerror_rate (ρ = 0.92469) Rejected
dst_host_srv_serror_rate is highly correlated with dst_host_serror_rate (ρ = 0.98505) Rejected
flag_S0 is highly correlated with dst_host_srv_serror_rate (ρ = 0.98121) Rejected
rerror_rate has 109783 / 87.1% zeros Zeros
same_srv_rate has 2766 / 2.2% zeros Zeros
serror_rate has 86829 / 68.9% zeros Zeros
src_bytes is highly skewed (γ1 = 190.67) Skewed
src_bytes has 49392 / 39.2% zeros Zeros
srv_diff_host_rate has 97574 / 77.5% zeros Zeros
srv_rerror_rate is highly correlated with rerror_rate (ρ = 0.98901) Rejected
srv_serror_rate is highly correlated with serror_rate (ρ = 0.99329) Rejected
Dataset has 1265 duplicate rows Warning
```

Figure 25: Profiling

According to the above mentioned figure, there are 14 inter-correlated features and all the features on the left hand side: dst_host_rerror_rate, dst_host_serror_rate, dst_host_srv_rerror_rate, dst_host_srv_serror_rate, flag_s0, srv_rerror_rate, srv_serror_rate are dropped out the data to remove irrelevant features in data as it can decrease the accuracy of the models.

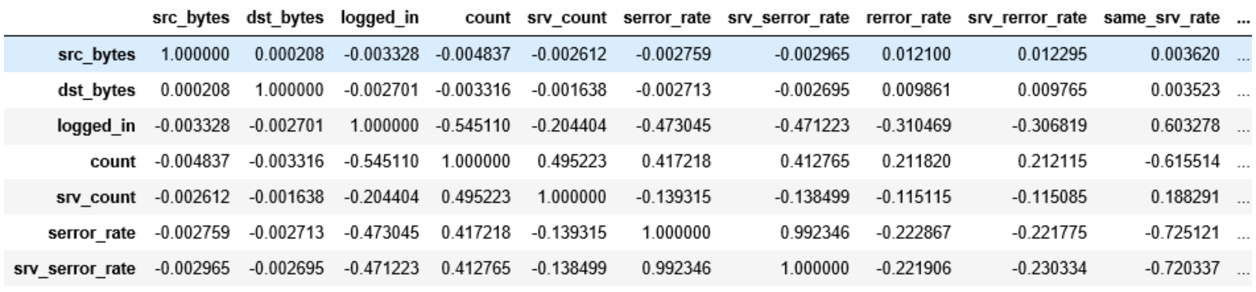
After removing the irreverent features, the remain features are checked for correlations by using **corr()** function.



```
data4=data2.corr()
```

Figure 26: Correlation

The correlation table is saved to excel and the excel file is analyzed to select the columns correlated with ‘attack'. The correlation range considered is between 0.1 to 0.7 and -0.7 to -0.1. These are the columns that will be used for further analysis.



	src_bytes	dst_bytes	logged_in	count	srv_count	serror_rate	srv_serror_rate	rerror_rate	srv_rerror_rate	same_srv_rate	...
src_bytes	1.000000	0.000208	-0.003328	-0.004837	-0.002612	-0.002759	-0.002965	0.012100	0.012295	0.003620	...
dst_bytes	0.000208	1.000000	-0.002701	-0.003316	-0.001638	-0.002713	-0.002695	0.009861	0.009765	0.003523	...
logged_in	-0.003328	-0.002701	1.000000	-0.545110	-0.204404	-0.473045	-0.471223	-0.310469	-0.306819	0.603278	...
count	-0.004837	-0.003316	-0.545110	1.000000	0.495223	0.417218	0.412765	0.211820	0.212115	-0.615514	...
srv_count	-0.002612	-0.001638	-0.204404	0.495223	1.000000	-0.139315	-0.138499	-0.115115	-0.115085	0.188291	...
serror_rate	-0.002759	-0.002713	-0.473045	0.417218	-0.139315	1.000000	0.992346	-0.222867	-0.221775	-0.725121	...
srv_serror_rate	-0.002965	-0.002695	-0.471223	0.412765	-0.138499	0.992346	1.000000	-0.221906	-0.230334	-0.720337	...

Figure 27: After corr() function is used

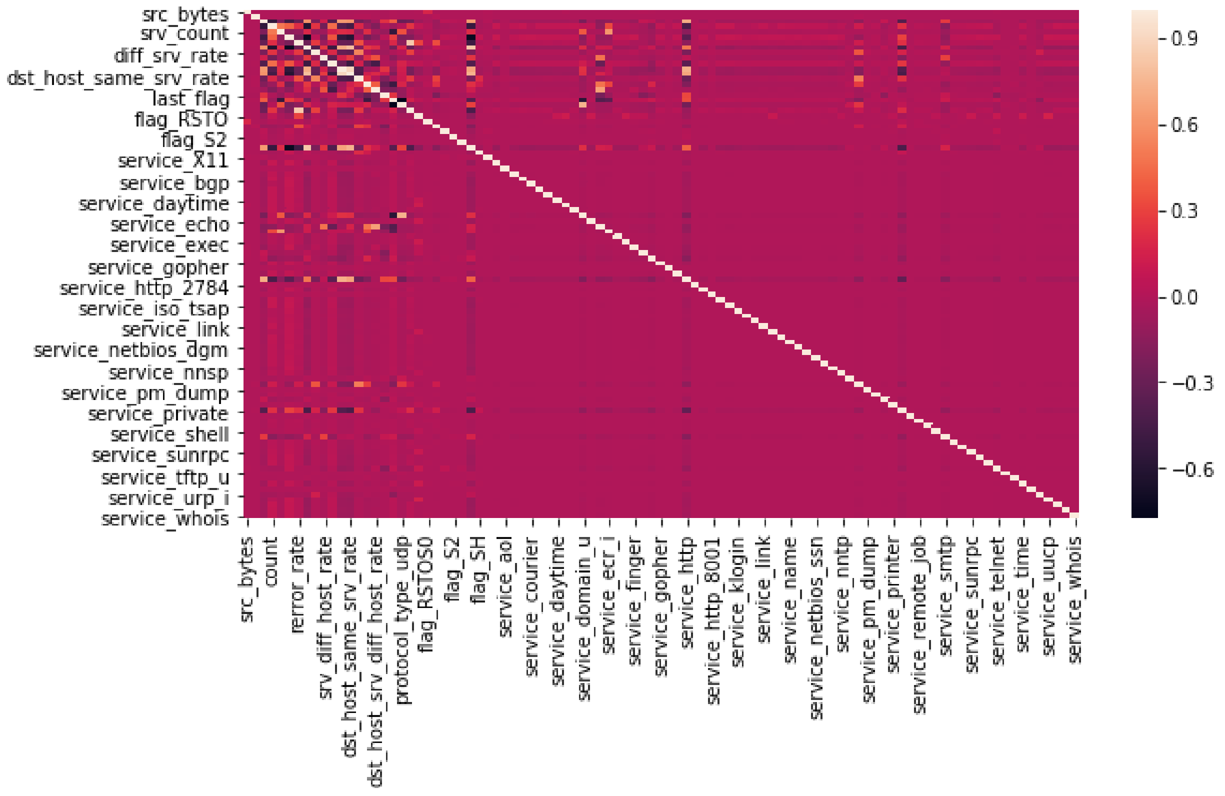


Figure 28: Heatmap correlation

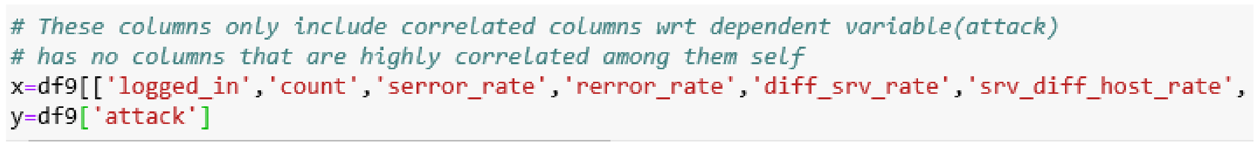
Correlation between features can also be found out by using heatmap. Heatmap is used to visualize the correlation between features with respect to the dependent variable (attack).

The features considered from the correlation matrix are:

Logged_in, count, serror_rate, rerror_rate, diff_srv_rate, srv_diff_host_rate, dst_host_count, dst_host_same_srv_rate, dst_host_diff_srv_rate, last_flag, protocol_type_udp, service_domain_u, service_eco_i, service_ecr_i, service_http, service_private, service_smtp, flag_REJ, flag_RSTR,src_bytes, dst_bytes, dst_host_srv_count, flag_RST.

Features selected

The features mentioned in x and y terms are obtained from the feature selection process. Where x represents independent variables and y represents the dependent variable.

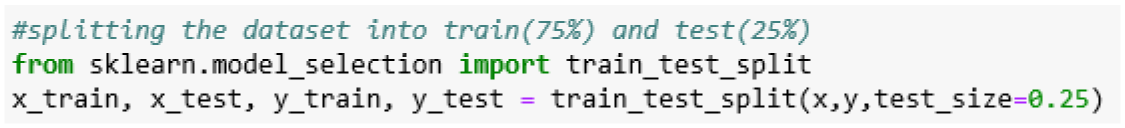


```
# These columns only include correlated columns wrt dependent variable(attack)
# has no columns that are highly correlated among them self
x=df9[['logged_in','count','serror_rate','rerror_rate','diff_srv_rate','srv_diff_host_rate',
y=df9['attack']
```

Figure 29: features selected for logistic regression

Splitting of data

Since the train and test dataset are concatenated as one data, so the data is split into train and test by a percentage of 75% (train) and 25% (test).



```
#splitting the dataset into train(75%) and test(25%)
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.25)
```

Figure 30: splitting of data (LR)

The data we use is usually split into training data and test data. The training set contains a known output and the model learns on this data in order to be generalized to other data later on. the test dataset (or subset) is provided in order to test a model's prediction.

Model Building

Building a model or set of models to solve the problem, test how well they perform and iterate until you have a model that gives satisfactory results. Stabilize and scale your model as well as your data collection and processing to produce useful outputs in your production environment.

The main aim of the project is to predict if the activity by the network is an **attack** or **normal**. The given data is based on the **classification** problem as the output variable given in the data is a **categorical** variable, such as “attack” and “normal”. The algorithms used:

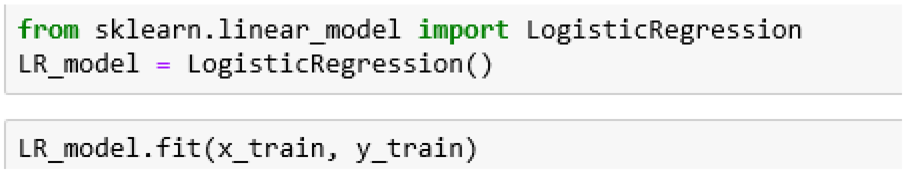
- Logistic Regression
- Random Forest
- Decision Tree

Logistic Regression

Logistic Regression is a Machine Learning algorithm which is used for the classification problems; it is a predictive analysis algorithm and based on the concept of probability. We can call a Logistic Regression a Linear Regression model but the Logistic Regression uses a more complex cost function; this cost function can be defined as the ‘Sigmoid function’ or also known as the ‘logistic function’.

Fitting the model

Sklearn.linear_model is used to implement the logistic regression algorithm.



```
from sklearn.linear_model import LogisticRegression
LR_model = LogisticRegression()
```

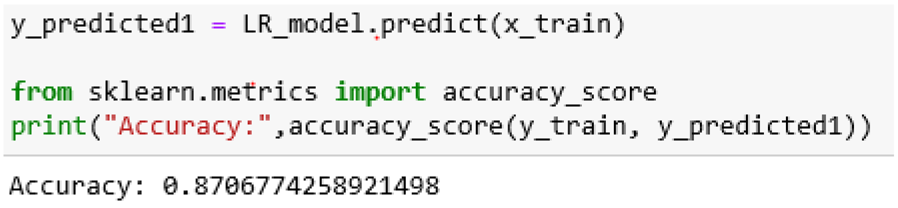
```
LR_model.fit(x_train, y_train)
```

Figure 31: Logistic regression Model fitting

Since the data is processed, the model is trained using train data which contains a known output and the model learns on this data.

Prediction made on train data

The model built is used to predict the target variable by using train data and the accuracy score obtained by the model is 87%.



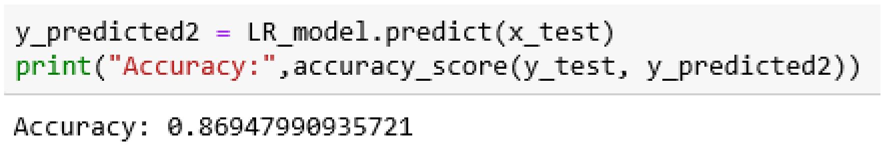
```
y_predicted1 = LR_model.predict(x_train)

from sklearn.metrics import accuracy_score
print("Accuracy:",accuracy_score(y_train, y_predicted1))
```

```
Accuracy: 0.8706774258921498
```

Figure 32: Prediction on train data (logistic regression)

Prediction made on test data

The model built is used to predict the target variable by using train data and the accuracy score obtained by the model is 86%.



```
y_predicted2 = LR_model.predict(x_test)
print("Accuracy:",accuracy_score(y_test, y_predicted2))
```

```
Accuracy: 0.86947990935721
```

Figure 33: Prediction on test data (logistic regression)

Confusion matrix

A confusion matrix is a summary of prediction results on a classification problem on a set of data for which the true values are known. The confusion matrix shows the ways in which your classification model is confused when it makes predictions. It gives us insight not only into the errors being made by a classifier but more importantly the types of errors that are being made.

	normal	attack
normal	42982	7880
attack	6159	51537

Figure 34: Confusion matrix on Train data (LR)

	normal	attack
normal	14094	2734
attack	1989	17369

Figure 35: Confusion matrix on test data (LR)

Mentioned below are the calculations made using the confusion matrix:

Sensitivity: it gives us an idea about when actually prediction is true, how often it predicts true.

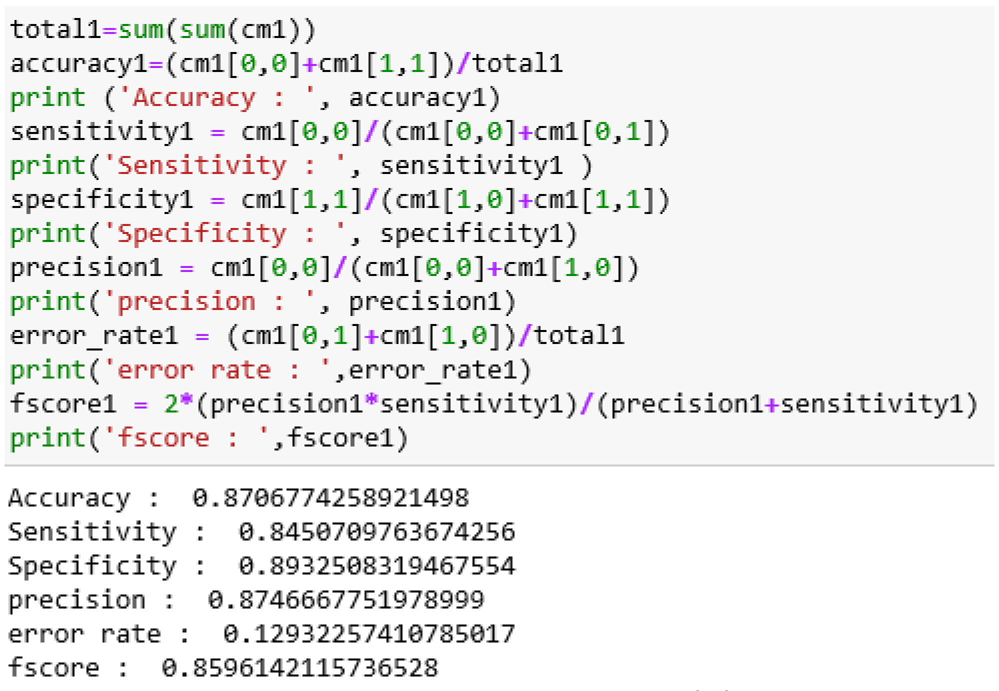
Specificity: it gives us an idea about when the prediction is false, how often it predicts true.

Precision: Precision tells us about when prediction is true, how often is it true.

F-score: It is the Harmonic mean of the two values which we have i.e. Precision and Recall. It considers both the Precision and Recall of the procedure to compute the score. Higher the F-score, the better will be the predictive power of the classification procedure.

Error rate: overall, how often is the classifier wrong with the prediction?

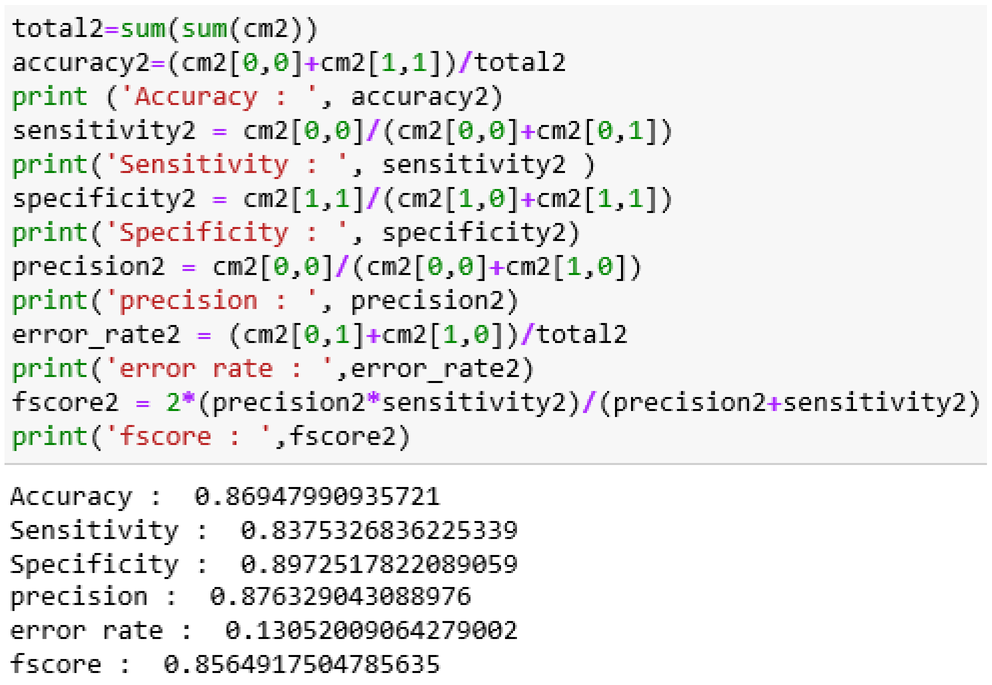
Accuracy: overall, how often is the classifier correct with the prediction?



```
total1=sum(sum(cm1))
accuracy1=(cm1[0,0]+cm1[1,1])/total1
print ('Accuracy : ', accuracy1)
sensitivity1 = cm1[0,0]/(cm1[0,0]+cm1[0,1])
print('Sensitivity : ', sensitivity1 )
specificity1 = cm1[1,1]/(cm1[1,0]+cm1[1,1])
print('Specificity : ', specificity1)
precision1 = cm1[0,0]/(cm1[0,0]+cm1[1,0])
print('precision : ', precision1)
error_rate1 = (cm1[0,1]+cm1[1,0])/total1
print('error rate : ',error_rate1)
fscore1 = 2*(precision1*sensitivity1)/(precision1+sensitivity1)
print('fscore : ',fscore1)
```

```
Accuracy :  0.8706774258921498
Sensitivity :  0.8450709763674256
Specificity :  0.8932508319467554
precision :  0.8746667751978999
error rate :  0.12932257410785017
fscore :  0.8596142115736528
```

Figure 36: Calculations made on train data (LR)



```python
total2=sum(sum(cm2))
accuracy2=(cm2[0,0]+cm2[1,1])/total2
print ('Accuracy : ', accuracy2)
sensitivity2 = cm2[0,0]/(cm2[0,0]+cm2[0,1])
print('Sensitivity : ', sensitivity2 )
specificity2 = cm2[1,1]/(cm2[1,0]+cm2[1,1])
print('Specificity : ', specificity2)
precision2 = cm2[0,0]/(cm2[0,0]+cm2[1,0])
print('precision : ', precision2)
error_rate2 = (cm2[0,1]+cm2[1,0])/total2
print('error rate : ',error_rate2)
fscore2 = 2*(precision2*sensitivity2)/(precision2+sensitivity2)
print('fscore : ',fscore2)
```

```
Accuracy :  0.86947990935721
Sensitivity :  0.8375326836225339
Specificity :  0.8972517822089059
precision :  0.876329043088976
error rate :  0.13052009064279002
fscore :  0.8564917504785635
```

Figure 37: Calculations made on test data (LR)

Auc-Roc curve

AUC - ROC curve is a performance measurement for the classification problem at various thresholds settings. ROC is a probability curve and AUC represents the degree or measure of separability. It tells how much model is capable of distinguishing between classes. Higher the AUC, better the model is at predicting normal as normal and attack as attack.

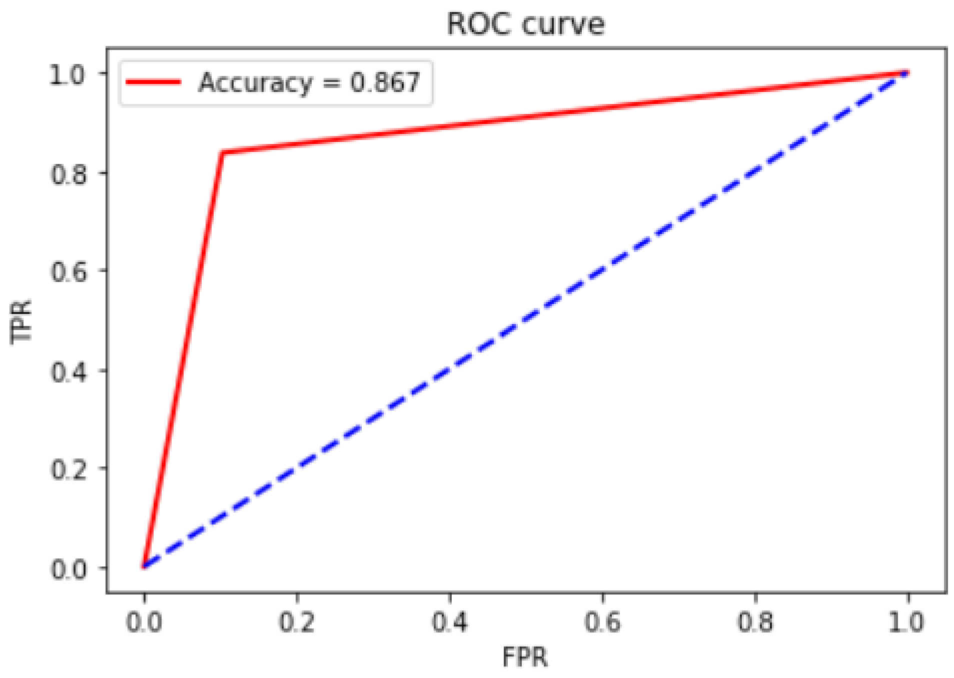


Figure 38: Auc-Roc curve

The ROC curve is plotted with TPR against the FPR where TPR is on y-axis and FPR is on the x-axis.

Next, Decision tree algorithm is implemented to achieve a better an accuracy of the model.

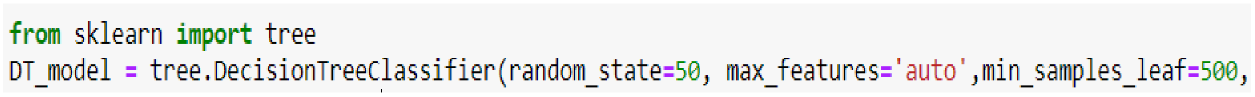
Decision Tree

A decision tree is a decision support tool that uses a tree-like graph or model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility. It is one way to display an algorithm that only contains conditional control statements.

A decision tree is a flowchart-like structure in which each internal node represents a "test" on an attribute (e.g. whether a coin flip comes up heads or tails), each branch represents the outcome of the test, and each leaf node represents a class label (decision taken after computing all attributes). The paths from the root to the leaf represent classification rules.

Parameters used

from the sklearn import, the tree is used to implement the **Decision Tree** algorithm.



```
from sklearn import tree
DT_model = tree.DecisionTreeClassifier(random_state=50, max_features='auto',min_samples_leaf=500,
```

Figure 39: Decision tree model

In the above figure, a few parameters are specially used by mentioning particular values or features to each parameter. The parameters used are:

Random state, max_features, min_sample_leaf, max_leaf_nodes, max_depth, min_samples_split.

Minimum samples for a node split

It defines the minimum number of samples (or observations) which are required in a node to be considered for splitting. It is used to control over-fitting. Higher values prevent a model from learning relations which might be highly specific to the particular sample selected for a tree.

Minimum samples for a terminal node (leaf)

Defines the minimum samples (or observations) required in a terminal node or leaf. It is used to control over-fitting similar to min_samples_split.

Maximum depth of the tree

It is the maximum depth of a tree. It is used to control over-fitting as higher depth will allow the model to learn relations very specific to a particular sample.

Maximum number of terminal nodes

It is the maximum number of terminal nodes or leaves in a tree. It can be defined in place of max_depth. Since binary trees are created, a depth of 'n' would produce a maximum of 2^n leaves.

Maximum features to consider for a split

It is the number of features to consider while searching for the best split. These will be randomly selected. If "auto" is chosen, then `max_features=sqrt(n_features).`

Random state

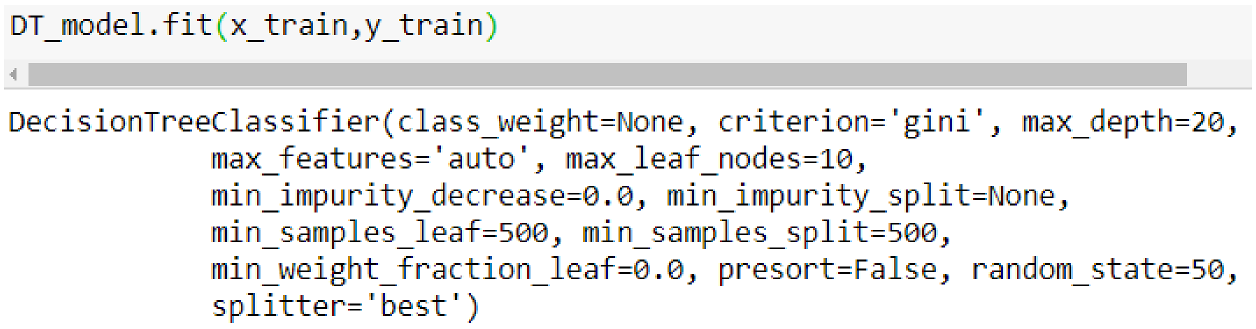
If int, random_state is the seed used by the random number generator; If RandomState instance, random_state is the random number generator.

The unmentioned parameters are: criterion, splitter, min_weight_fraction_leaf, min_impurity_decrease , min_impurity_split , presort , class_weight

The unmentioned parameters are considered in the model by their default value.

Model fitting

Since the data is processed, the model is trained using train data which contains a known output and the model learns on this data.



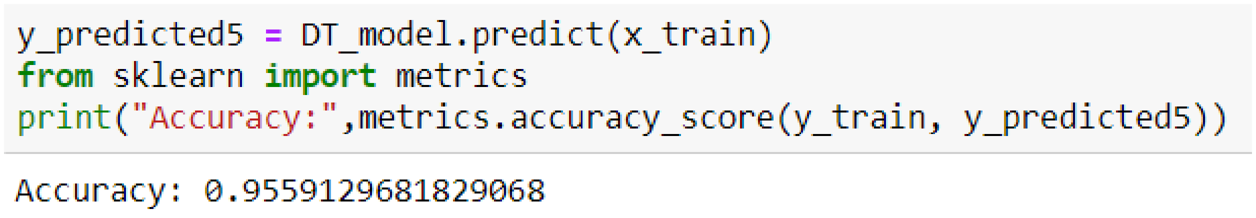
```
DT_model.fit(x_train,y_train)
```

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=20,
            max_features='auto', max_leaf_nodes=10,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=500, min_samples_split=500,
            min_weight_fraction_leaf=0.0, presort=False, random_state=50,
            splitter='best')
```

Figure 40: Decision tree model fitting

Prediction made on train data

The model built is used to predict the target variable by using train data and the accuracy score obtained by the model is 95%.



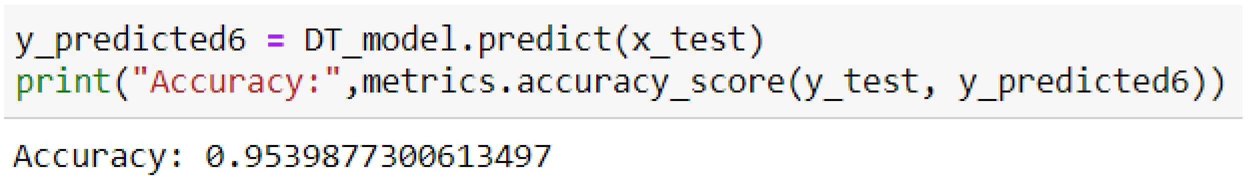
```
y_predicted5 = DT_model.predict(x_train)
from sklearn import metrics
print("Accuracy:",metrics.accuracy_score(y_train, y_predicted5))
```

```
Accuracy: 0.9559129681829068
```

Figure 41: Prediction on train data (decision tree)

Prediction made on test data

The model built is used to predict the target variable by using train data and the accuracy score obtained by the model is 86%.



```
y_predicted6 = DT_model.predict(x_test)
print("Accuracy:",metrics.accuracy_score(y_test, y_predicted6))
```

```
Accuracy: 0.9539877300613497
```

Figure 42: Prediction on test data (decision tree)

Confusion matrix

A confusion matrix is a summary of prediction results on a classification problem on a set of data for which the true values are known. The confusion matrix shows the ways in which your classification model is confused when it makes predictions. It gives us insight not only into the errors being made by a classifier but more importantly the types of errors that are being made.

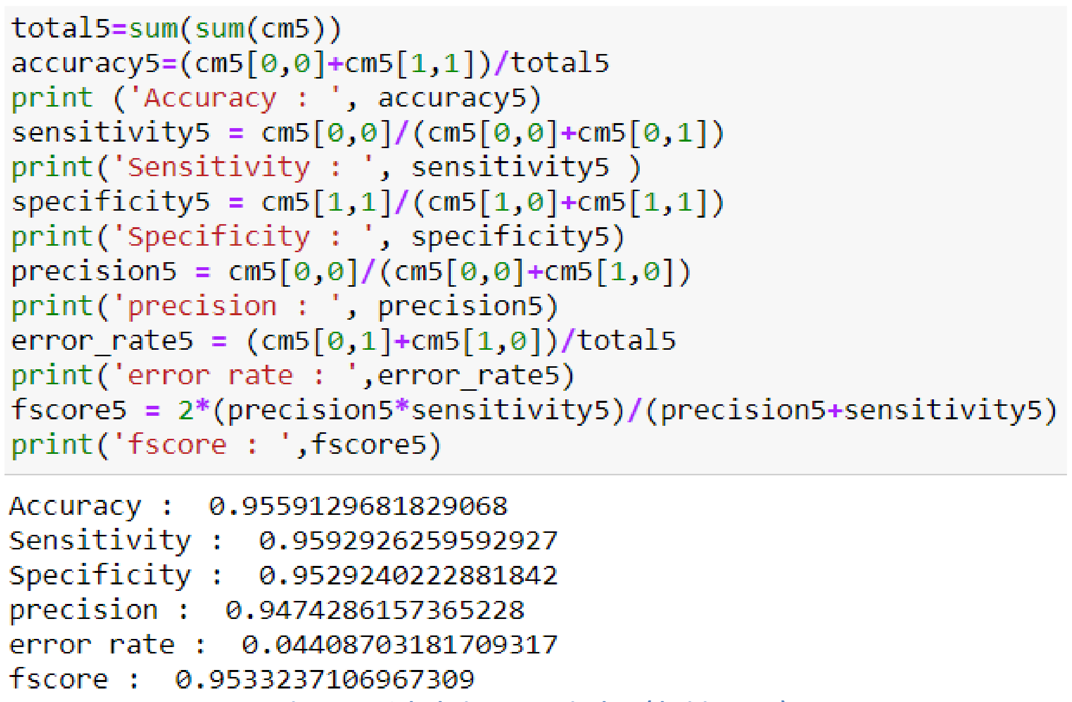
	normal	attack
normal	48875	2074
attack	2712	54897

	normal	attack
normal	16033	708
attack	957	18488

Figure 43: Confusion matrix on train data and test data

The below calculations are made using the confusion matrix:

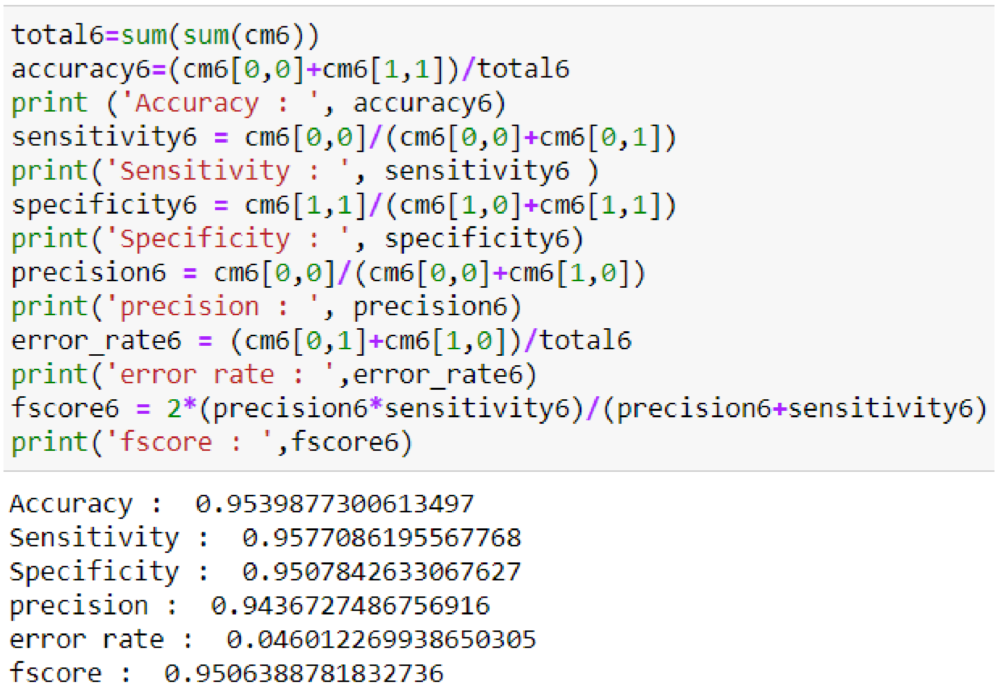
If any doubt regarding the calculations made refer to the page **28**



```
total5=sum(sum(cm5))
accuracy5=(cm5[0,0]+cm5[1,1])/total5
print ('Accuracy : ', accuracy5)
sensitivity5 = cm5[0,0]/(cm5[0,0]+cm5[0,1])
print('Sensitivity : ', sensitivity5 )
specificity5 = cm5[1,1]/(cm5[1,0]+cm5[1,1])
print('Specificity : ', specificity5)
precision5 = cm5[0,0]/(cm5[0,0]+cm5[1,0])
print('precision : ', precision5)
error_rate5 = (cm5[0,1]+cm5[1,0])/total5
print('error rate : ',error_rate5)
fscore5 = 2*(precision5*sensitivity5)/(precision5+sensitivity5)
print('fscore : ',fscore5)
```

```
Accuracy :  0.9559129681829068
Sensitivity :  0.9592926259592927
Specificity :  0.9529240222881842
precision :  0.9474286157365228
error rate :  0.04408703181709317
fscore :  0.9533237106967309
```

Figure 44: Calculations on train data (decision tree)



```
total6=sum(sum(cm6))
accuracy6=(cm6[0,0]+cm6[1,1])/total6
print ('Accuracy : ', accuracy6)
sensitivity6 = cm6[0,0]/(cm6[0,0]+cm6[0,1])
print('Sensitivity : ', sensitivity6 )
specificity6 = cm6[1,1]/(cm6[1,0]+cm6[1,1])
print('Specificity : ', specificity6)
precision6 = cm6[0,0]/(cm6[0,0]+cm6[1,0])
print('precision : ', precision6)
error_rate6 = (cm6[0,1]+cm6[1,0])/total6
print('error rate : ',error_rate6)
fscore6 = 2*(precision6*sensitivity6)/(precision6+sensitivity6)
print('fscore : ',fscore6)
```

```
Accuracy :  0.9539877300613497
Sensitivity :  0.9577086195567768
Specificity :  0.9507842633067627
precision :  0.9436727486756916
error rate :  0.046012269938650305
fscore :  0.9506388781832736
```

Figure 45: Calculations on test data (decision tree)

Auc-Roc curve

AUC - ROC curve is a performance measurement for the classification problem at various thresholds settings. It tells how much model is capable of distinguishing between classes. Higher the AUC, better the model is at predicting normal as normal and attack as attack.

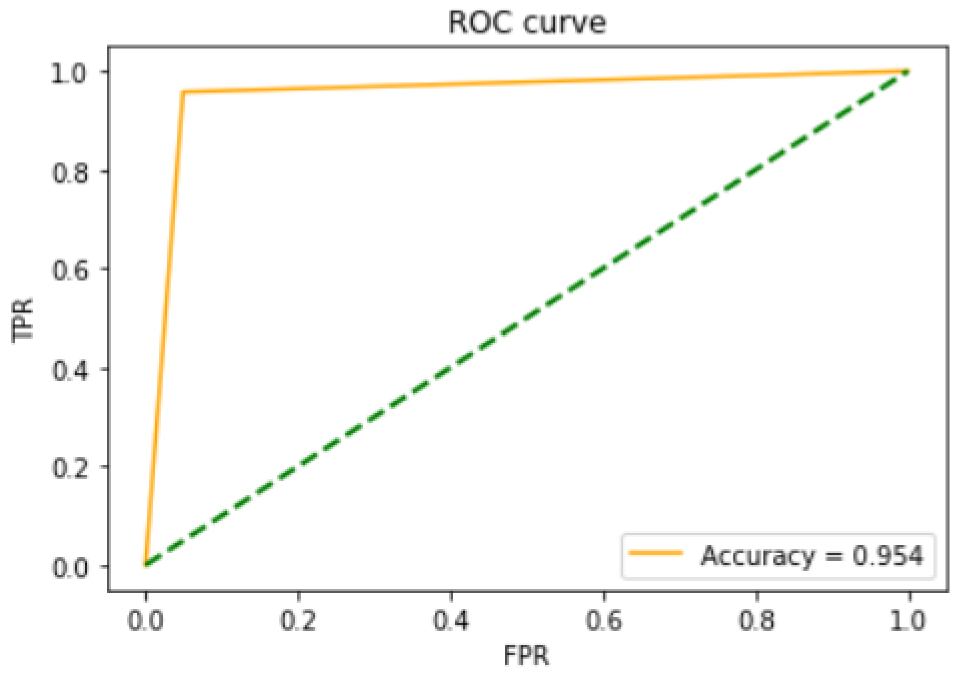


Figure 46: Auc-Roc curve (decision tree)

The ROC curve is plotted with TPR against the FPR where TPR is on y-axis and FPR is on the x-axis.

Plotting a decision tree

A decision tree is plotted by using **Graphviz** libraries

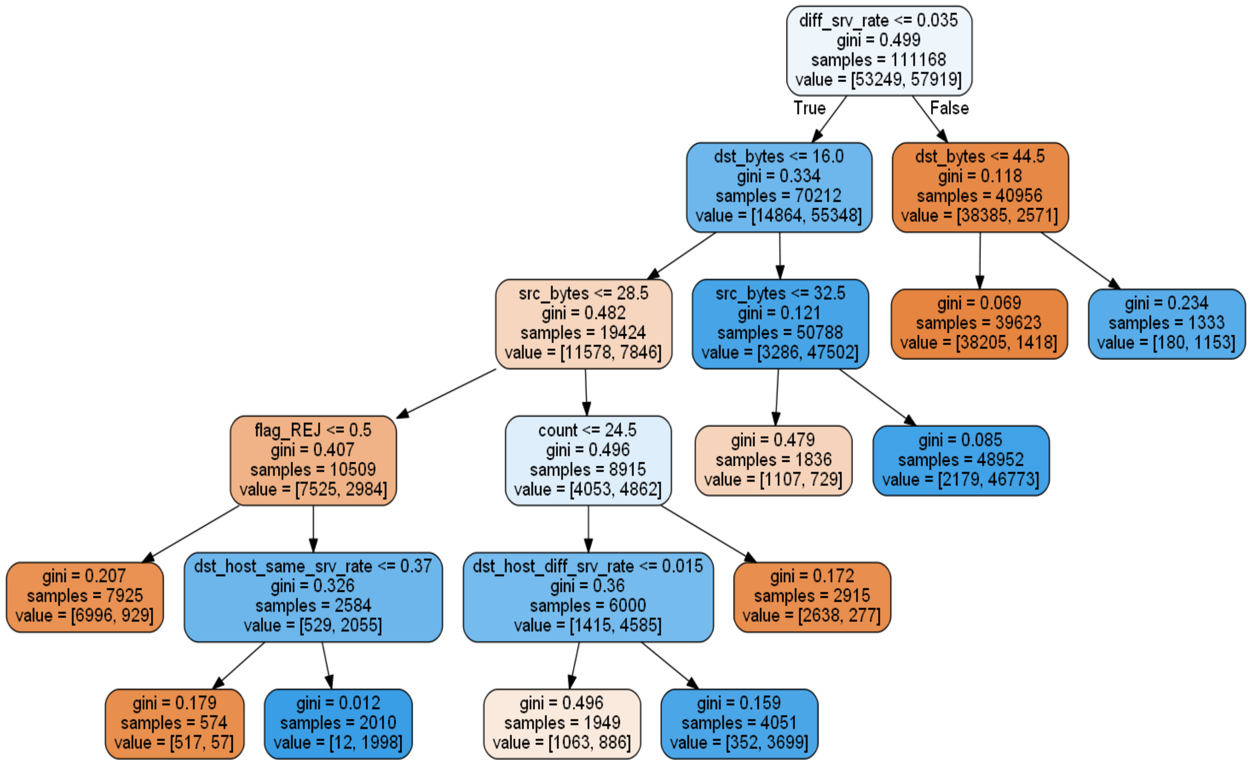


Figure 47: Decision tree plot

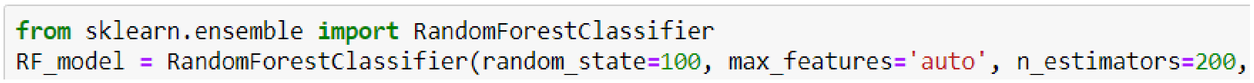
Next, Random forest algorithm is implemented to achieve a better an accuracy of the model.

Random Forest

Random forest, like its name implies, consists of a large number of individual decision trees that operate as an ensemble. Each tree in the random forest spits out a class prediction and the class with the most votes becomes our model's prediction. Ensemble methods use multiple learning models to gain better predictive results — in the case of a random forest, the model creates an entire forest of random uncorrelated decision trees to arrive at the best possible answer.

Parameters used

from sklearn.ensemble import RandomForestClassifer is used to implement the **Random Forest** algorithm.



```
from sklearn.ensemble import RandomForestClassifier
RF_model = RandomForestClassifier(random_state=100, max_features='auto', n_estimators=200,
```

Figure 48: Random forest model

In the above figure, a few parameters are specially used by mentioning particular values or features to each parameter. The parameters used are:

Random state, max_features, n_estimators, oob_score=True, n_jobs, min_samples_leaf, max_leaf_nodes, max_depth, min_samples_split.

Minimum samples for a node split

It defines the minimum number of samples (or observations) which are required in a node to be considered for splitting. It is used to control over-fitting. Higher values prevent a model from learning relations which might be highly specific to the particular sample selected for a tree.

Minimum samples for a terminal node (leaf)

Defines the minimum samples (or observations) required in a terminal node or leaf. It is used to control over-fitting similar to min_samples_split.

Maximum depth of the tree

It is the maximum depth of a tree. And is used to control over-fitting as higher depth will allow the model to learn relations very specific to a particular sample.

Maximum number of terminal nodes

It is the maximum number of terminal nodes or leaves in a tree. It can be defined in place of max_depth. Since binary trees are created, a depth of 'n' would produce a maximum of 2^n leaves.

Maximum features to consider for a split

It is the number of features to consider while searching for the best split. These will be randomly selected. If "auto" is chosen, it will simply take all the features which make sense in every tree. Here we simply do not put any restrictions on the individual tree.

Random state

If int, random_state is the seed used by the random number generator; If RandomState instance, random_state is the random number generator.

Oobs_score

It is a Random forest cross-validation method. In this sampling, about one-third of the data is not used to train the model and can be used to evaluate its performance. These samples are called out bag samples it is similar to the leave-one-out cross-validation method, but almost additional computation burden goes along with it.

N_estimators:
This is the number of trees you want to build before taking the maximum voting or averages of predictions. Higher the number of trees gives you better performance but makes your code slower. You should choose as high value as your processor can handle because this makes your predictions stronger and more stable.

N_job's

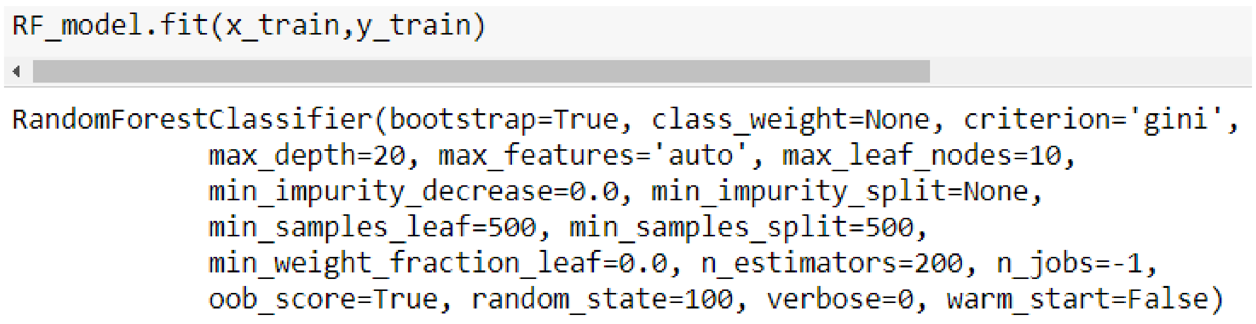
It tells the engine how many processors it is allocated to use if it has a value of 1, it can only use one processor, a value of -1 means that there is no limit. After making the model learn let's try to predict the target variable of the test and train dataset.

The unmentioned parameters: criterion, min_weight_fraction_leaf, min_impurity_decrease, min_impurity_split, bootstrap, verbose, warm_start, class_weight

The other unmentioned parameters without any particular values are considered in the model by their default value.

Model fitting

Since the data is processed, the model is trained using train data which contains a known output and the model learns on this data.



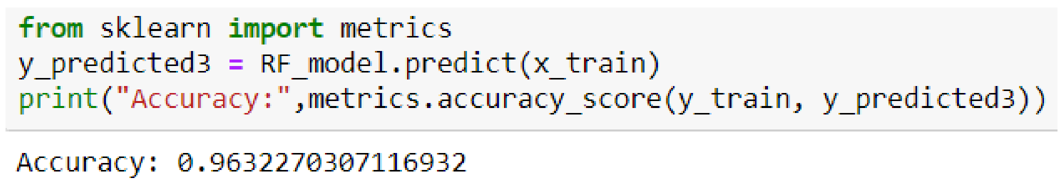
```
RF_model.fit(x_train,y_train)
```

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=20, max_features='auto', max_leaf_nodes=10,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=500, min_samples_split=500,
            min_weight_fraction_leaf=0.0, n_estimators=200, n_jobs=-1,
            oob_score=True, random_state=100, verbose=0, warm_start=False)
```

Figure 49: Random forest model fitting

Prediction made on train data

The model built is used to predict the target variable by using train data and the accuracy score obtained by the model is 95%.



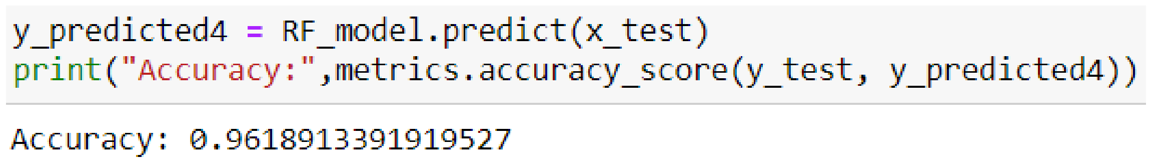
```
from sklearn import metrics
y_predicted3 = RF_model.predict(x_train)
print("Accuracy:",metrics.accuracy_score(y_train, y_predicted3))
```

```
Accuracy: 0.9632270307116932
```

Figure 50: Prediction of train data (random forest)

Prediction made on test data

The model built is used to predict the target variable by using train data and the accuracy score obtained by the model is 86%.



```
y_predicted4 = RF_model.predict(x_test)
print("Accuracy:",metrics.accuracy_score(y_test, y_predicted4))
```

```
Accuracy: 0.9618913391919527
```

Figure 51: Prediction of test data (random forest)

Confusion matrix

A confusion matrix is a summary of prediction results on a classification problem on a set of data for which the true values are known. The confusion matrix shows the ways in which your classification model is confused when it makes predictions. It gives us insight not only into the errors being made by a classifier but more importantly the types of errors that are being made.

Predicted /actual	normal	attack
normal	47875	2987
attack	1005	56691

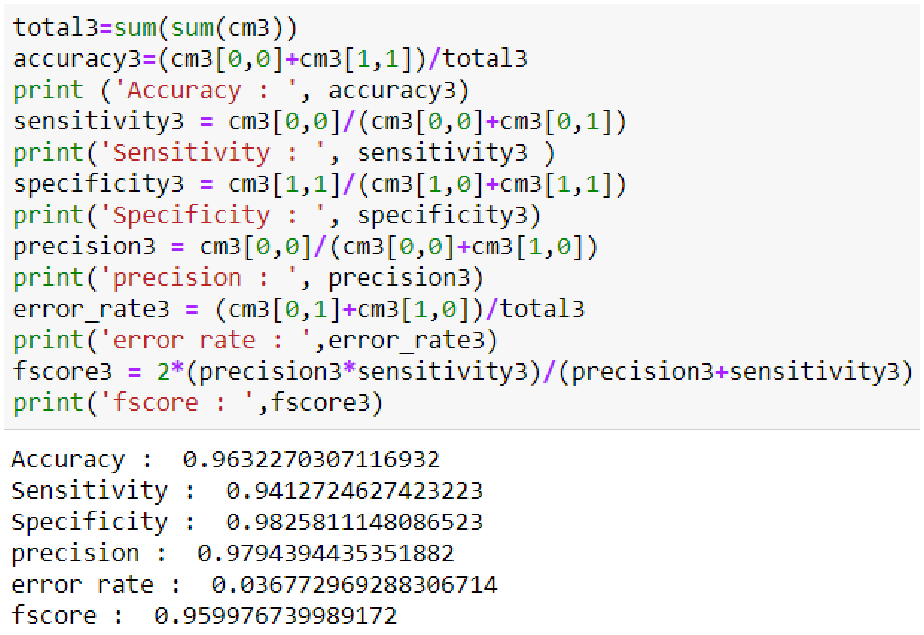
Predicted /actual	normal	attack
normal	15804	1023
attack	356	19003

Figure 52: Confusion matrix on train data and test data (RF)

The below calculations are made using the confusion matrix:

The below calculations are made using the confusion matrix:

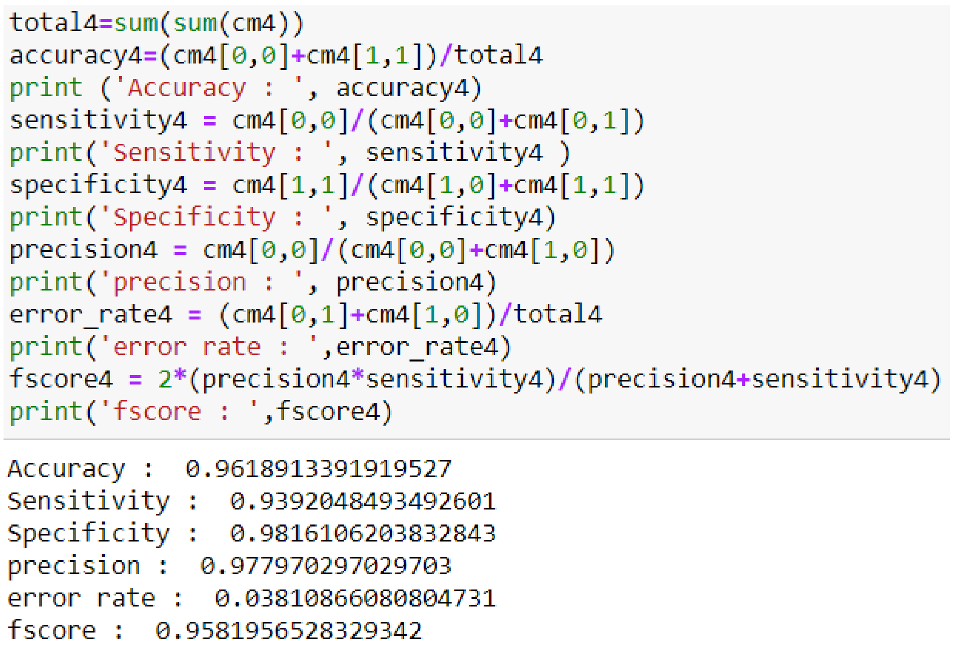
If any doubt regarding the calculations made refer to the page **28**



```
total3=sum(sum(cm3))
accuracy3=(cm3[0,0]+cm3[1,1])/total3
print ('Accuracy : ', accuracy3)
sensitivity3 = cm3[0,0]/(cm3[0,0]+cm3[0,1])
print('Sensitivity : ', sensitivity3 )
specificity3 = cm3[1,1]/(cm3[1,0]+cm3[1,1])
print('Specificity : ', specificity3)
precision3 = cm3[0,0]/(cm3[0,0]+cm3[1,0])
print('precision : ', precision3)
error_rate3 = (cm3[0,1]+cm3[1,0])/total3
print('error rate : ',error_rate3)
fscore3 = 2*(precision3*sensitivity3)/(precision3+sensitivity3)
print('fscore : ',fscore3)
```

```
Accuracy :  0.9632270307116932
Sensitivity :  0.9412724627423223
Specificity :  0.9825811148086523
precision :  0.9794394435351882
error rate :  0.036772969288306714
fscore :  0.959976739989172
```

Figure 53: Calculations made on train data (random forest)



```
total4=sum(sum(cm4))
accuracy4=(cm4[0,0]+cm4[1,1])/total4
print ('Accuracy : ', accuracy4)
sensitivity4 = cm4[0,0]/(cm4[0,0]+cm4[0,1])
print('Sensitivity : ', sensitivity4 )
specificity4 = cm4[1,1]/(cm4[1,0]+cm4[1,1])
print('Specificity : ', specificity4)
precision4 = cm4[0,0]/(cm4[0,0]+cm4[1,0])
print('precision : ', precision4)
error_rate4 = (cm4[0,1]+cm4[1,0])/total4
print('error rate : ',error_rate4)
fscore4 = 2*(precision4*sensitivity4)/(precision4+sensitivity4)
print('fscore : ',fscore4)
```

```
Accuracy :  0.9618913391919527
Sensitivity :  0.9392048493492601
Specificity :  0.9816106203832843
precision :  0.977970297029703
error rate :  0.03810866080804731
fscore :  0.9581956528329342
```

Figure 54: Calculations made on test data (random forest)

Auc-Roc curve

AUC - ROC curve is a performance measurement for the classification problem at various thresholds settings. It tells how much model is capable of distinguishing between classes. Higher the AUC, better the model is at predicting normal as normal and attack as attack.

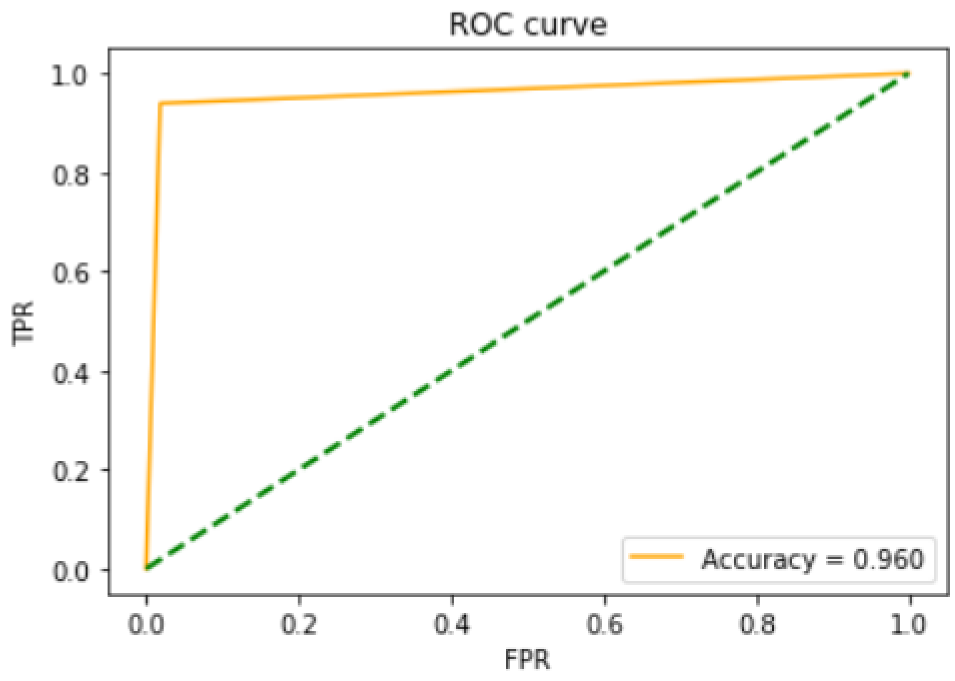


Figure 55: Auc-Roc curve (random forest)

The ROC curve is plotted with TPR against the FPR where TPR is on y-axis and FPR is on the x-axis.

K-FOLD CROSS-VALIDATION

K-Fold consists of splitting the data into K partitions of equal size. For each partition I, the model is trained with the remaining K-1 partitions and it is evaluated on partition i. The final score is the average of the K scored obtained. This technique is especially helpful when the performance of the model is significantly different from the train-test split.

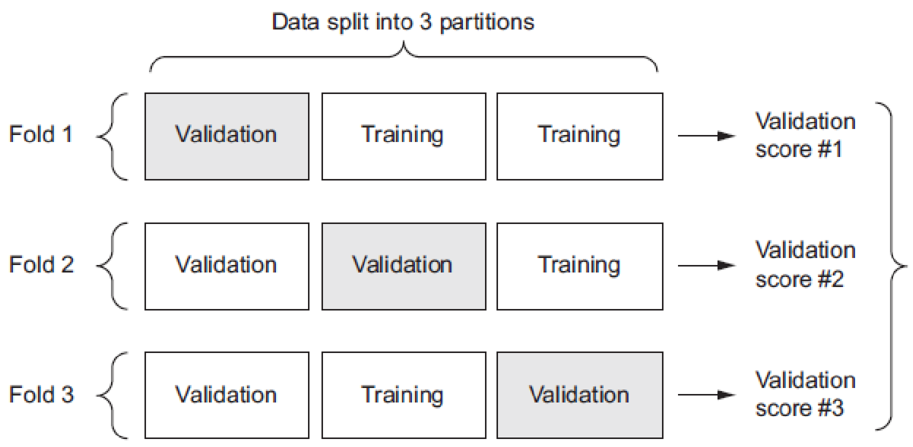
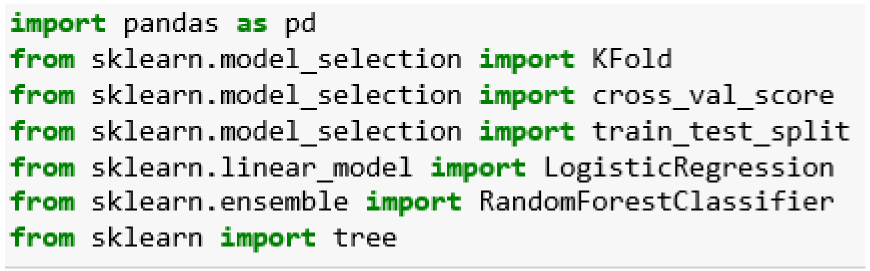


Figure 56: Kfold example

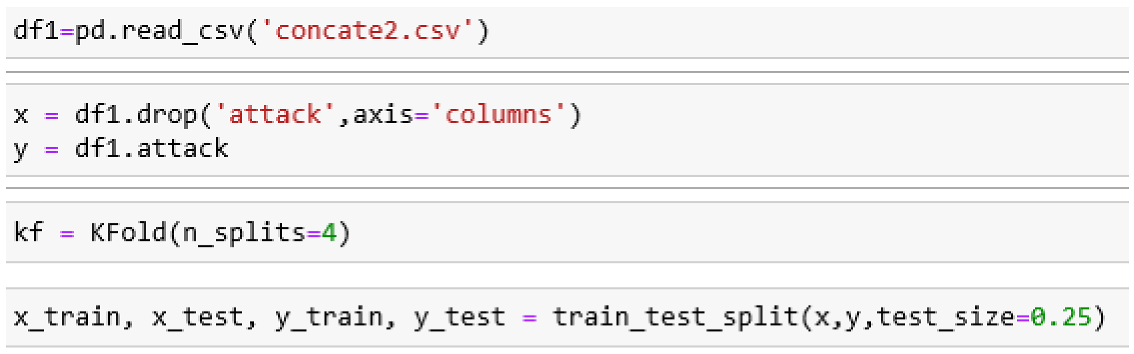
Implementing Kfold cross-validation



```
import pandas as pd
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn import tree
```

Figure 57: libraries imported (kfold)

In the below figure, all the required libraries are imported to implement k fold cross-validation on Logistic regression, Decision tree, and Random forest



```
df1=pd.read_csv('concate2.csv')
```

```
x = df1.drop('attack',axis='columns')
y = df1.attack
```

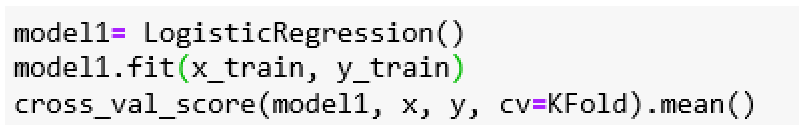
```
kf = KFold(n_splits=4)
```

```
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.25)
```

Figure 58: Kfold implementation

The concatenated data is read into the new data frame and the x and y terms consist of the features that are selected. The no of folds mentioned are 4 and the data is split into train and test at 75% and 25%.

Logistic Regression



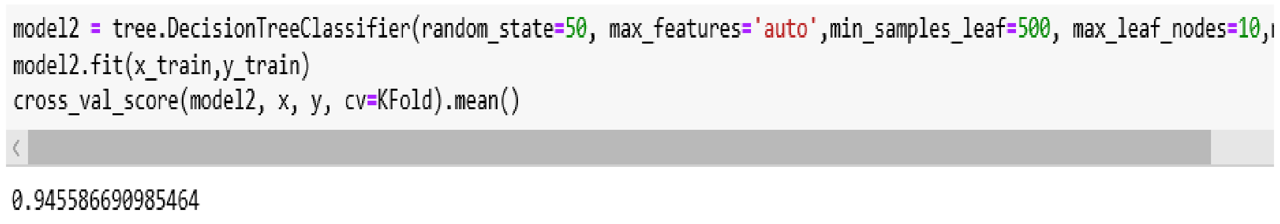
```
model1= LogisticRegression()
model1.fit(x_train, y_train)
cross_val_score(model1, x, y, cv=KFold).mean()
```

Figure 59: Kfold CV logistic regression

```
0.8672000221079976
```

The mean of cross-validation score obtained is 86%.

Decision Tree



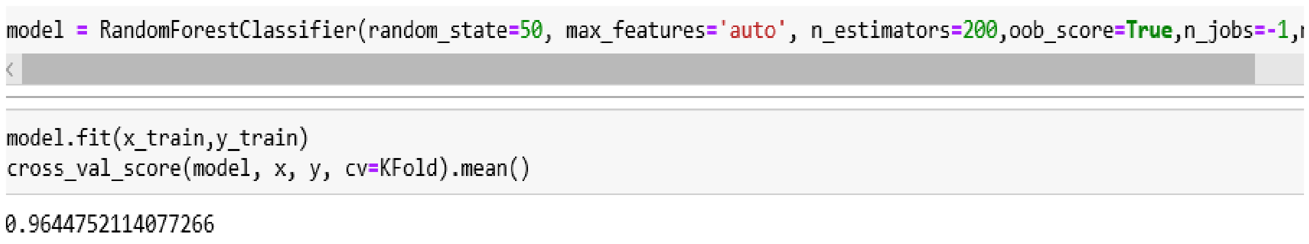
```
model2 = tree.DecisionTreeClassifier(random_state=50, max_features='auto',min_samples_leaf=500, max_leaf_nodes=10,
model2.fit(x_train,y_train)
cross_val_score(model2, x, y, cv=KFold).mean()
```

```
0.945586690985464
```

Figure 60: Kfold CV decision tree

The parameters used in the decision tree are the same as the ones used during model fitting. The mean of cross-validation score obtained is 94%.

Random Forest



```
model = RandomForestClassifier(random_state=50, max_features='auto', n_estimators=200,oob_score=True,n_jobs=-1,
```

```
model.fit(x_train,y_train)
cross_val_score(model, x, y, cv=KFold).mean()
```

```
0.9644752114077266
```

Figure 61: Kfold CV random forest

The parameters used in the random forest are the same as the ones used during model fitting. The mean of cross-validation score obtained is 96%.

RESULT

According to all the models used for model fitting, the models that fit best are **Random forest** and **Decision tree**. This conclusion is reached on the bases of:

- Accuracy calculated for both train and test data
- Calculation of accuracy, precision, sensitivity, specificity, error rate from the confusion matrix
- K fold cross validation

Random Forest	**Decision Tree**
Accuracy of model • Train: 96% • Test: 96%	Accuracy of model • Train: 95% • Test: 95%
Accuracy (CF): 96%	Accuracy (CF): 95%
Precision: 97%	Precision: 94%
Sensitivity: 93%	Sensitivity: 95%
Specificity: 98%	Specificity: 95%
Error rate: 3%	Error rate: 4%
K fold CV: 96%	K fold CV: 94%

Here **CF** is confusion matrix and **CV** is cross-validation.

Since all the above calculations made on Random forest and Decision tree are almost nearby so both of them are considered to the best fit models used on the data.

WEEKLY REPORT

TIME FRAME	PROPOSED SCHEDULE
WEEK 1 (22-05-19 TO 24-05-19)	-Introduction -Project Introduction
WEEK 2 (27-05-19 TO 31-05-19)	-learning of the basics of python -learning of the numpy and pandas package -practice the pandas and numpy commands in Jupyter notebook
WEEK 3 (03-06-19 TO 07-06-19)	-learning basics of data science -understanding different machine learning algorithms
WEEK 4 (10-06-19 TO 14-06-19)	-Understanding the data -Identifying the output variable -cleaning and preparing the dataset -Identifying which algorithm to be used
WEEK 5 (17-06-19 TO 21-06-19)	-Application of different classification model and trained the data -accuracy of performance of each algorithm obtained
WEEK 6 (24-06-19 TO 28-06-19)	-Learning about chatbot -Learning about the most commonly used chatbots -Implementation of Chatbot
WEEK 7 (01-07-19 TO 05-07-19)	-Worked on the report of project 1 and project 2
WEEK 8 (08-07-19 TO 12-07-19)	-Understanding the Wholesale customer data -cleaning and preparing of data -Application of Kmean algorithm
WEEK 9 (15-06-19 TO 19-06-19)	Worked on the report of project 3

CONCLUSION

After the completion of the project: Network anomaly detection, I got to learn about the network and different kinds of attacks faced by it. This project has helped obtain the basics of data science and how the data is analyzed in real time.

The project aimed to predict the target variable by binomial classification. The data given was understood, prepared and analyzed. A statistical model was successfully built, generating a final equation to predict if the activity of the network was an attack or normal. The model predicted the activity of the network with an accuracy of 96.5%.

Undergoing this internship experience has made sure that a lot was learned about working within deadlines in a professional environment.

RESOURCES USED

The resources used in this project are

- IDE'S – Jupyter Notebook
- Software – Microsoft Excel
- Programming Languages -Python

REFERENCE

- https://towardsdatascience.com/methods-for-dealing-with-imbalanced-data-5b761be45a18
- https://towardsdatascience.com/multi-class-text-classification-with-scikit-learn12f1e60e0a9f
- https://www.geeksforgeeks.org/python-lemmatization-with-nltk/
- https://www.geeksforgeeks.org/tokenize-text-using-nltk-python/
- https://www.analyticsvidhya.com/blog/2017/03/imbalanced-classification-problem/
- https://www.lexalytics.com/lexablog/text-analytics-functions-explained
- https://scikit-learn.org/stable/