Configuration Manual

Academic Internship
MSc in Cyber Security

Catherine Stack
Student ID: x20178573

School of Computing
National College of Ireland

Supervisor: Vikas Sahni

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Catherine Stack		
Student ID:	x20178573		
Programme:	MSc in Cyber Security	**Year:**	2023
Module:	Academic Internship		
Lecturer:	Vikas Sahni		
Submission Due Date:	25 April 2023		
Project Title:	A critical analysis of Machine Learning Algorithms for detecting Distributed Denial of Service attacks		
Word Count:	1607	**Page Count:**	**14**

Signature: [illegible]

Date: April 24th 2023

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual
A critical analysis of machine learning algorithms for detecting Distributed Denial of Service attacks

Catherine Stack
Student ID: x20178573

1 Introduction

This configuration manual provides an overview of the hardware and software used during the implementation of this research project and also the requirements needed to implement and run the research project. The manual then outlines details involved in the data pre-processing, construction of data subsets, training the machine learning models and finally testing the models. Also contained in the manual are some python snippets of code used during the development and analysis phases of the research project. The evaluation section outlines the of the results of the findings from the work on: "*A critical analysis of Machine Learning Algorithms for detecting Distributed Denial of Service attacks*"

The results show that out of the five ML algorithms assessed, Random Forest, and K-Nearest Neighbour satisfied the problem statement goal of predicting 95% or greater accuracy. The Random Forest classifier performed the best overall with a 99% accuracy followed by K-NN with 96% accuracy. Logistic Regression performing least favourably with a 50% accuracy with Naïve Bayes and Decision Tree having an 85% and 92% respective accuracy percentage rates. Training time on the other hand had the Random Forest classification model perform poor with a time of 422ms recorded which was many times slower than that of all of the other classification models with KNN perform the quickest with 15.6ms and the remaining having a time of 31.2ms each.

2 System Configuration

The system hardware and the software that were used during the project research was of personal use and these are as follows:

2.1 Hardware Configuration

- Operating system: Windows 10 Home
- Processor: Intel i5-7300 CPU @ 2.60GHz

- System Type: 64-bit operating system, x64-based processor
- Hard Disk: Hybrid (256GB SSD + 1 TB HDD)
- Installed physical memory RAM: 16GB

2.2 Software Configurations:

Table 1. Software tools specifications

Software/Tools	**Version**	**Details**
Python	3.9.12	To develop the model python is used in this project.
Anaconda Navigator	1.9.0	It is windows suitable platform that allows users computations, package management and model deployments. 64 bit
NumPy	1.19.5	It is an open-source tools used to perform complex mathematical problems in data.
Sci-Kit Learn	0.24.1	It is the library which is utilized for problems such as Classification, Regression as well as for data pre-processing. (scikit-learn: machine learning in Python — scikit-learn 0.24.2 documentation, 2021)

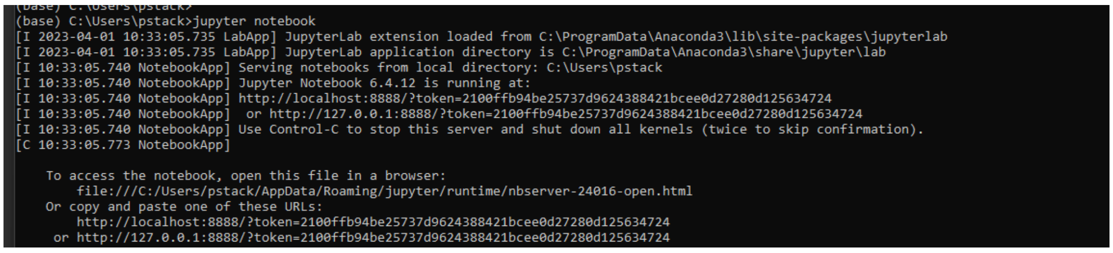
The Jupyter configuration of default memory resource allocation available on my environment was not sufficient for the initial loading of the datasets. This required the available memory to be increased so that the initial datafile “final_dataset.csv”. I updated the memory resource allocation from 2GB to 10GB. This is detailed in section 4.

3 Download and Implementation

This is a step-by-step guide to the run the project in any windows system.

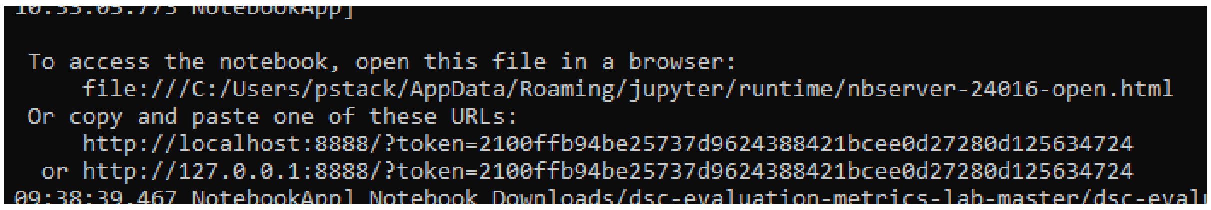
1. Download and install Anaconda Navigator software
   In this section the step-by-step guide is mentioned to run the project in any windows system. 1. Download and Install Anaconda Software in windows system.[1]

2. After download and install, go to start and go to Anaconda on the start menu and from there choose Anaconda command prompt.
3. Once open type ‘***jupyter notebook***’



```
(base) C:\Users\pstack>jupyter notebook
[I 2023-04-01 10:33:05.735 LabApp] JupyterLab extension loaded from C:\ProgramData\Anaconda3\lib\site-packages\jupyterlab
[I 2023-04-01 10:33:05.735 LabApp] JupyterLab application directory is C:\ProgramData\Anaconda3\share\jupyter\lab
[I 10:33:05.740 NotebookApp] Serving notebooks from local directory: C:\Users\pstack
[I 10:33:05.740 NotebookApp] Jupyter Notebook 6.4.12 is running at:
[I 10:33:05.740 NotebookApp] http://localhost:8888/?token=2100ffb94be25737d9624388421bcee0d27280d125634724
[I 10:33:05.740 NotebookApp]  or http://127.0.0.1:8888/?token=2100ffb94be25737d9624388421bcee0d27280d125634724
[I 10:33:05.740 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 10:33:05.773 NotebookApp]

    To access the notebook, open this file in a browser:
        file:///C:/Users/pstack/AppData/Roaming/jupyter/runtime/nbserver-24016-open.html
    Or copy and paste one of these URLs:
        http://localhost:8888/?token=2100ffb94be25737d9624388421bcee0d27280d125634724
     or http://127.0.0.1:8888/?token=2100ffb94be25737d9624388421bcee0d27280d125634724
```

4. If the Anaconda notebook interface doesn’t open in the default browser, open any browser and type http://localhost:8888/ as directed the logs on the anaconda command prompt:



```
10:33:03.773 NotebookApp]

 To access the notebook, open this file in a browser:
     file:///C:/Users/pstack/AppData/Roaming/jupyter/runtime/nbserver-24016-open.html
 Or copy and paste one of these URLs:
     http://localhost:8888/?token=2100ffb94be25737d9624388421bcee0d27280d125634724
  or http://127.0.0.1:8888/?token=2100ffb94be25737d9624388421bcee0d27280d125634724
09:38:39.467 NotebookApp] Notebook Downloads/dsc-evaluation-metrics-lab-master/dsc-evalu
```

5. From there navigate to the downloaded directory holding the project. (E.g. C:\Users\xxxx\Desktop\X20178573\X20178573\).

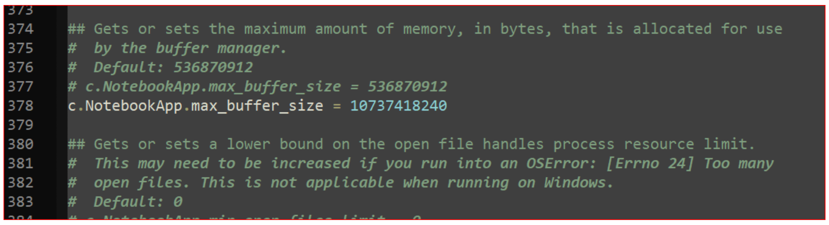
   This directory holds the jupyter files:

   - X20178573.ipynb and
   - x20178573_Main.ipynb

   Also in this directory is the smaller processed dataset .csv file processed_dataset.csv).

   Note* -** The initial dataset csv file (final_dataset.csv)*** was too large to include in this directory. The dataset can be downloaded from "***https://www.kaggle.com/datasets/devendra416/ddos-datasets?select=ddos_balanced/final_dataset/csv***"

   If downloading and importing the original dataset (***final_dataset.csv***) then the default memory allocation for Juypter will need to be increased from 3.2GB to approx. 16GB. This is done by the following altering the property ***NotebookApp.max_buffer_size*** in the file ***jupyter_notebook_config.p.***



```
## Gets or sets the maximum amount of memory, in bytes, that is allocated for use
#  by the buffer manager.
#  Default: 536870912
# c.NotebookApp.max_buffer_size = 536870912
c.NotebookApp.max_buffer_size = 10737418240

## Gets or sets a lower bound on the open file handles process resource limit.
#  This may need to be increased if you run into an OSError: [Errno 24] Too many
#  open files. This is not applicable when running on Windows.
#  Default: 0
```

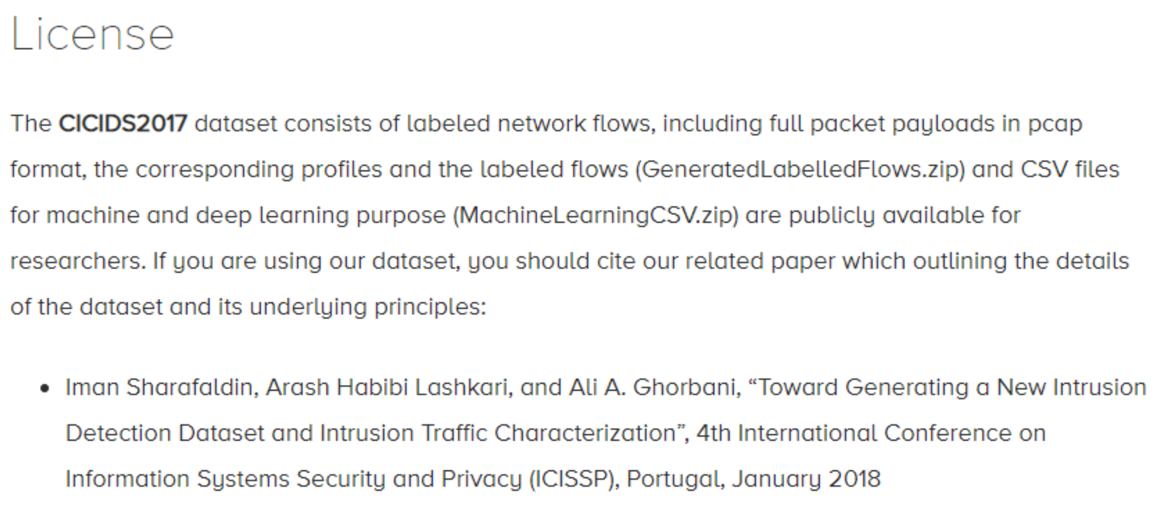
See Appendix 1 for details

6. The first jupyter notebook file is **X20178573.ipynb** and holds the initial dataset import, data cleaning and pre-processing python code.
   The file **x20178573_Main.ipynb** holds the classification model training and testing code.

3.1 Dataset

The dataset used in this study is an opensource dataset available on the Kaggle dataset repository website. It consists of an amalgamation of DDoS traffic extracted from three different Canadian Institute for Cybersecurity (CIC) datasets

- CIC DoS dataset(2016) : https://www.unb.ca/cic/datasets/dos-dataset.html;
- CSE-CIC-IDS2018-AWS: https://www.unb.ca/cic/datasets/ids-2017.html;
- CICIDS2017: https://www.unb.ca/cic/datasets/ids-2018.html



License

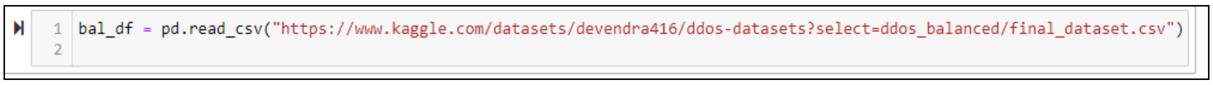
The **CICIDS2017** dataset consists of labeled network flows, including full packet payloads in pcap format, the corresponding profiles and the labeled flows (GeneratedLabelledFlows.zip) and CSV files for machine and deep learning purpose (MachineLearningCSV.zip) are publicly available for researchers. If you are using our dataset, you should cite our related paper which outlining the details of the dataset and its underlying principles:

- Iman Sharafaldin, Arash Habibi Lashkari, and Ali A. Ghorbani, "Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization", 4th International Conference on Information Systems Security and Privacy (ICISSP), Portugal, January 2018

4 Configuration and Exceution:

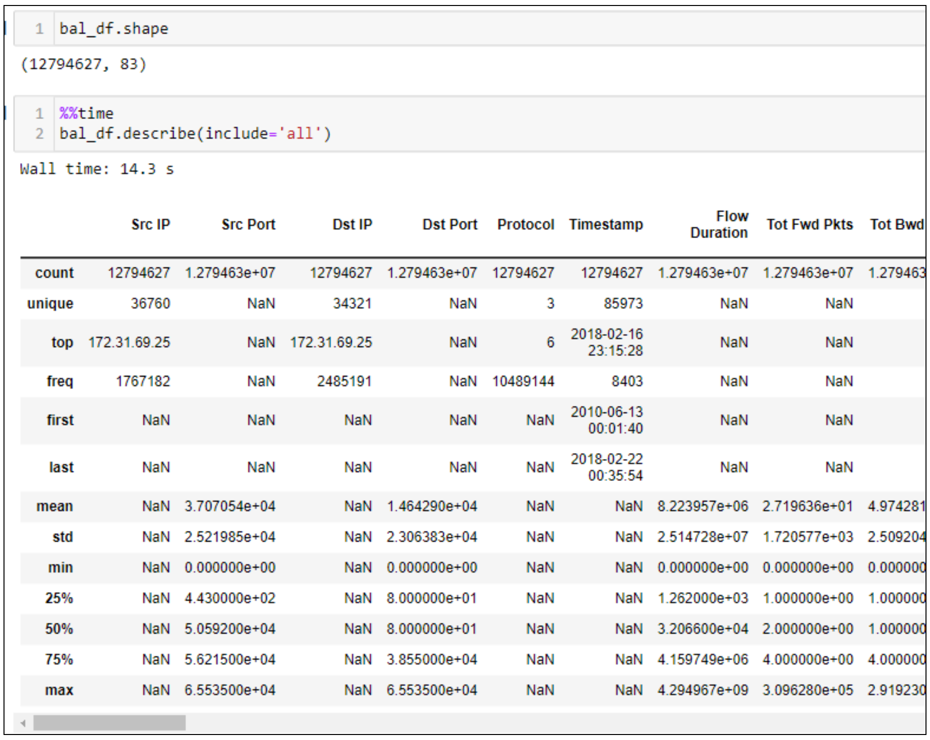
This research project implementation used the Python programming language on the Jupyter notebook platform utilising packages such as NumPy, pandas, Sklearn, Matplotlib, and Seaborn libraries. The hardware environment was ran on 64bit Windows operating system running Intel i5-7300 CPU and 16GB RAM.

Initial raw file imported and converting it to more readable format – final_dataset.csv



```
bal_df = pd.read_csv("https://www.kaggle.com/datasets/devendra416/ddos-datasets?select=ddos_balanced/final_dataset.csv")

```

The configuration of the default memory resource allocation available on my environment was not sufficient for the initial loading of the datasets. This required the available memory to be increased so this had to be increased from 2GB to 10GB for the final_dataset.csv to be read in. See Appendix 1 for details



```
bal_df.shape
```

```
(12794627, 83)
```

```
%%time
bal_df.describe(include='all')
```

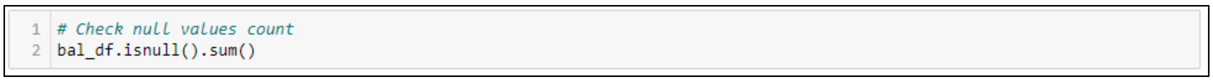
```
Wall time: 14.3 s
```

	Src IP	Src Port	Dst IP	Dst Port	Protocol	Timestamp	Flow Duration	Tot Fwd Pkts	Tot Bwd
count	12794627	1.279463e+07	12794627	1.279463e+07	12794627	12794627	1.279463e+07	1.279463e+07	1.279463
unique	36760	NaN	34321	NaN	3	85973	NaN	NaN	
top	172.31.69.25	NaN	172.31.69.25	NaN	6	2018-02-16 23:15:28	NaN	NaN	
freq	1767182	NaN	2485191	NaN	10489144	8403	NaN	NaN	
first	NaN	NaN	NaN	NaN	NaN	2010-06-13 00:01:40	NaN	NaN	
last	NaN	NaN	NaN	NaN	NaN	2018-02-22 00:35:54	NaN	NaN	
mean	NaN	3.707054e+04	NaN	1.464290e+04	NaN	NaN	8.223957e+06	2.719636e+01	4.974281
std	NaN	2.521985e+04	NaN	2.306383e+04	NaN	NaN	2.514728e+07	1.720577e+03	2.509204
min	NaN	0.000000e+00	NaN	0.000000e+00	NaN	NaN	0.000000e+00	0.000000e+00	0.000000
25%	NaN	4.430000e+02	NaN	8.000000e+01	NaN	NaN	1.262000e+03	1.000000e+00	1.000000
50%	NaN	5.059200e+04	NaN	8.000000e+01	NaN	NaN	3.206600e+04	2.000000e+00	1.000000
75%	NaN	5.621500e+04	NaN	3.855000e+04	NaN	NaN	4.159749e+06	4.000000e+00	4.000000
max	NaN	6.553500e+04	NaN	6.553500e+04	NaN	NaN	4.294967e+09	3.096280e+05	2.919230

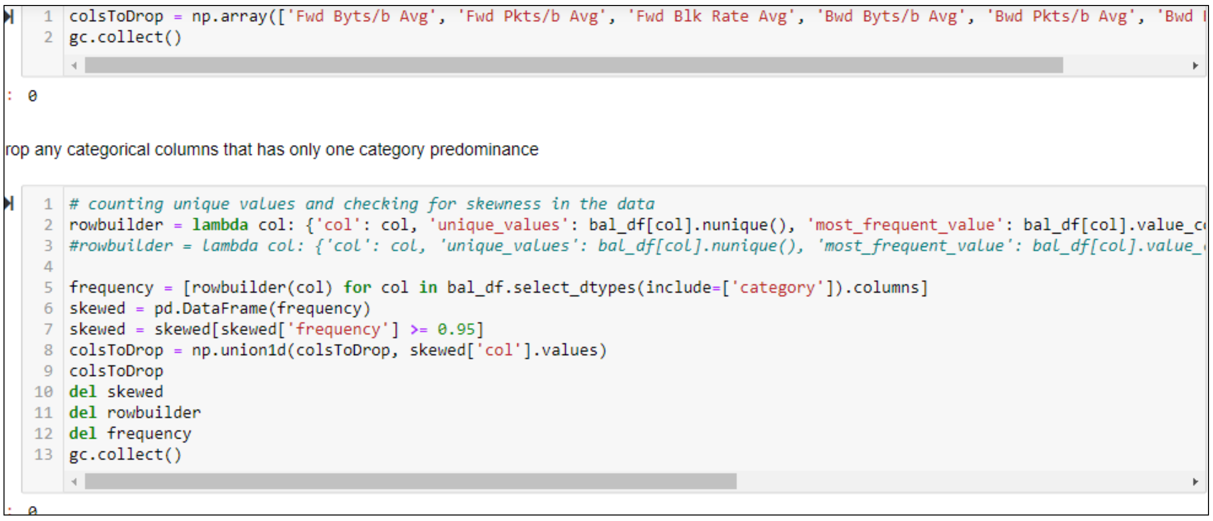
The size of the dataset file is 6.1 GB containing 12,794,627 observations and 83 columns. Due to hardware limitations, the dataset had to be reduced as the original volume of data was too large to process any of the classification models.

4.1 Data Pre-processing and feature extraction

Processing the data to a readable format for the classification models. Sample of checking for null values:



```
# Check null values count
bal_df.isnull().sum()
```



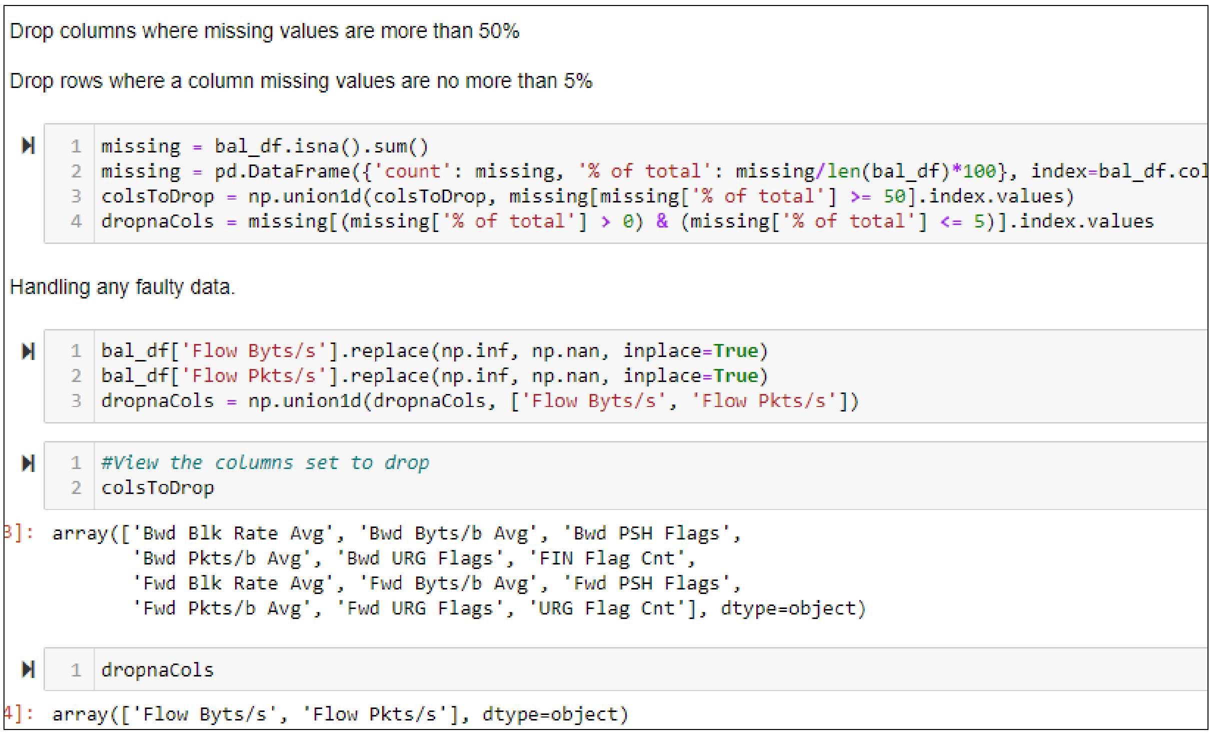
```
colsToDrop = np.array(['Fwd Byts/b Avg', 'Fwd Pkts/b Avg', 'Fwd Blk Rate Avg', 'Bwd Byts/b Avg', 'Bwd Pkts/b Avg', 'Bwd
gc.collect()
```

```
: 0
```

rop any categorical columns that has only one category predominance

```
# counting unique values and checking for skewness in the data
rowbuilder = lambda col: {'col': col, 'unique_values': bal_df[col].nunique(), 'most_frequent_value': bal_df[col].value_c
#rowbuilder = lambda col: {'col': col, 'unique_values': bal_df[col].nunique(), 'most_frequent_value': bal_df[col].value_

frequency = [rowbuilder(col) for col in bal_df.select_dtypes(include=['category']).columns]
skewed = pd.DataFrame(frequency)
skewed = skewed[skewed['frequency'] >= 0.95]
colsToDrop = np.union1d(colsToDrop, skewed['col'].values)
colsToDrop
del skewed
del rowbuilder
del frequency
gc.collect()
```

```
: 0
```


Drop columns where missing values are more than 50%

Drop rows where a column missing values are no more than 5%

```
missing = bal_df.isna().sum()
missing = pd.DataFrame({'count': missing, '% of total': missing/len(bal_df)*100}, index=bal_df.col
colsToDrop = np.union1d(colsToDrop, missing[missing['% of total'] >= 50].index.values)
dropnaCols = missing[(missing['% of total'] > 0) & (missing['% of total'] <= 5)].index.values
```

Handling any faulty data.

```
bal_df['Flow Byts/s'].replace(np.inf, np.nan, inplace=True)
bal_df['Flow Pkts/s'].replace(np.inf, np.nan, inplace=True)
dropnaCols = np.union1d(dropnaCols, ['Flow Byts/s', 'Flow Pkts/s'])
```

```
#View the columns set to drop
colsToDrop
```

```
3]:  array(['Bwd Blk Rate Avg', 'Bwd Byts/b Avg', 'Bwd PSH Flags',
            'Bwd Pkts/b Avg', 'Bwd URG Flags', 'FIN Flag Cnt',
            'Fwd Blk Rate Avg', 'Fwd Byts/b Avg', 'Fwd PSH Flags',
            'Fwd Pkts/b Avg', 'Fwd URG Flags', 'URG Flag Cnt'], dtype=object)
```

```
dropnaCols
```

```
4]:  array(['Flow Byts/s', 'Flow Pkts/s'], dtype=object)
```

Then high correlation features were checked for in order to perform the feature selection. Pearson's correlation coefficient was used to see measure of strength of association. The use of scatter plots and graph analysis were also used to identify correlated features.

The below code snippets outline the display of correlated features and the removal of some of those features.

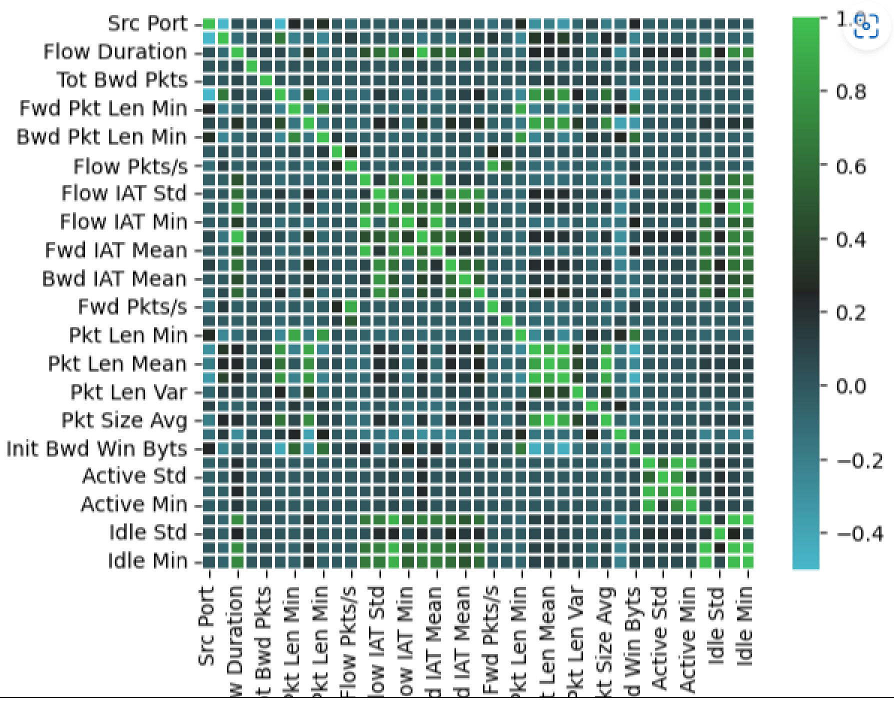
Using graph analysis to analyse the data

Examining the variables and the count of benign labels to ddos lables in the dataset.

Using Pearson's correlation coefficient to see measure of strength of association.

```python
#independent variable check
corr = ttsdf.iloc[:,:-1].corr(method="pearson")
cmap = sns.diverging_palette(210,490,85,69,center='dark',as_cmap=True)
sns.heatmap(corr, vmax=1, vmin=-.5, cmap=cmap, square=True, linewidths=.2)
```

```
1]:  <AxesSubplot:>
```



```python
correlatedFeatures = correlatedFeatures | getCorrelatedFeatures(corr)
print(correlatedFeatures)

```

```
{'Bwd IAT Max', 'Active Min', 'Idle Min', 'Pkt Len Std', 'Pkt Len Max', 'Bwd IAT Tot', 'Fwd IAT Mean', 'Fwd IAT Tot', 'Activ
e Max', 'Bwd Seg Size Avg', 'Subflow Fwd Byts', 'TotLen Bwd Pkts', 'Subflow Bwd Byts', 'Idle Mean', 'Fwd Act Data Pkts', 'Fw
d Header Len', 'Fwd Pkt Len Mean', 'Bwd Pkt Len Std', 'Bwd Pkt Len Mean', 'Fwd IAT Max', 'Pkt Len Min', 'Pkt Size Avg', 'Fwd
IAT Min', 'Pkt Len Mean', 'Fwd Seg Size Min', 'Fwd Pkts/s', 'Flow IAT Min', 'Subflow Bwd Pkts', 'Idle Max', 'Fwd Pkt Len St
d', 'Subflow Fwd Pkts', 'Bwd Header Len', 'Dst Port', 'Bwd IAT Min', 'TotLen Fwd Pkts', 'Fwd Seg Size Avg'}
```

Correlated features: {'Idle Mean', 'Fwd Pkts/s', 'Subflow Fwd Byts', 'Fwd IAT Min', 'Bwd Pkt Len Std', 'Active Max', 'Bwd Seg Size Avg', 'Fwd IAT Tot', 'Bwd IAT Max', 'Flow IAT Min', 'Pkt Len Mean', 'Idle Min', 'Subflow Bwd Byts', 'Fwd Act Data Pkts', 'TotLen Bwd Pkts', 'Pkt Len Std', 'Fwd IAT Max', 'Fwd Header Len', 'Fwd Pkt Len Mean', 'Bwd Pkt Len Mean', 'Bwd Header Len', 'Pkt Len Min', 'Bwd IAT Min', 'Fwd Pkt Len Std', 'Subflow Fwd Pkts', 'Pkt Len Max', 'TotLen Fwd Pkts', 'Bwd IAT Tot', 'Fwd IAT Mean', 'Pkt Size Avg', 'Fwd Seg Size Avg', 'Subflow Bwd Pkts', 'Fwd Seg Size Min', 'Idle Max', 'Active Min'}

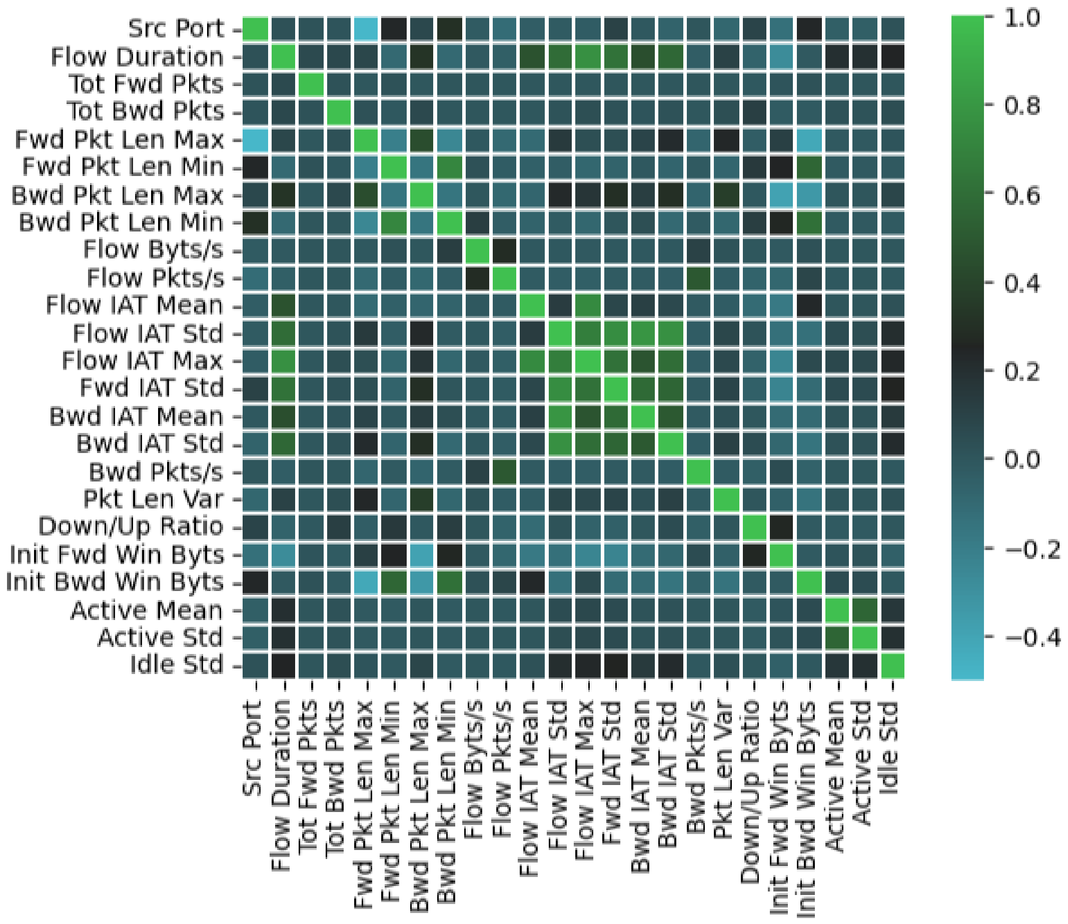
```python
#dropping correlated feature listed above

correlatedFeatures = correlatedFeatures | getCorrelatedFeatures(corr)
ttsdf.drop(columns=getCorrelatedFeatures(corr), inplace=True)
```

```python
ttsdf.shape
#41 features now remaining
```

```
[64]:  (10197152, 41)
```

```python
#independent variable check
corr = ttsdf.iloc[:,:-1].corr(method="pearson")
cmap = sns.diverging_palette(210,490,85,69,center='dark',as_cmap=True)
sns.heatmap(corr, vmax=1, vmin=-.5, cmap=cmap, square=True, linewidths=.2)
```

```
<AxesSubplot:>
```

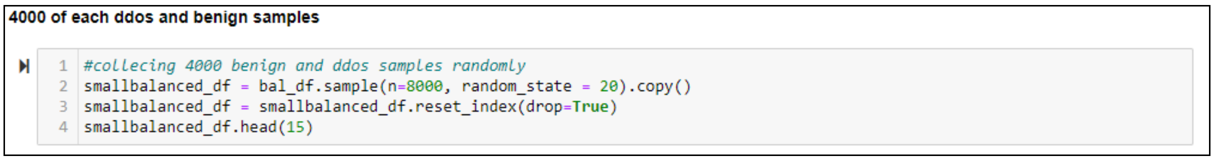


```python
correlatedFeatures = correlatedFeatures | getCorrelatedFeatures(corr)
print(correlatedFeatures)

```

```
{'Bwd IAT Max', 'Active Min', 'Bwd Header Len', 'Fwd IAT Tot', 'Fwd Act Data Pkts', 'Fwd Header Len', 'Fwd
wd Pkt Len Std', 'Fwd IAT Min', 'Fwd Seg Size Min', 'Bwd IAT Min', 'Pkt Len Max', 'TotLen Fwd Pkts', 'Fwd 
le Min', 'Pkt Len Std', 'Bwd IAT Tot', 'Fwd IAT Mean', 'Active Max', 'Bwd Seg Size Avg', 'Subflow Fwd Byts
s', 'Subflow Bwd Byts', 'Idle Mean', 'Bwd Pkt Len Mean', 'Fwd IAT Max', 'Pkt Len Min', 'Pkt Size Avg', 'Pk
Pkts/s', 'Flow IAT Min', 'Idle Max', 'Fwd Pkt Len Std', 'Subflow Fwd Pkts', 'Subflow Bwd Pkts', 'Dst Port'
```

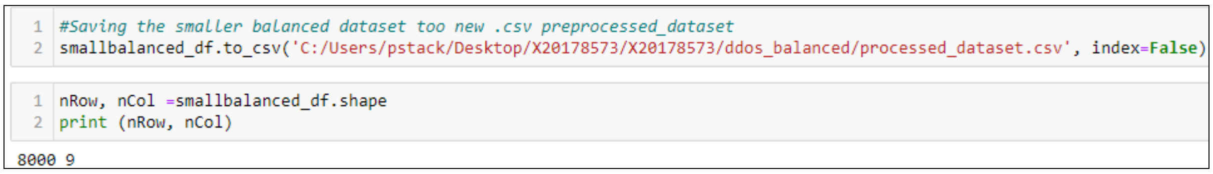
```python
# Going to drop correlated features
#drop multiple columns by name

ttsdf.drop(columns=getCorrelatedFeatures(corr), inplace=True)
```

Once the data was pre-processed a subset of the dataset was saved out to a smaller csv file. The new smaller file name below is '***processed_dataset.csv***'. The subset consists of 4000 benign and 4000 DDoS observations randomly selected so that the smaller dataset is manageable for processing due to hardware limitations.



4000 of each ddos and benign samples

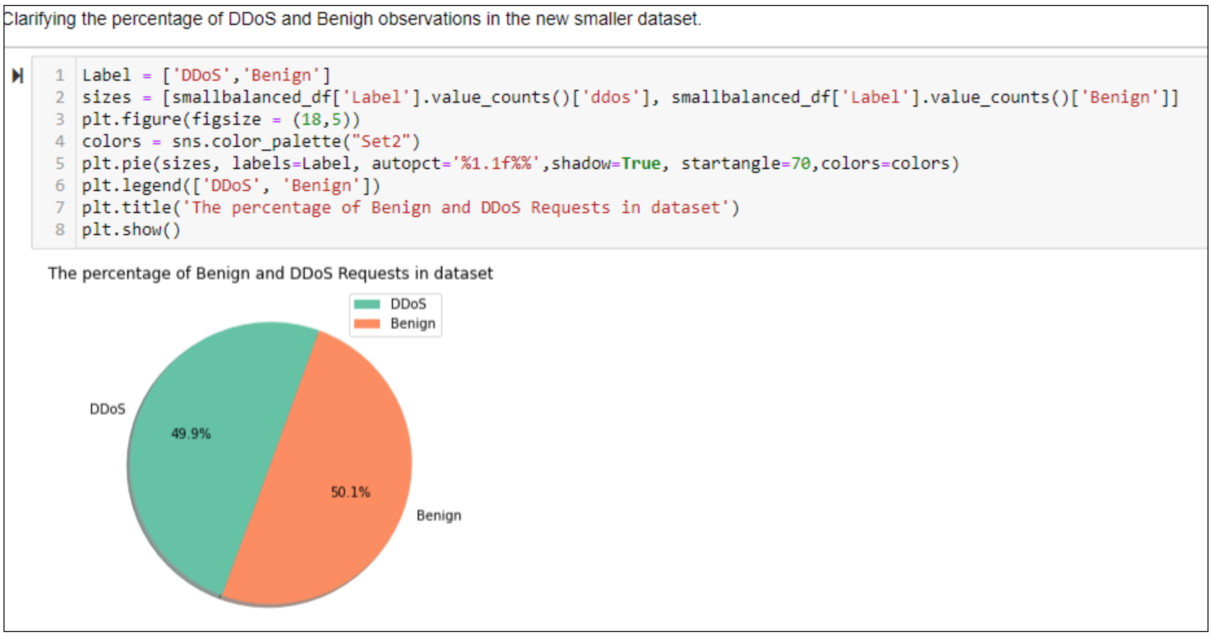
```
#collecing 4000 benign and ddos samples randomly
smallbalanced_df = bal_df.sample(n=8000, random_state = 20).copy()
smallbalanced_df = smallbalanced_df.reset_index(drop=True)
smallbalanced_df.head(15)
```



```
#Saving the smaller balanced dataset too new .csv preprocessed_dataset
smallbalanced_df.to_csv('C:/Users/pstack/Desktop/X20178573/X20178573/ddos_balanced/processed_dataset.csv', index=False)
```

```
nRow, nCol =smallbalanced_df.shape
print (nRow, nCol)
```

```
8000 9
```



Clarifying the percentage of DDoS and Benigh observations in the new smaller dataset.

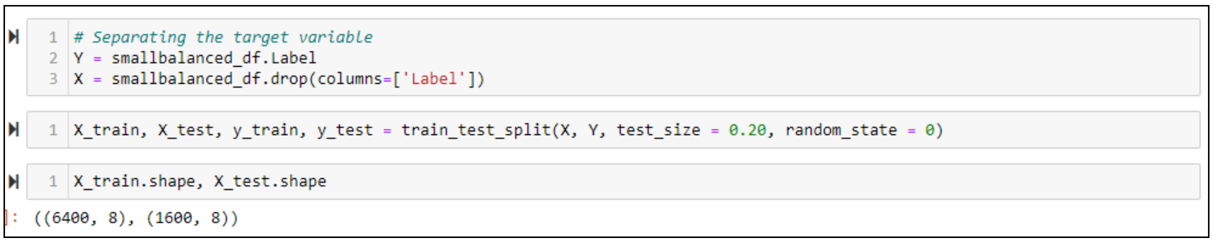
```
Label = ['DDoS','Benign']
sizes = [smallbalanced_df['Label'].value_counts()['ddos'], smallbalanced_df['Label'].value_counts()['Benign']]
plt.figure(figsize = (18,5))
colors = sns.color_palette("Set2")
plt.pie(sizes, labels=Label, autopct='%1.1f%%',shadow=True, startangle=70,colors=colors)
plt.legend(['DDoS', 'Benign'])
plt.title('The percentage of Benign and DDoS Requests in dataset')
plt.show()
```

This new smaller dataset will be used for the classification model training and testing.

5 Implementation

The file **x20178573_Main.ipynb** consists of the splitting of the dataset into train and test datasets with 80% and 20% respectively.

Splitting the data set then into train and test datasets.



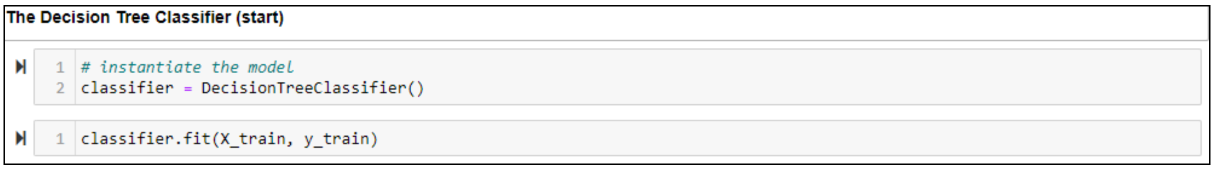
```
# Separating the target variable
Y = smallbalanced_df.Label
X = smallbalanced_df.drop(columns=['Label'])
```

```
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.20, random_state = 0)
```

```
X_train.shape, X_test.shape
```

```
]: ((6400, 8), (1600, 8))
```

After the splitting the classification models are fit and tested one at a time.
The Decision Tree classifier as a sample.

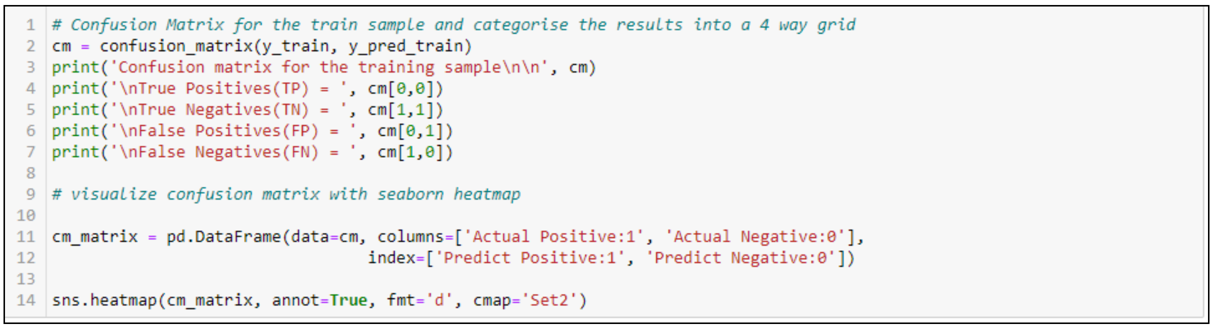


The Decision Tree Classifier (start)

```
# instantiate the model
classifier = DecisionTreeClassifier()
```

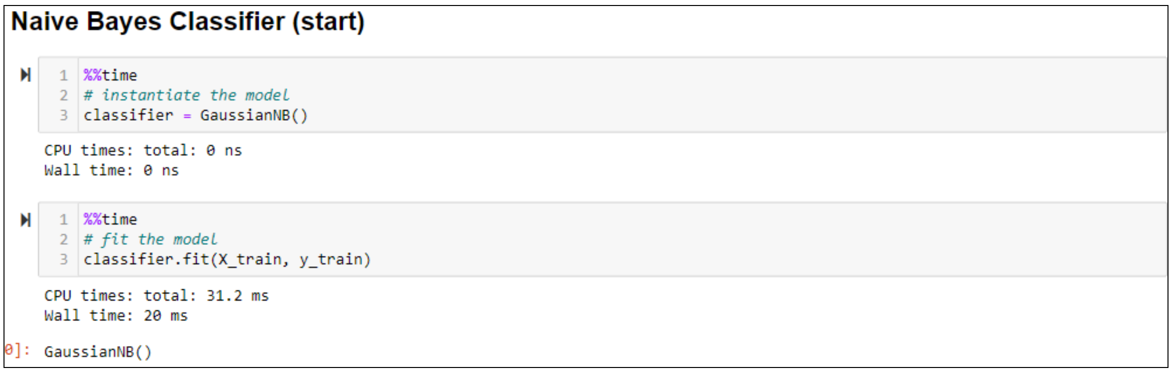
```
classifier.fit(X_train, y_train)
```

Testing for evaluation metrics:



```
# Confusion Matrix for the train sample and categorise the results into a 4 way grid
cm = confusion_matrix(y_train, y_pred_train)
print('Confusion matrix for the training sample\n\n', cm)
print('\nTrue Positives(TP) = ', cm[0,0])
print('\nTrue Negatives(TN) = ', cm[1,1])
print('\nFalse Positives(FP) = ', cm[0,1])
print('\nFalse Negatives(FN) = ', cm[1,0])

# visualize confusion matrix with seaborn heatmap

cm_matrix = pd.DataFrame(data=cm, columns=['Actual Positive:1', 'Actual Negative:0'],
                                 index=['Predict Positive:1', 'Predict Negative:0'])

sns.heatmap(cm_matrix, annot=True, fmt='d', cmap='Set2')
```

The Naïve Bayes classifier sample



Naive Bayes Classifier (start)

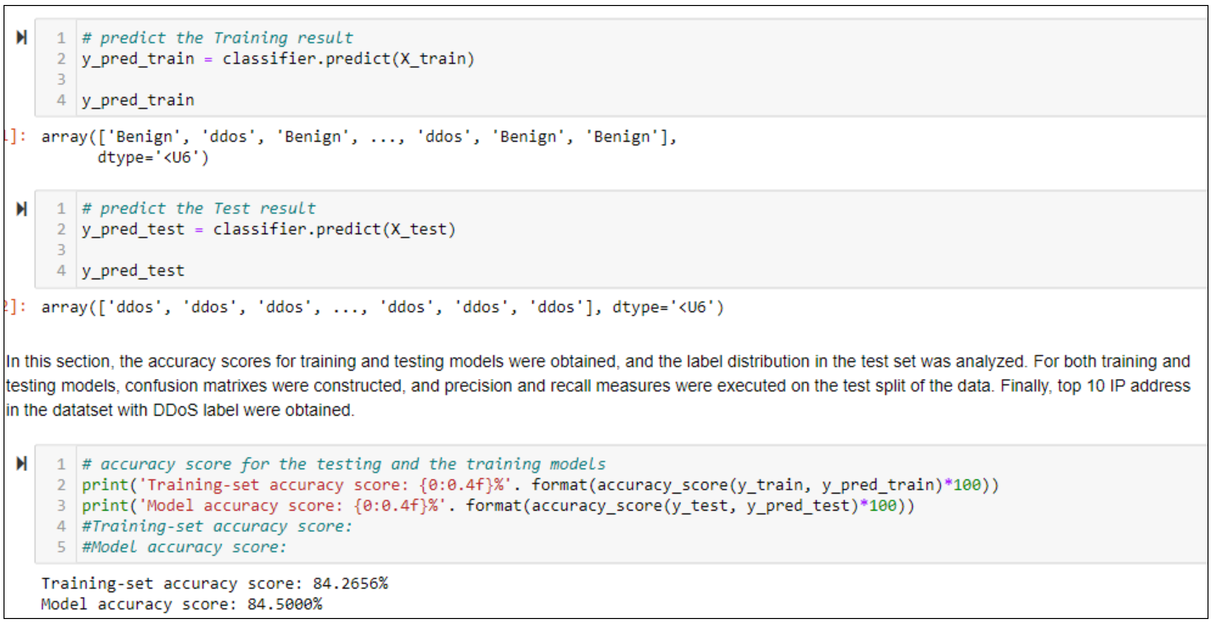
```
%%time
# instantiate the model
classifier = GaussianNB()
```

```
CPU times: total: 0 ns
Wall time: 0 ns
```

```
%%time
# fit the model
classifier.fit(X_train, y_train)
```

```
CPU times: total: 31.2 ms
Wall time: 20 ms
```

```
0]:  GaussianNB()
```



```
# predict the Training result
y_pred_train = classifier.predict(X_train)

y_pred_train
```

```
]:  array(['Benign', 'ddos', 'Benign', ..., 'ddos', 'Benign', 'Benign'],
          dtype='<U6')
```

```
# predict the Test result
y_pred_test = classifier.predict(X_test)

y_pred_test
```

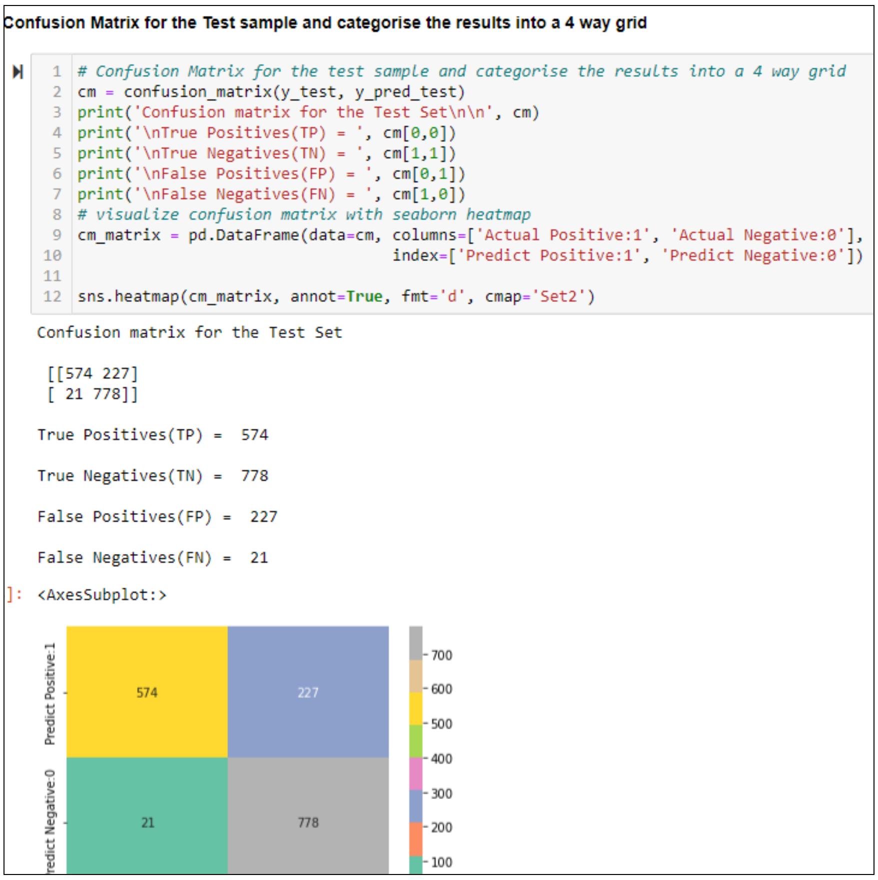
```
]:  array(['ddos', 'ddos', 'ddos', ..., 'ddos', 'ddos', 'ddos'], dtype='<U6')
```

In this section, the accuracy scores for training and testing models were obtained, and the label distribution in the test set was analyzed. For both training and testing models, confusion matrixes were constructed, and precision and recall measures were executed on the test split of the data. Finally, top 10 IP address in the datatset with DDoS label were obtained.

```
# accuracy score for the testing and the training models
print('Training-set accuracy score: {0:0.4f}%'. format(accuracy_score(y_train, y_pred_train)*100))
print('Model accuracy score: {0:0.4f}%'. format(accuracy_score(y_test, y_pred_test)*100))
#Training-set accuracy score:
#Model accuracy score:
```

```
Training-set accuracy score: 84.2656%
Model accuracy score: 84.5000%
```

Confusion Matrix for Naïve Bayes test sample



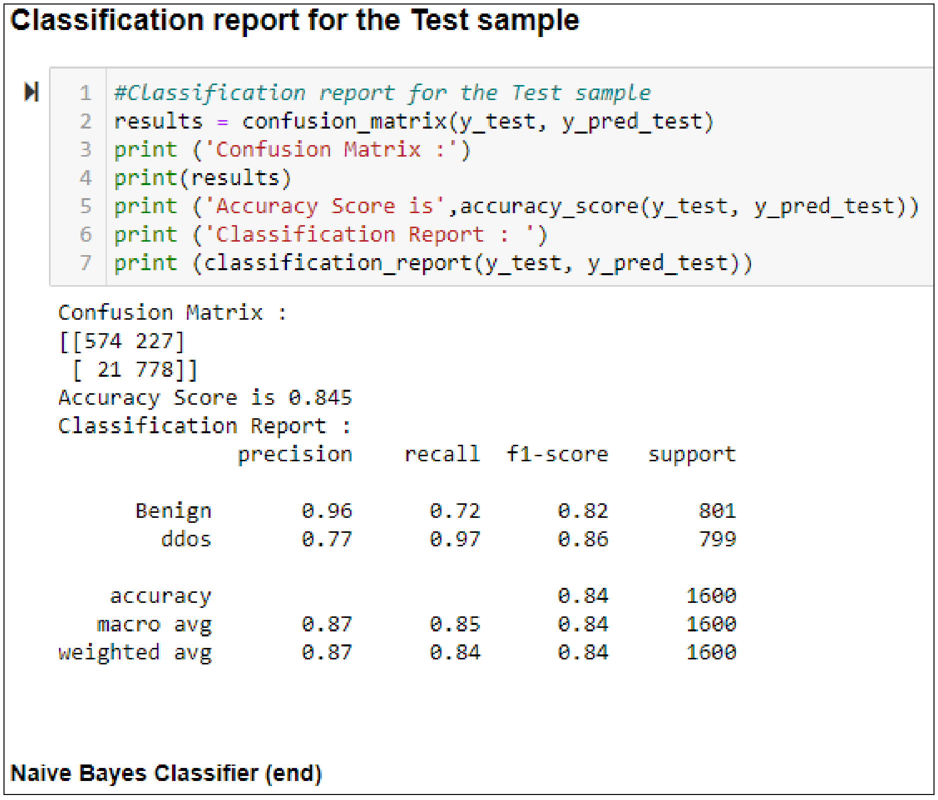
Confusion Matrix for the Test sample and categorise the results into a 4 way grid

```
# Confusion Matrix for the test sample and categorise the results into a 4 way grid
cm = confusion_matrix(y_test, y_pred_test)
print('Confusion matrix for the Test Set\n\n', cm)
print('\nTrue Positives(TP) = ', cm[0,0])
print('\nTrue Negatives(TN) = ', cm[1,1])
print('\nFalse Positives(FP) = ', cm[0,1])
print('\nFalse Negatives(FN) = ', cm[1,0])
# visualize confusion matrix with seaborn heatmap
cm_matrix = pd.DataFrame(data=cm, columns=['Actual Positive:1', 'Actual Negative:0'],
                                  index=['Predict Positive:1', 'Predict Negative:0'])

sns.heatmap(cm_matrix, annot=True, fmt='d', cmap='Set2')
```

```
Confusion matrix for the Test Set

 [[574 227]
 [ 21 778]]

True Positives(TP) =  574

True Negatives(TN) =  778

False Positives(FP) =  227

False Negatives(FN) =  21
```

```
]: <AxesSubplot:>
```

Classification report for the Naïve Bayes test sample



Classification report for the Test sample

```
#Classification report for the Test sample
results = confusion_matrix(y_test, y_pred_test)
print ('Confusion Matrix :')
print(results)
print ('Accuracy Score is',accuracy_score(y_test, y_pred_test))
print ('Classification Report : ')
print (classification_report(y_test, y_pred_test))
```

```
Confusion Matrix :
[[574 227]
 [ 21 778]]
Accuracy Score is 0.845
Classification Report :
              precision    recall  f1-score   support

      Benign       0.96      0.72      0.82       801
        ddos       0.77      0.97      0.86       799

    accuracy                           0.84      1600
   macro avg       0.87      0.85      0.84      1600
weighted avg       0.87      0.84      0.84      1600
```

Naive Bayes Classifier (end)

Model Execution Time:

The best time performer was KNN outperforming Naïve Bayes, Logistic Regression and Decision Tree by 50% taking 15.6ms over 31.2ms CPU time for each of the other three models. However, the poorest performer of training time lay with the Random Forest model taking 422.0ms CPU time which is substantially more than any of the other models and this is much more notable in the Figure 8 graph.

	Naive Bayes	K-nearest neighbour	Logistic Regression	Decision Tree	Random Forest	Average
CPU Time (ms)	31.20	15.60	31.20	31.20	422.00	106.24
Wall Time (ms)	20.00	35.00	31.30	26.00	433.00	109.06
Average	0.851	0.985	0.462	0.913	0.998	

References

[1] "Anaconda Navigator Distribution," *Anaconda Distribution*, May 06, 2022. https://www.anaconda.com/products/individual

Appendix 1

The original dataset (final_dataset.csv) will be too large for the default memory allocation for Juypter and thus the memory allocation will need to be increased from 3.2GB to approx. 16GB. This is done by the following altering the property NotebookApp.max_buffer_size in the file jupyter_notebook_config.py.

On the Anaconda command prompt:

1) Generate Config file using command: jupyter notebook --generate-config

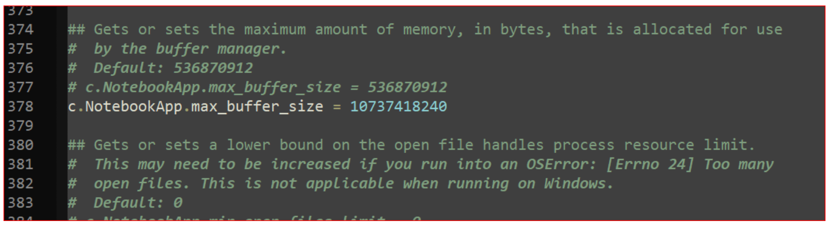
2) Open jupyter_notebook_config.py file situated inside 'jupyter' directory and edit the following property:

NotebookApp.max_buffer_size = your desired value

Remember to remove the '#' before the property value.

3) Save the file

In the command prompt run jupyter notebook and it should now utilize the set memory value.



```
## Gets or sets the maximum amount of memory, in bytes, that is allocated for use
#  by the buffer manager.
#  Default: 536870912
# c.NotebookApp.max_buffer_size = 536870912
c.NotebookApp.max_buffer_size = 10737418240

## Gets or sets a lower bound on the open file handles process resource limit.
#  This may need to be increased if you run into an OSError: [Errno 24] Too many
#  open files. This is not applicable when running on Windows.
#  Default: 0
```