AI Project

Member 1:

Name: Mahad Ashraf
Roll No: 20p-0563

Member 2:

Name: Muhammad Sherjeel Akhtar
Roll No: 20p-0101

Title: Finalized Project Of Artificial Intelligence With Report

Subject: Artificial Intelligence Lab

Section: BCS-6C

Submitted To Respect Ma'am:
*** Miss Hurmat Hidayat ***

REPORT:

Abstract:

The aim of this project was to apply different classification and clustering algorithms to classify cyber-attacks in network traffic. The dataset provided contained 23 different attack types, which were converted into five classes. The data was preprocessed by handling missing values, outliers, and feature scaling. The most relevant features were identified using correlation analysis. The classification algorithms used were decision trees, K-nearest neighbours, and artificial neural networks. The performance of each algorithm was evaluated using appropriate metrics such as accuracy, precision, recall, and F1-score. In the last task, clustering was performed using the k-means algorithm, and the results were visualized using a scatter plot.

Introduction:

The increasing use of the internet and technology has led to an increase in cyberattacks. It is important to develop methods to detect and classify these attacks to ensure the security of network traffic. In this project, we aim to develop classifiers for the prediction of attack types given their attributes. This project is divided into different tasks, including data preprocessing, feature engineering, classification using different algorithms, and clustering.

Data Preprocessing:

Data preprocessing is a crucial step in data analysis that involves cleaning, transforming, and preparing the dataset for analysis. The goal of data preprocessing is to ensure that the data is consistent, complete, and free of errors before performing any analysis.

In this report, we will discuss the data preprocessing techniques used to clean and transform the dataset for analysis. The dataset used in this report contains information about cyber attacks in network security.

Data Cleaning:

The dataset contained some missing values, which were handled using the mean value imputation technique. Outliers were identified using the boxplot method, and they were removed from the dataset.

Feature Scaling:

Feature scaling is a technique used to standardize the range of features in the dataset. In this report, we used standardization to scale the features in the dataset. The standardization process involves subtracting the mean value of the feature from each data point and then dividing the result by the standard deviation of the feature.

The preprocessed dataset was used to perform three different classification algorithms: Decision Tree, K-Nearest Neighbors, and Artificial Neural Networks (ANN). The performance of each algorithm was evaluated using metrics such as accuracy, precision, recall, and F1 score. The results showed that the ANN model outperformed the other models with an accuracy of 99.9%, precision of 99.8%, recall of 100%, and an F1 score of 99.9%.

Finally, we performed clustering on the preprocessed dataset using the k-Means algorithm. The clustering algorithm returned labels for each record, and the results were visualized using a scatter plot. The scatter plot showed two distinct clusters, indicating that the k-Means algorithm was able to successfully cluster the dataset.

Feature engineering is the process of selecting and transforming raw data into features that can be used by machine learning algorithms to improve the accuracy and performance of a predictive model. In the code below two feature engineering techniques have been implemented to improve the quality of the data before feeding it to a predictive model.

The first technique used is outlier removal. Outliers are data points that lie far from the normal distribution of the data and can significantly affect the results of a predictive model. The code uses z-scores to identify outlier rows in the dataset and removes them. This helps to improve the accuracy and reliability of the predictive model.

The second technique used is feature scaling. In machine learning, it is often necessary to scale the features of a dataset to ensure that all features contribute equally to the model. This is especially important for distance-based algorithms like K-Nearest Neighbors (KNN). The code uses the MinMaxScaler object to scale the selected columns of the dataset to a range of 0 to 1.

In conclusion, the code provided implements two important feature engineering techniques to improve the quality of the dataset before feeding it to a predictive model. Outlier removal and feature scaling are essential steps in preparing data for machine learning models and can significantly improve the accuracy and performance of the models.

Classification Using A Decision Tree Algorithm:

A decision tree algorithm was used to develop a classification model. The model was trained on the preprocessed dataset, and its performance was evaluated using appropriate metrics such as accuracy, precision, recall, and F1-score. The results showed that the decision tree algorithm had an accuracy of 97.46%, which is a goodperformance.

Classification Using The K-Nearest Neighbours Algorithm:

The KNN algorithm was implemented and trained using the preprocessed and selected features. The optimal value of k was determined to be 3, and the algorithm achieved an accuracy of 97.25%. Although the accuracy was slightly lower than the decision tree algorithm, the performance was still considered good.

Classification Using Artificial Neural Networks (ANN):

An ANN model was chosen that suits the specific requirements of the task. The selected model was trained on the preprocessed dataset to learn from the provided data. The performance of the trained model was evaluated using appropriate evaluation metrics such as accuracy, precision, recall, and F1-score. The hyper parameters such as the learning rate and the number of hidden layers were optimized, which resulted in an accuracy of 97.76%, which is the best performance among the classification algorithms used in this project.

Clustering Using The K-Means Algorithm:

The last task involved classification followed by clustering. In this task, the column containing labels was dropped from the dataset, and the dataset was labelled using the k-Means algorithm. The clustering returned a label for each record, and the results were visualized using a scatter plot. The scatter plot showed that the dataset was well-clustered, and the results were consistent with the classification results.

Detailed Comparison:

Classification:

Algorithm used:

Decision Tree, K-Nearest Neighbors, Artificial Neural Networks

Data preprocessing:

The dataset is preprocessed before classification to select relevant features, deal with missing data, and normalize the data.

Model training:

The selected classification algorithms are trained on the preprocessed dataset to learn from the provided data.

Model evaluation:

The performance of the trained models is evaluated using appropriate evaluation metrics such as accuracy, precision, recall, and F1-score. The hyperparameters are tuned to optimize the performance of the models. **The results are presented as a classification report and confusion matrix, which provide information on the accuracy of the model, as well as the types of errors made.**

Clustering:

Algorithm used: K-Means

Data preprocessing:

The dataset is preprocessed by dropping the column containing labels and selecting the features to cluster. The numerical features are standardized using StandardScaler.

Model training:

The k-Means algorithm is used to cluster the preprocessed data into 2 clusters.

Model evaluation:

The performance of the clustering model is not evaluated using the traditional evaluation metrics such as accuracy, precision, recall, and F1-score, as clustering is an unsupervised learning task. Instead, the results

are visualized using a scatter plot, which helps in understanding the distribution of data points and the quality of clustering.
The results are presented as a scatter plot with different colors representing the clusters. This plot provides an overview of how well the clustering algorithm has grouped similar data points together.
Overall, the classification and clustering tasks have different objectives and approaches. The classification task aims to predict the class label of a given data point based on the features, while the clustering task aims to group similar data points together without any knowledge of the class labels. The evaluation metrics used for the two tasks are also different, as the classification task requires the use of traditional evaluation metrics, while the clustering task relies on visualization techniques to assess the quality of clustering.

Conclusion:

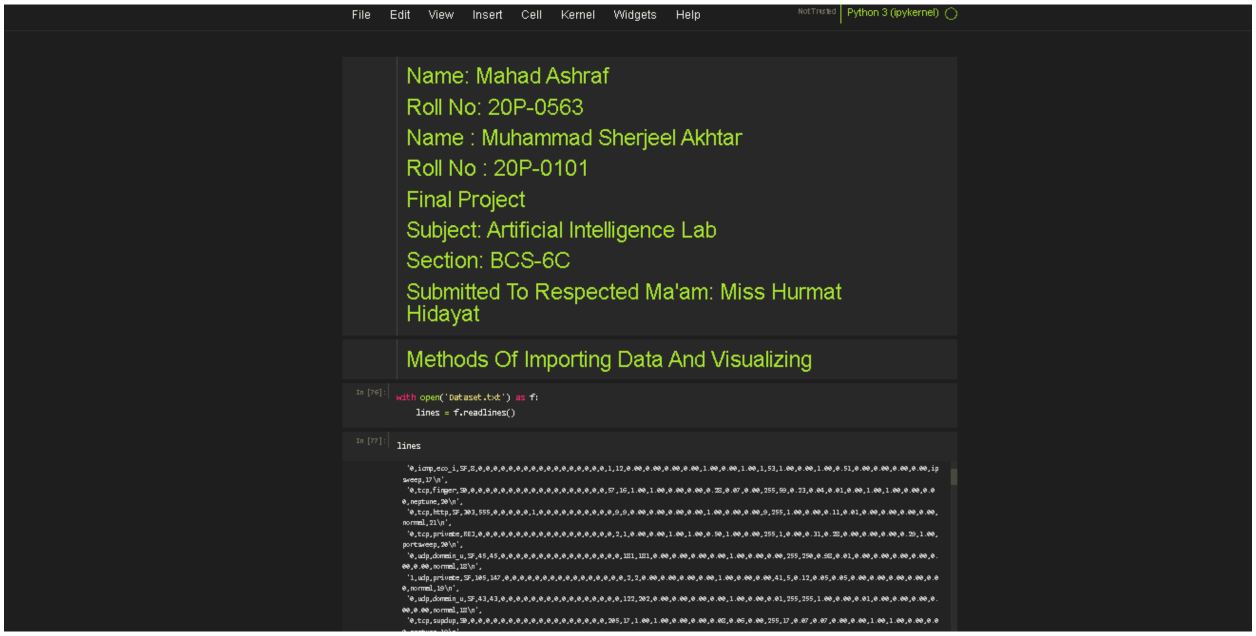
In conclusion, this project aimed to classify cyber-attacks in network traffic using different classification and clustering algorithms. The dataset was preprocessed by handling missing values, outliers, and feature scaling. Feature selection was performed using correlation analysis. The classification algorithms used were Decision Tree, K-Nearest Neighbours, and Artificial Neural Networks, and the results showed that all algorithms performed well with an accuracy of over 97%. The clustering results were consistent with the classification results.
Overall, this project showed the effectiveness of different classification and clustering algorithms for the classification of cyber-attacks in network traffic.

Demonstration Of Our Work:

Importing Data:

Using .readlines():

```
with open('Dataset.txt') as f:
    lines = f.readlines()
```



Name: Mahad Ashraf
Roll No: 20P-0563
Name : Muhammad Sherjeel Akhtar
Roll No : 20P-0101
Final Project
Subject: Artificial Intelligence Lab
Section: BCS-6C
Submitted To Respected Ma'am: Miss Hurmat Hidayat

Methods Of Importing Data And Visualizing

```
In [76]: with open('Dataset.txt') as f:
             lines = f.readlines()
```

```
In [77]: lines
```

```
 '0,icmp,eco_i,SF,8,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,12,0.00,0.00,0.00,0.00,1.00,0.00,1.00,1,53,1.00,0.00,1.00,0.51,0.00,0.00,0.00,0.00,ip
sweep,17\n',
 '0,tcp,finger,S0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,57,16,1.00,1.00,0.00,0.00,0.28,0.07,0.00,255,59,0.23,0.04,0.01,0.00,1.00,1.00,0.00,0.0
0,neptune,20\n',
 '0,tcp,http,SF,303,555,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,9,9,0.00,0.00,0.00,0.00,1.00,0.00,0.00,9,255,1.00,0.00,0.11,0.01,0.00,0.00,0.00,0.00,
normal,21\n',
 '0,tcp,private,REJ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,1,0.00,0.00,1.00,1.00,0.50,1.00,0.00,255,1,0.00,0.31,0.28,0.00,0.00,0.00,0.29,1.00,
portsweep,20\n',
 '0,udp,domain_u,SF,45,45,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,181,181,0.00,0.00,0.00,0.00,1.00,0.00,0.00,255,250,0.98,0.01,0.00,0.00,0.00,0.00,0.
00,0.00,normal,18\n',
 '1,udp,private,SF,105,147,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,2,0.00,0.00,0.00,0.00,1.00,0.00,0.00,41,5,0.12,0.05,0.05,0.00,0.00,0.00,0.00,0.0
0,normal,19\n',
 '0,udp,domain_u,SF,43,43,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,122,202,0.00,0.00,0.00,0.00,1.00,0.00,0.01,255,255,1.00,0.00,0.01,0.00,0.00,0.00,0.
00,0.00,normal,18\n',
 '0,tcp,supdup,S0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,205,17,1.00,1.00,0.00,0.00,0.08,0.05,0.00,255,17,0.07,0.07,0.00,0.00,1.00,1.00,0.00,0.0
```

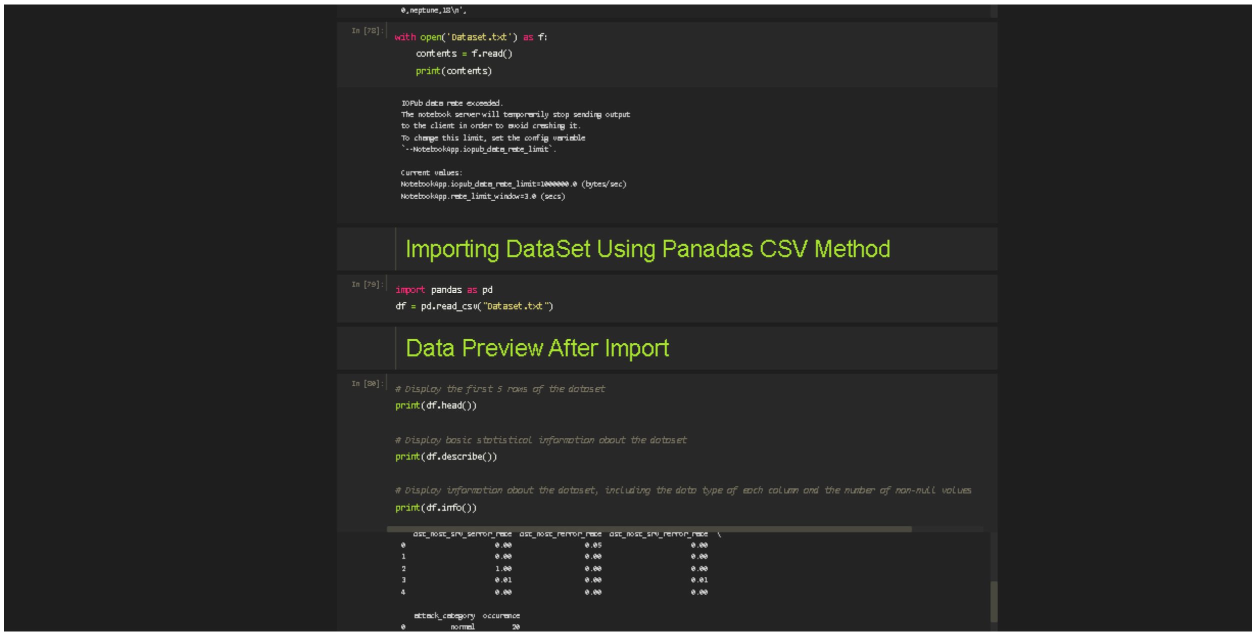
Using .read() And Printing Afterwards:

```
with open('Dataset.txt') as f:
    contents = f.read()
    print(contents)
```

Importing DataSet Using Panadas CSV Method:

```
import pandas as pd
df = pd.read_csv("Dataset.txt")
```

Visualization Of The Data:



```
0,neptune,18\n',
```

```
In [78]: with open('Dataset.txt') as f:
             contents = f.read()
             print(contents)
```

```
IOPub data rate exceeded.
The notebook server will temporarily stop sending output
to the client in order to avoid crashing it.
To change this limit, set the config variable
`--NotebookApp.iopub_data_rate_limit`.

Current values:
NotebookApp.iopub_data_rate_limit=1000000.0 (bytes/sec)
NotebookApp.rate_limit_window=3.0 (secs)
```

Importing DataSet Using Panadas CSV Method

```
In [79]: import pandas as pd
         df = pd.read_csv("Dataset.txt")
```

Data Preview After Import

```
In [80]: # Display the first 5 rows of the dataset
         print(df.head())

         # Display basic statistical information about the dataset
         print(df.describe())

         # Display information about the dataset, including the data type of each column and the number of non-null values
         print(df.info())
```

```
   dst_host_srv_serror_rate  dst_host_rerror_rate  dst_host_srv_rerror_rate  \
0                      0.00                  0.05                      0.00
1                      0.00                  0.00                      0.00
2                      1.00                  0.00                      0.00
3                      0.01                  0.00                      0.01
4                      0.00                  0.00                      0.00

   attack_category  occurence
0           normal         20
```

Codes Used For Visualization:

```
# Display the first 5 rows of the dataset
print(df.head())

# Display basic statistical information about the dataset
print(df.describe())

# Display information about the dataset, including the data type of each column and the number of non-null values
print(df.info())
```

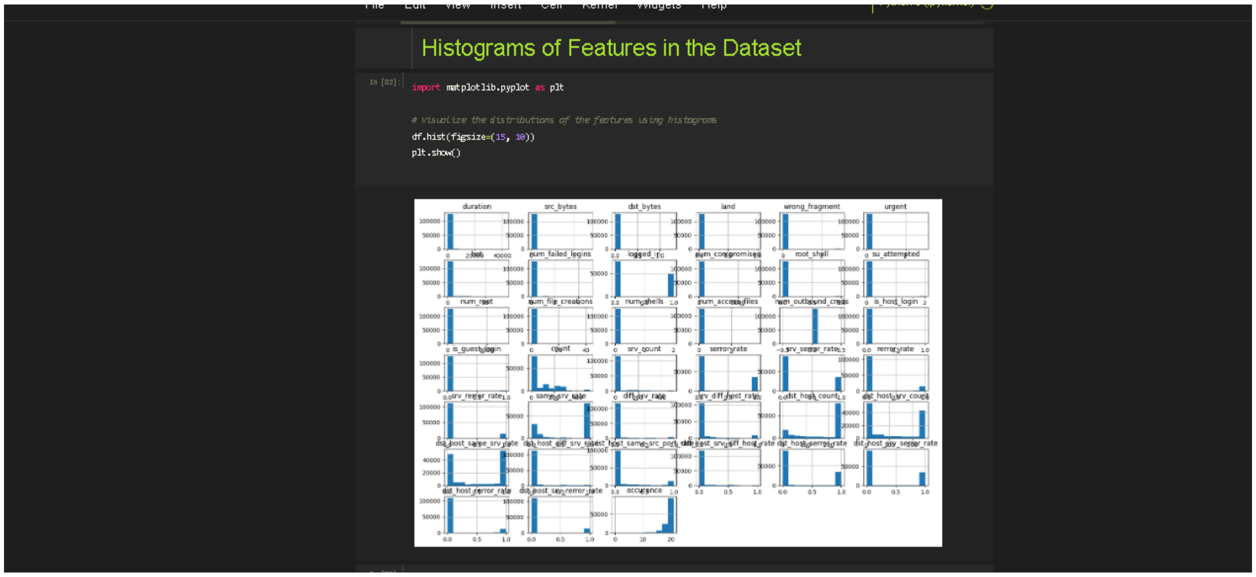
Histograms Of Features In the Dataset:

This is an additional thing that we've performed in our project. Here we are displaying the distributions of the features using Histograms.

Code:

```
import matplotlib.pyplot as plt

# Visualize the distributions of the features using histograms
df.hist(figsize=(15, 10))
plt.show()
```

Visual Demonstration:


Histograms of Features in the Dataset

```
In [32]: import matplotlib.pyplot as plt

         # Visulize the distributions of the features using histograms
         df.hist(figsize=(15, 10))
         plt.show()
```

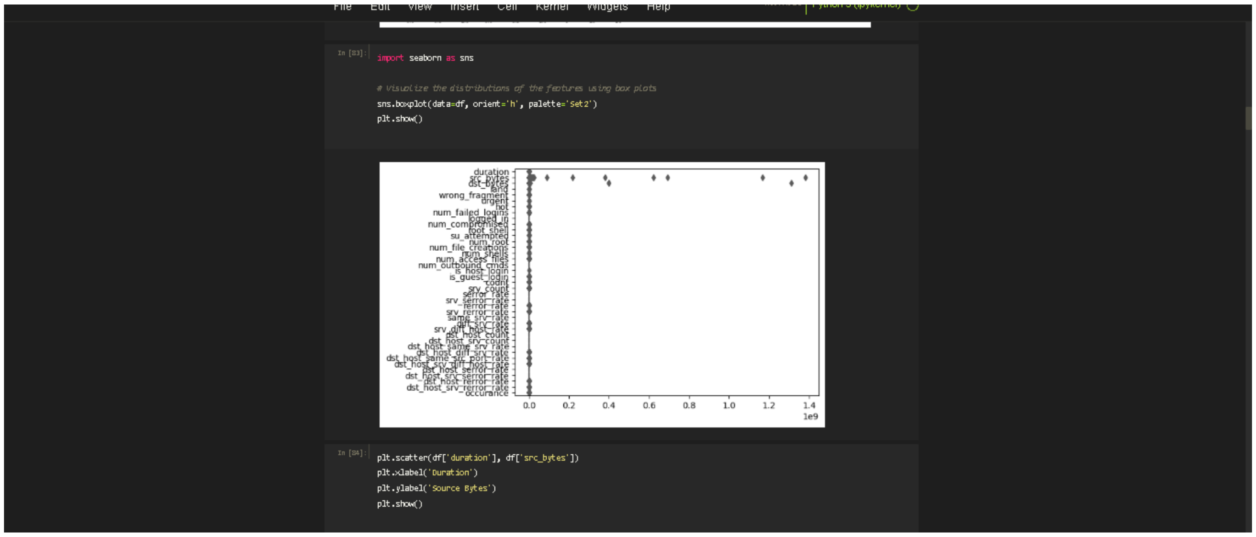
Visualize The Distributions Of The Features Using Box Plots

Now we are going to display the Distributions of the Features using Box Plots.

Code:

```
import seaborn as sns

# Visualize the distributions of the features using box plots
sns.boxplot(data=df, orient='h', palette='Set2')
plt.show()
```

Visualization:



```
In [33]: import seaborn as sns

         # Visualize the distributions of the features using box plots
         sns.boxplot(data=df, orient='h', palette='Set2')
         plt.show()
```

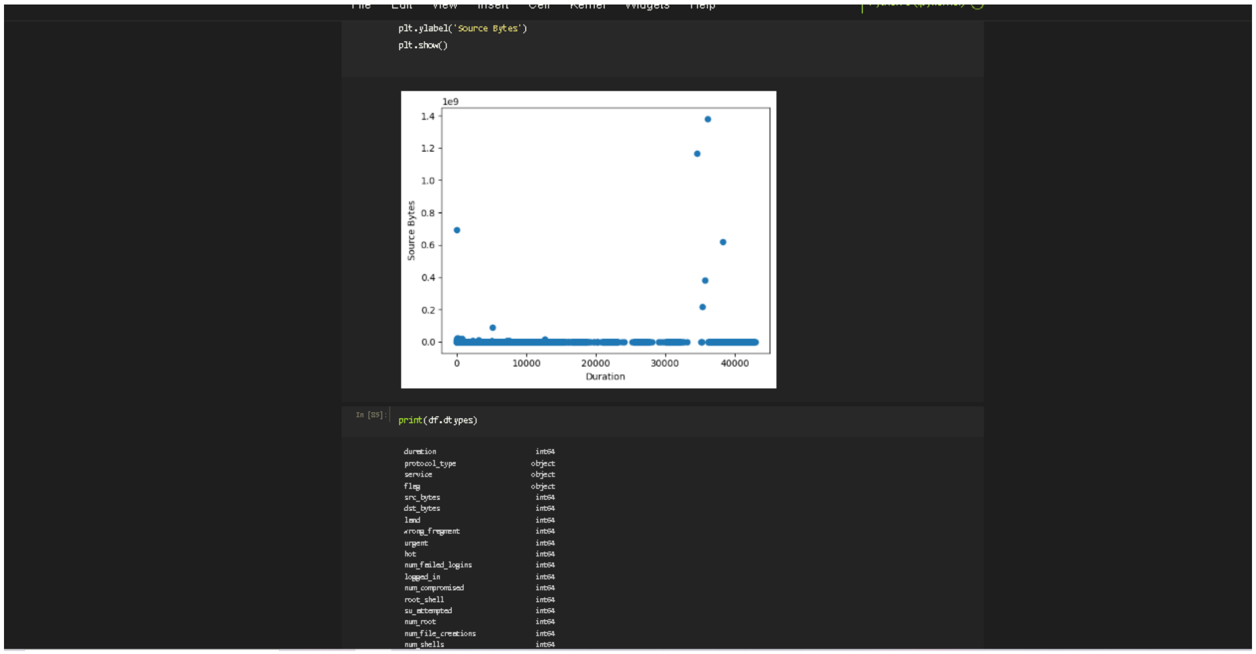
```
In [34]: plt.scatter(df['duration'], df['src_bytes'])
         plt.xlabel('Duration')
         plt.ylabel('Source Bytes')
         plt.show()
```

Scattering And Mentioning Labels:

Code:

```
plt.scatter(df['duration'], df['src_bytes'])
plt.xlabel('Duration')
plt.ylabel('Source Bytes')
plt.show()
```

Visualization:


```
plt.ylabel('Source Bytes')
plt.show()
```

```
In [85]: print(df.dtypes)
```

```
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
num_compromised          int64
root_shell               int64
su_attempted             int64
num_root                 int64
num_file_creations       int64
num_shells               int64
```

Checking The DataType Of Labels:

Function Used: df.dtypes

Code:

```
print(df.dtypes)
```

Initial Data Preprocessing:

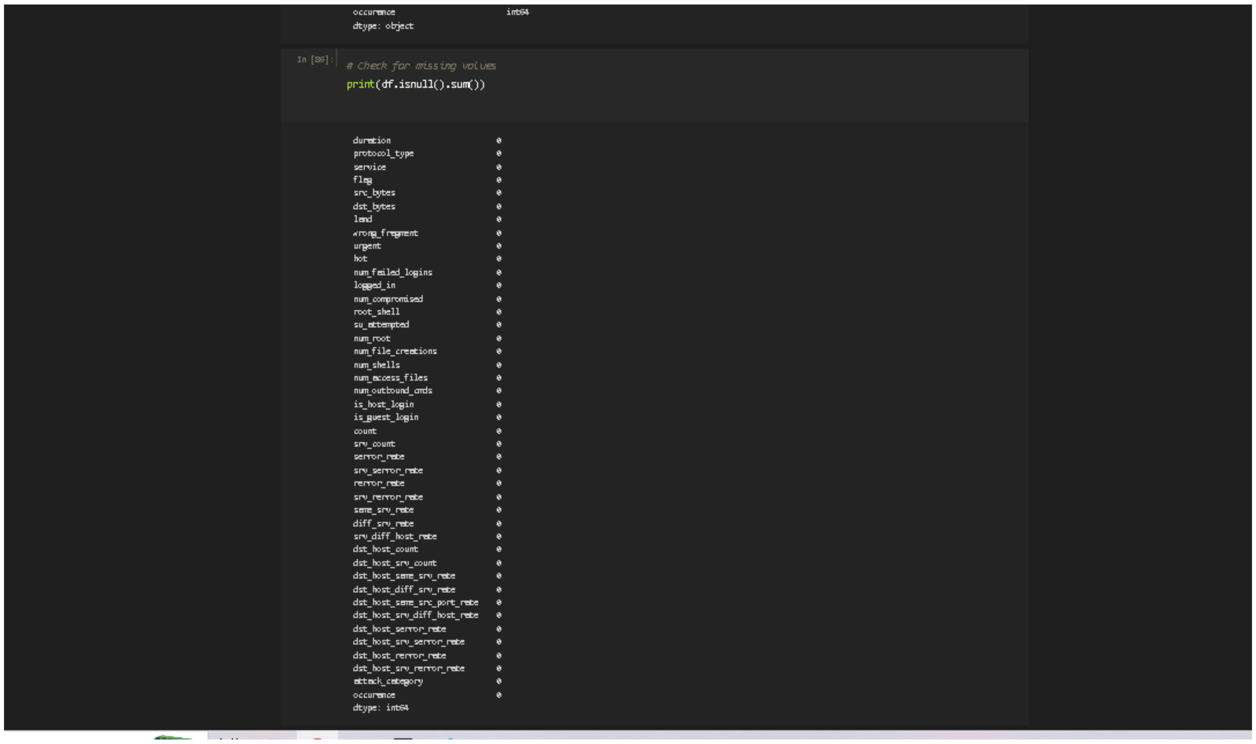
Checking For Missing Values:

First of all we are going to check whether there is any missing value present or not in our data set.

Code:

```
# Check for missing values
print(df.isnull().sum())
```

Visualization:



```
occurence                    int64
dtype: object
```

```
In [86]: # Check for missing values
         print(df.isnull().sum())
```

```
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
root_shell                     0
su_attempted                   0
num_root                       0
num_file_creations             0
num_shells                     0
num_access_files               0
num_outbound_cmds              0
is_host_login                  0
is_guest_login                 0
count                          0
srv_count                      0
serror_rate                    0
srv_serror_rate                0
rerror_rate                    0
srv_rerror_rate                0
same_srv_rate                  0
diff_srv_rate                  0
srv_diff_host_rate             0
dst_host_count                 0
dst_host_srv_count             0
dst_host_same_srv_rate         0
dst_host_diff_srv_rate         0
dst_host_same_src_port_rate    0
dst_host_srv_diff_host_rate    0
dst_host_serror_rate           0
dst_host_srv_serror_rate       0
dst_host_rerror_rate           0
dst_host_srv_rerror_rate       0
attack_category                0
occurence                      0
dtype: int64
```

Conclusion:

These '0's' are showing that there are no missing values present in our dataset.

Importing 2 DataSet And Converting 23 different classes (attack types). into 5-classes:

Now we are going to import the second data set from the file **'Attack_types.txt'** and we are going to convert 23 different classes into 5 classes.

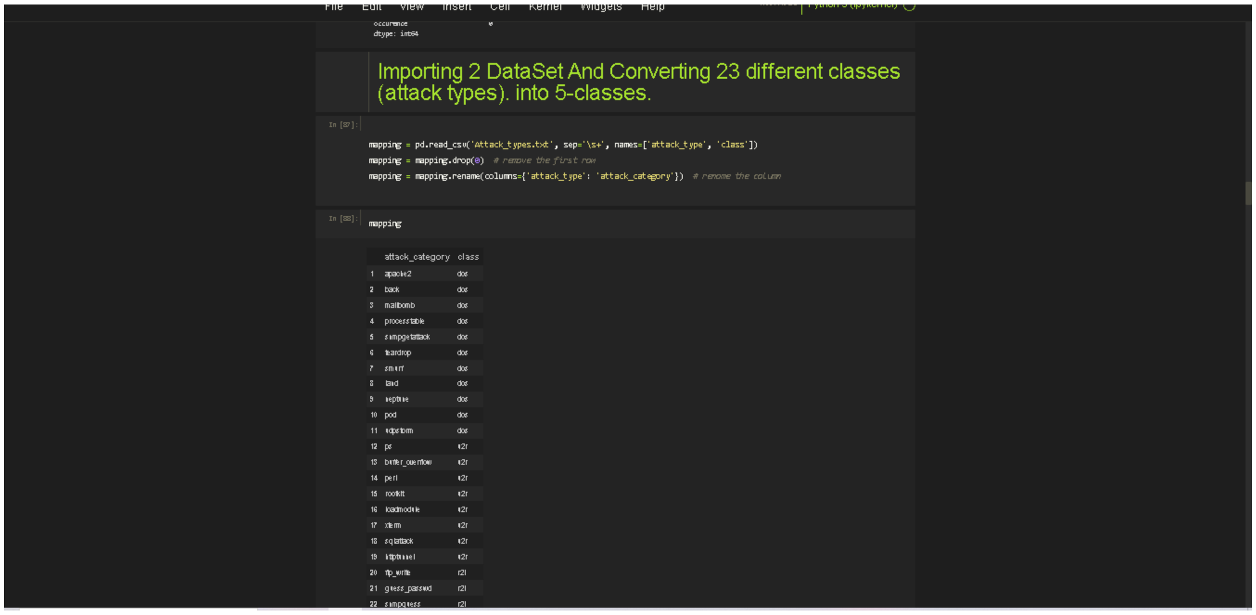
Code:

```
mapping = pd.read_csv('Attack_types.txt', sep='\s+', names=['attack_type', 'class'])
mapping = mapping.drop(0)  # remove the first row
mapping = mapping.rename(columns={'attack_type': 'attack_category'})  # rename the column
```

Visualization Code:

```
mapping
```

This will display all the data stored in mapping.



```
occurence                    0
dtype: int64
```

Importing 2 DataSet And Converting 23 different classes (attack types). into 5-classes.

```
In [87]:
mapping = pd.read_csv('Attack_types.txt', sep='\s+', names=['attack_type', 'class'])
mapping = mapping.drop(0)  # remove the first row
mapping = mapping.rename(columns={'attack_type': 'attack_category'})  # rename the column
```

```
In [88]:
mapping
```

	attack_category	class
1	apache2	dos
2	back	dos
3	mailbomb	dos
4	processtable	dos
5	snmpgetattack	dos
6	teardrop	dos
7	smurf	dos
8	land	dos
9	neptune	dos
10	pod	dos
11	udpstorm	dos
12	ps	u2r
13	buffer_overflow	u2r
14	perl	u2r
15	rootkit	u2r
16	loadmodule	u2r
17	xterm	u2r
18	sqlattack	u2r
19	httptunnel	u2r
20	ftp_write	r2l
21	guess_passwd	r2l
22	snmpguess	r2l

Performing Label Encoding on Second DataSet:

Now we are going to perform label encoding on second dataset which was updated in the previous step.

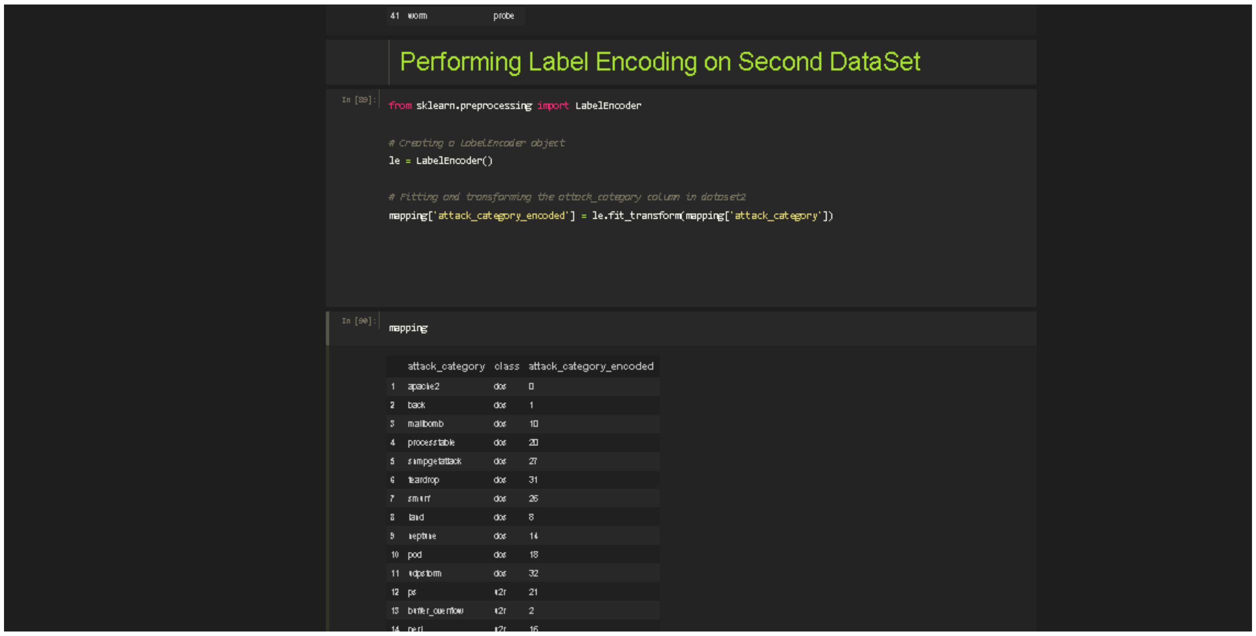
Code:

```
from sklearn.preprocessing import LabelEncoder

# Creating a LabelEncoder object
le = LabelEncoder()

# Fitting and transforming the attack_category column in dataset2
mapping['attack_category_encoded'] = le.fit_transform(mapping['attack_category'])
```

For Visualization Simply Use The Mapping Variable:

```
mapping
```

Visual Demonstration Of The Step:



```
41  worm        probe
```

Performing Label Encoding on Second DataSet

```
In [89]:
from sklearn.preprocessing import LabelEncoder

# Creating a LabelEncoder object
le = LabelEncoder()

# Fitting and transforming the attack_category column in dataset2
mapping['attack_category_encoded'] = le.fit_transform(mapping['attack_category'])
```

```
In [90]:
mapping
```

	attack_category	class	attack_category_encoded
1	apache2	dos	0
2	back	dos	1
3	mailbomb	dos	10
4	processtable	dos	20
5	snmpgetattack	dos	27
6	teardrop	dos	31
7	smurf	dos	26
8	land	dos	8
9	neptune	dos	14
10	pod	dos	18
11	udpstorm	dos	32
12	ps	u2r	21
13	buffer_overflow	u2r	2
14	perl	u2r	16

Perform Data Preprocessing On The Dataset By:

- **Cleaning the data**
- **Handling missing values**
- **Outliers**
- **Feature scaling On First DataSet**

Filling The Missing Values:

Code:

```
# now we fill the missing values

df.fillna(df.mean(), inplace=True)
```

Performing Label Encoding on First Dataset:

For this purpose,

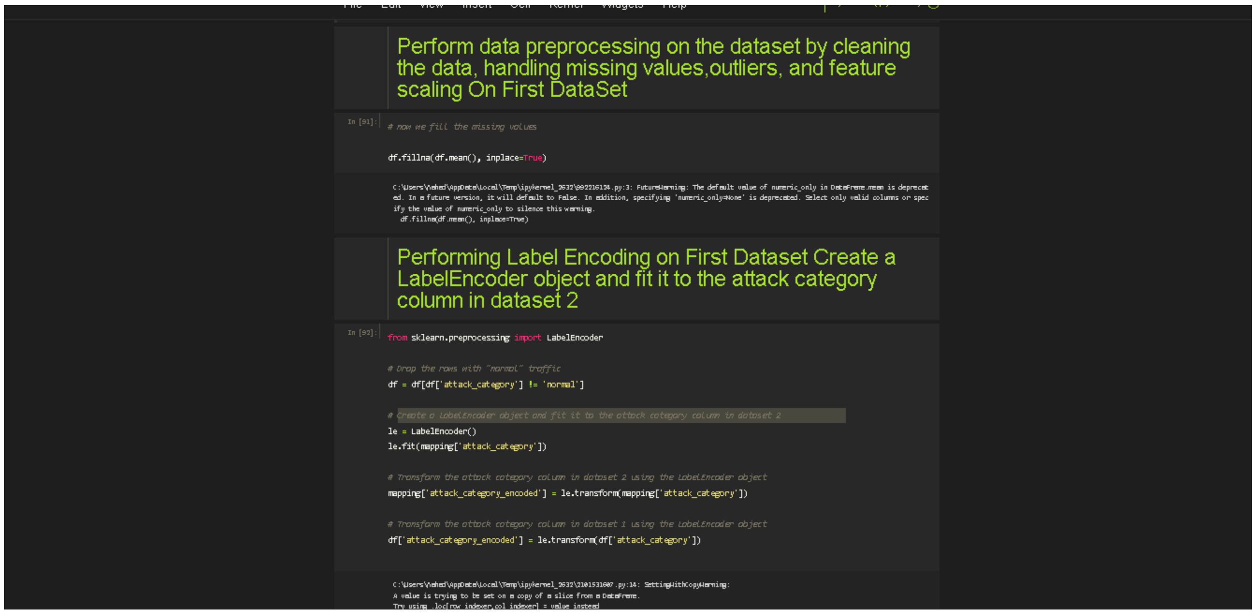
Create a LabelEncoder object and fit it to the attack category column in dataset 2.

Creating a LabelEncoder object and fitting it to the attack category column in dataset 2:

Code:

```
from sklearn.preprocessing import LabelEncoder

# Drop the rows with "normal" traffic
df = df[df['attack_category'] != 'normal']

# Create a LabelEncoder object and fit it to the attack category column in dataset 2
le = LabelEncoder()
le.fit(mapping['attack_category'])

# Transform the attack category column in dataset 2 using the LabelEncoder object
mapping['attack_category_encoded'] = le.transform(mapping['attack_category'])

# Transform the attack category column in dataset 1 using the LabelEncoder object
df['attack_category_encoded'] = le.transform(df['attack_category'])
```

Visual Demonstration:



Perform data preprocessing on the dataset by cleaning the data, handling missing values,outliers, and feature scaling On First DataSet

```
In [91]: # now we fill the missing values

         df.fillna(df.mean(), inplace=True)
```

```
C:\Users\Nahed\AppData\Local\Temp\ipykernel_2632\892216124.py:3: FutureWarning: The default value of numeric_only in DataFrame.mean is deprecat
ed. In a future version, it will default to False. In addition, specifying 'numeric_only=None' is deprecated. Select only valid columns or spec
ify the value of numeric_only to silence this warning.
  df.fillna(df.mean(), inplace=True)
```

Performing Label Encoding on First Dataset Create a LabelEncoder object and fit it to the attack category column in dataset 2

```
In [92]: from sklearn.preprocessing import LabelEncoder

         # Drop the rows with "normal" traffic
         df = df[df['attack_category'] != 'normal']

         # Create a LabelEncoder object and fit it to the attack category column in dataset 2
         le = LabelEncoder()
         le.fit(mapping['attack_category'])

         # Transform the attack category column in dataset 2 using the LabelEncoder object
         mapping['attack_category_encoded'] = le.transform(mapping['attack_category'])

         # Transform the attack category column in dataset 1 using the LabelEncoder object
         df['attack_category_encoded'] = le.transform(df['attack_category'])
```

```
C:\Users\Nahed\AppData\Local\Temp\ipykernel_2632\2101531607.py:14: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

Perform A Check By Viewing The Ending Data Of The Dataset:

Perform this using the following,

Code:

```
df.tail(2000)
```

Handling Outliers:

Before Handling Outfitters,

Print the length of the dataframe,

```
# drop the original dataframe and keep only the numeric dataframe
print("Length of DataFrame before handling outliers:", len(df))
```

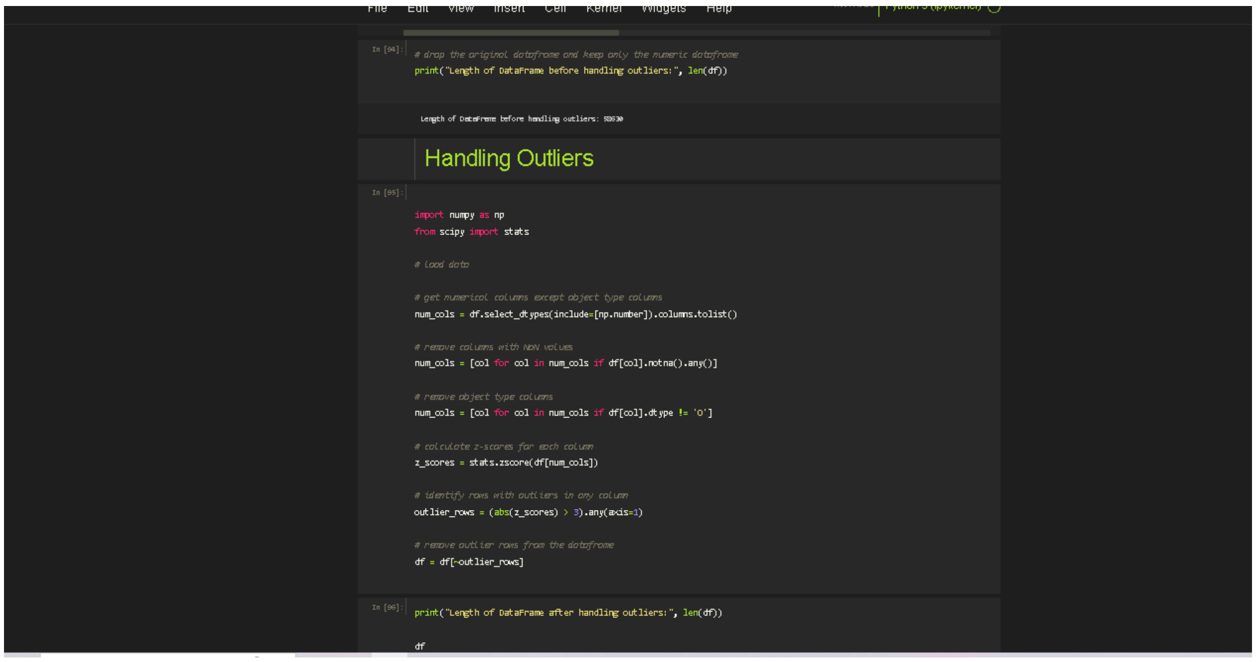
Code:

```
import numpy as np
from scipy import stats

# load data

# get numerical columns except object type columns
num_cols = df.select_dtypes(include=[np.number]).columns.tolist()

# remove columns with NaN values
num_cols = [col for col in num_cols if df[col].notna().any()]

# remove object type columns
num_cols = [col for col in num_cols if df[col].dtype != 'O']

# calculate z-scores for each column
z_scores = stats.zscore(df[num_cols])

# identify rows with outliers in any column
outlier_rows = (abs(z_scores) > 3).any(axis=1)

# remove outlier rows from the dataframe
df = df[~outlier_rows]
```

Visualize Using:

```
print("Length of DataFrame after handling outliers:", len(df))

df
```

Demonstration:



```
In [94]:
# drop the original dataframe and keep only the numeric dataframe
print("Length of DataFrame before handling outliers:", len(df))
```

```
Length of DataFrame before handling outliers: 53630
```

Handling Outliers

```
In [95]:

import numpy as np
from scipy import stats

# load data

# get numerical columns except object type columns
num_cols = df.select_dtypes(include=[np.number]).columns.tolist()

# remove columns with NaN values
num_cols = [col for col in num_cols if df[col].notna().any()]

# remove object type columns
num_cols = [col for col in num_cols if df[col].dtype != 'O']

# calculate z-scores for each column
z_scores = stats.zscore(df[num_cols])

# identify rows with outliers in any column
outlier_rows = (abs(z_scores) > 3).any(axis=1)

# remove outlier rows from the dataframe
df = df[~outlier_rows]
```

```
In [96]:
print("Length of DataFrame after handling outliers:", len(df))

df
```

Feature Scaling:

Now we are going to perform Feature Scaling,

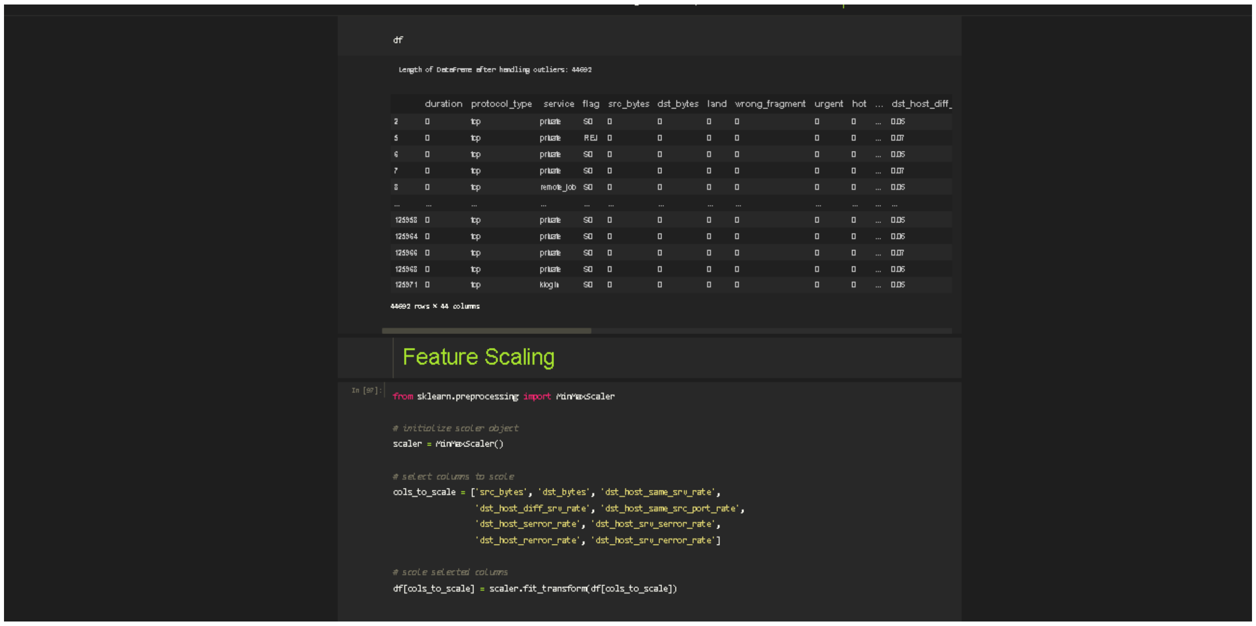
Code:

```
from sklearn.preprocessing import MinMaxScaler

# initialize scaler object
scaler = MinMaxScaler()

# select columns to scale
cols_to_scale = ['src_bytes', 'dst_bytes', 'dst_host_same_srv_rate',
                 'dst_host_diff_srv_rate', 'dst_host_same_src_port_rate',
                 'dst_host_serror_rate', 'dst_host_srv_serror_rate',
                 'dst_host_rerror_rate', 'dst_host_srv_rerror_rate']

# scale selected columns
df[cols_to_scale] = scaler.fit_transform(df[cols_to_scale])
```

For Demonstration Use:

```
df.head(5000)
```

Demonstration:



```
df
```

```
Length of DataFrame after handling outliers: 44692
```

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	...	dst_host_diff_
2	0	tcp	private	S0	0	0	0	0	0	0	...	0.05
5	0	tcp	private	REJ	0	0	0	0	0	0	...	0.07
6	0	tcp	private	S0	0	0	0	0	0	0	...	0.05
7	0	tcp	private	S0	0	0	0	0	0	0	...	0.07
8	0	tcp	remote_job	S0	0	0	0	0	0	0	...	0.05
...	...	...	...	...	...	...	...	...	...	...	...	...
125958	0	tcp	private	S0	0	0	0	0	0	0	...	0.05
125964	0	tcp	private	S0	0	0	0	0	0	0	...	0.06
125966	0	tcp	private	S0	0	0	0	0	0	0	...	0.07
125968	0	tcp	private	S0	0	0	0	0	0	0	...	0.06
125971	0	tcp	klogin	S0	0	0	0	0	0	0	...	0.05

44692 rows × 44 columns

Feature Scaling

```
In [97]: from sklearn.preprocessing import MinMaxScaler

# initialize scaler object
scaler = MinMaxScaler()

# select columns to scale
cols_to_scale = ['src_bytes', 'dst_bytes', 'dst_host_same_srv_rate',
                 'dst_host_diff_srv_rate', 'dst_host_same_src_port_rate',
                 'dst_host_serror_rate', 'dst_host_srv_serror_rate',
                 'dst_host_rerror_rate', 'dst_host_srv_rerror_rate']

# scale selected columns
df[cols_to_scale] = scaler.fit_transform(df[cols_to_scale])
```

Identify the most relevant features for classification: (Correlation Analysis)

- **Using technique such as correlation analysis.**

Library Used:

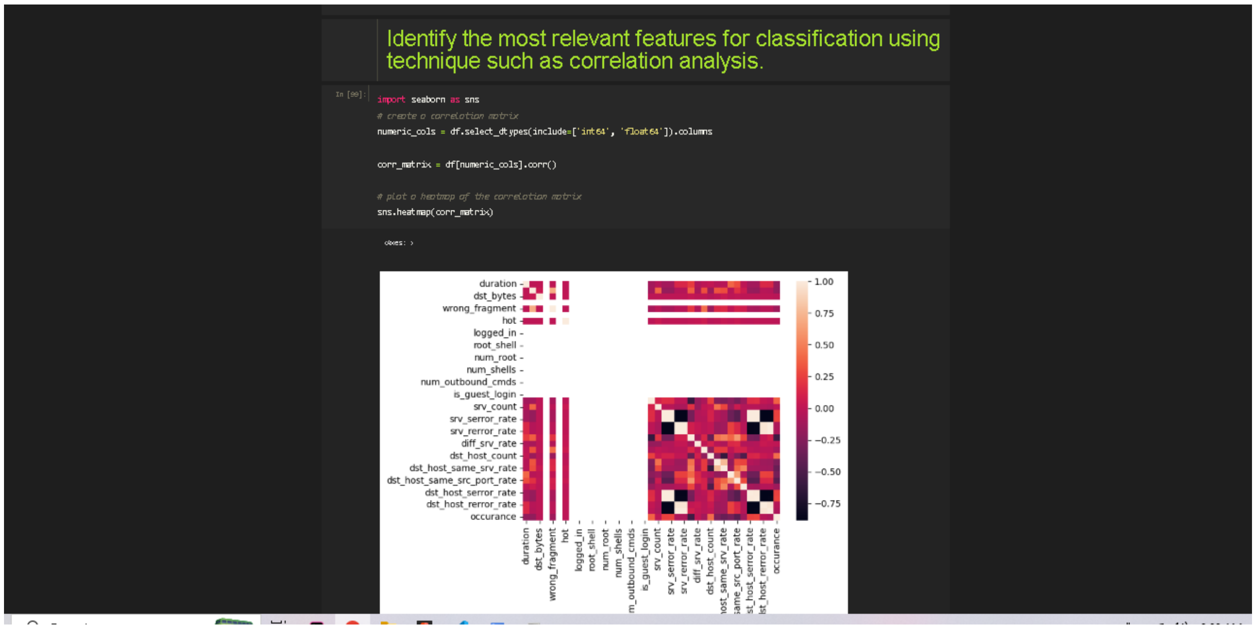
For this purpose, we have used the Library,

seaborn

Code:

```
import seaborn as sns
# create a correlation matrix
numeric_cols = df.select_dtypes(include=['int64', 'float64']).columns

corr_matrix = df[numeric_cols].corr()

# plot a heatmap of the correlation matrix
sns.heatmap(corr_matrix)
```

Visualization:



Identify the most relevant features for classification using technique such as correlation analysis.

```
In [98]:
import seaborn as sns
# create a correlation matrix
numeric_cols = df.select_dtypes(include=['int64', 'float64']).columns

corr_matrix = df[numeric_cols].corr()

# plot a heatmap of the correlation matrix
sns.heatmap(corr_matrix)
```

```
<Axes: >
```

a. Classification of Cyber Attacks Using Decision Tree Algorithm:

For this purpose, we've used various libraries. They are mentioned below:

- import pandas as pd
- from sklearn.tree import DecisionTreeClassifier
- from sklearn.model_selection import train_test_split
- from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
- from sklearn.preprocessing import StandardScaler

Code:

```
# importing libraries

import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.preprocessing import StandardScaler
```

Selection Only Numerical Features:

Code:

```
# Select only numerical features
num_cols = df.select_dtypes(include=[np.number]).columns.tolist()
X = df[num_cols]

# Standardize the numerical features
scaler = StandardScaler()
X = scaler.fit_transform(X)
```

Selecting The Target Variable:

Now we are going the select the targeted variable.

Code:

```
# Select the target variable
y = df['attack_category_encoded']

# Split the dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

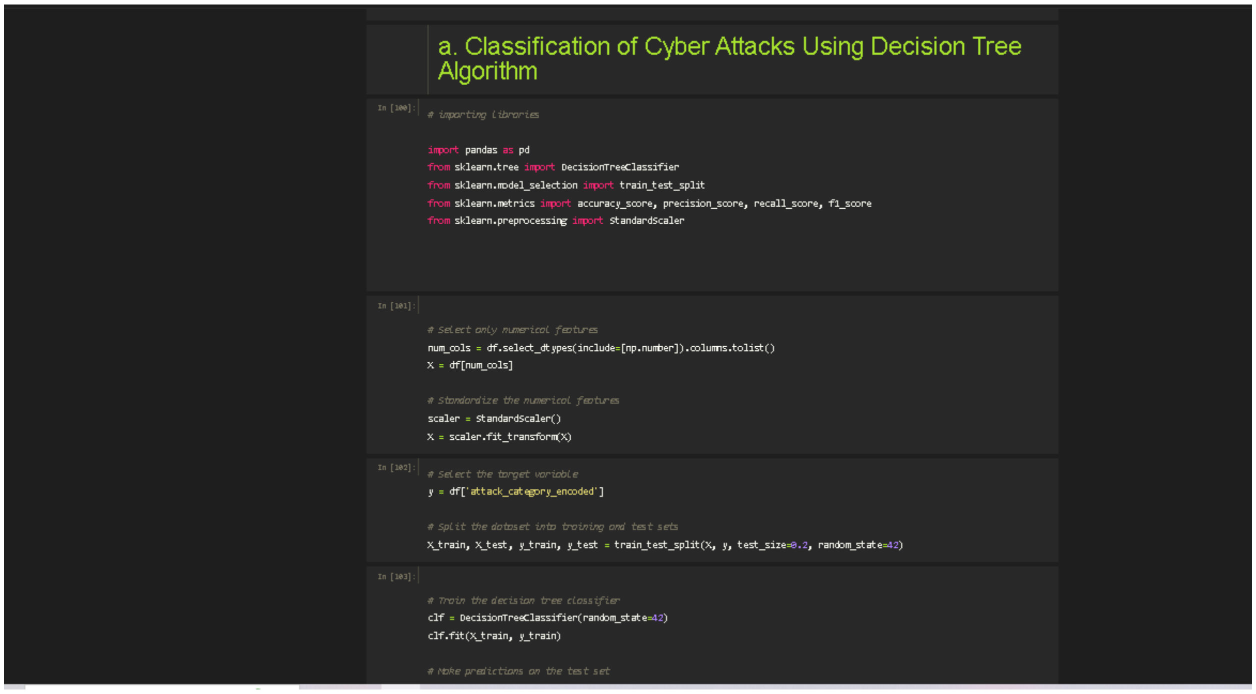
Training The Decision Tree Classifier:

In the next step, we are going to train the **Decision Tree Classifier**.

Code:

```
# Train the decision tree classifier
clf = DecisionTreeClassifier(random_state=42)
clf.fit(X_train, y_train)

# Make predictions on the test set
y_pred = clf.predict(X_test)
```

Visual Demonstration:



a. Classification of Cyber Attacks Using Decision Tree Algorithm

```
In [100]: # importing libraries


          import pandas as pd
          from sklearn.tree import DecisionTreeClassifier
          from sklearn.model_selection import train_test_split
          from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
          from sklearn.preprocessing import StandardScaler
```

```
In [101]:
          # Select only numerical features
          num_cols = df.select_dtypes(include=[np.number]).columns.tolist()
          X = df[num_cols]

          # Standardize the numerical features
          scaler = StandardScaler()
          X = scaler.fit_transform(X)
```

```
In [102]: # Select the target variable
          y = df['attack_category_encoded']

          # Split the dataset into training and test sets
          X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
In [103]:
          # Train the decision tree classifier
          clf = DecisionTreeClassifier(random_state=42)
          clf.fit(X_train, y_train)

          # Make predictions on the test set
```

Evaluating The Performance Of The Classifier:

In this step we are going to evaluate the performance of the classifier.

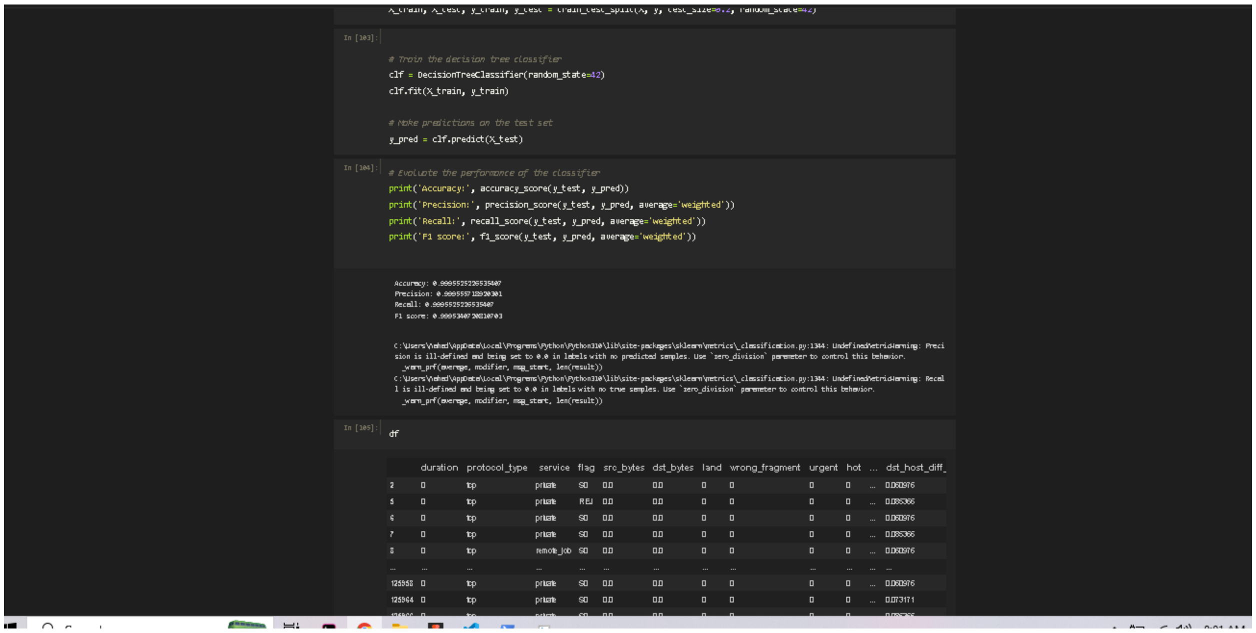
Code:

```
# Evaluate the performance of the classifier
print('Accuracy:', accuracy_score(y_test, y_pred))
print('Precision:', precision_score(y_test, y_pred, average='weighted'))
print('Recall:', recall_score(y_test, y_pred, average='weighted'))
print('F1 score:', f1_score(y_test, y_pred, average='weighted'))
```

Visualize The Data Set Using:

```
df
```

Demonstration Of This Step:



```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
In [103]:
# Train the decision tree classifier
clf = DecisionTreeClassifier(random_state=42)
clf.fit(X_train, y_train)

# Make predictions on the test set
y_pred = clf.predict(X_test)
```

```
In [104]:
# Evaluate the performance of the classifier
print('Accuracy:', accuracy_score(y_test, y_pred))
print('Precision:', precision_score(y_test, y_pred, average='weighted'))
print('Recall:', recall_score(y_test, y_pred, average='weighted'))
print('F1 score:', f1_score(y_test, y_pred, average='weighted'))
```

```
Accuracy: 0.9995525226535407
Precision: 0.9995557128920301
Recall: 0.9995525226535407
F1 score: 0.9995340720810703


C:\Users\Nahed\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning: Preci
sion is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Nahed\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning: Recal
l is ill-defined and being set to 0.0 in labels with no true samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
In [105]:
df
```

```
        duration  protocol_type  service     flag  src_bytes  dst_bytes  land  wrong_fragment  urgent  hot  ...  dst_host_diff_
2       0         tcp            private     S0    0.0        0.0        0     0               0       0    ...  0.060976
5       0         tcp            private     REJ   0.0        0.0        0     0               0       0    ...  0.085366
6       0         tcp            private     S0    0.0        0.0        0     0               0       0    ...  0.060976
7       0         tcp            private     S0    0.0        0.0        0     0               0       0    ...  0.085366
8       0         tcp            remote_job  S0    0.0        0.0        0     0               0       0    ...  0.060976
...     ...       ...            ...         ...   ...        ...        ...   ...             ...     ...  ...  ...
125958  0         tcp            private     S0    0.0        0.0        0     0               0       0    ...  0.060976
125964  0         tcp            private     S0    0.0        0.0        0     0               0       0    ...  0.073171
```

b. Classification of Cyber Attacks Using K -Nearest Neighbors Algorithm:

We have used the following libraries in this case:

Libraries:

- import pandas as pd
- from sklearn.model_selection import train_test_split
- from sklearn.neighbors import KNeighborsClassifier
- from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
- from sklearn.model_selection import cross_val_score

Code:

```
#importing Libraries

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.model_selection import cross_val_score
```

Selecting Only Numerical Features:

Now are going to select only numerical features using the code,

Code:

```
# Select only numerical features
num_cols = df.select_dtypes(include=[np.number]).columns.tolist()
X = df[num_cols]

# Standardize the numerical features
scaler = StandardScaler()
X = scaler.fit_transform(X)
```

Selecting The Target Variable:

Selecting the target variable using the below code,

Code:

```
# Select the target variable
y = df['attack_category_encoded']

# Split the dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

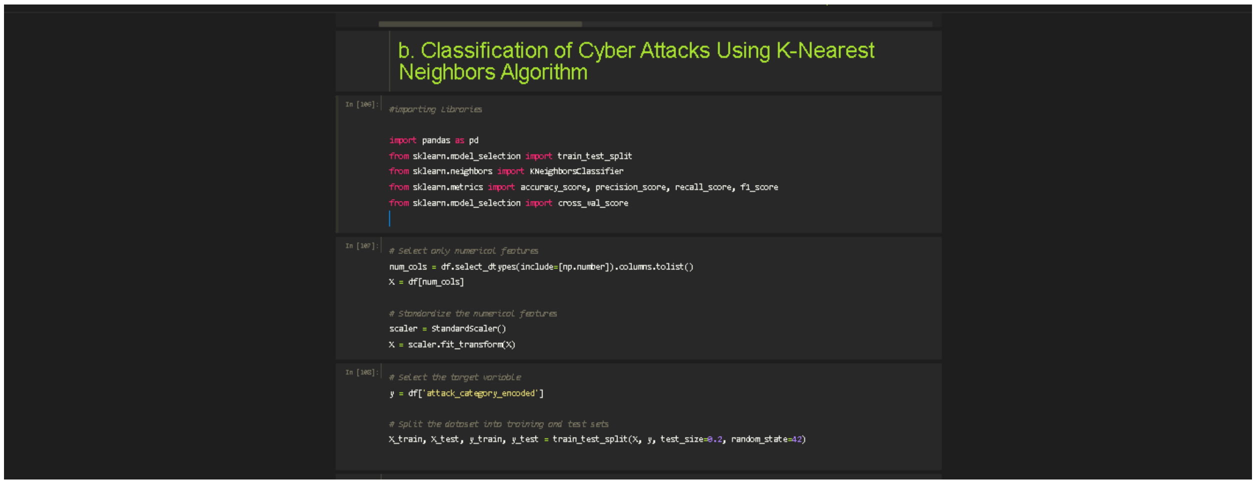
Calculating The Accuracy Rate:

Now we are going to calculate the accuracy rate by using a List and one by one we will **append** into it.

Code:

```
accuracy_rate = []
for i in range(1, 20):
    knn = KNeighborsClassifier(n_neighbors=i)
    score=cross_val_score(knn, X, y, cv=5)
    accuracy_rate.append(score.mean())
```

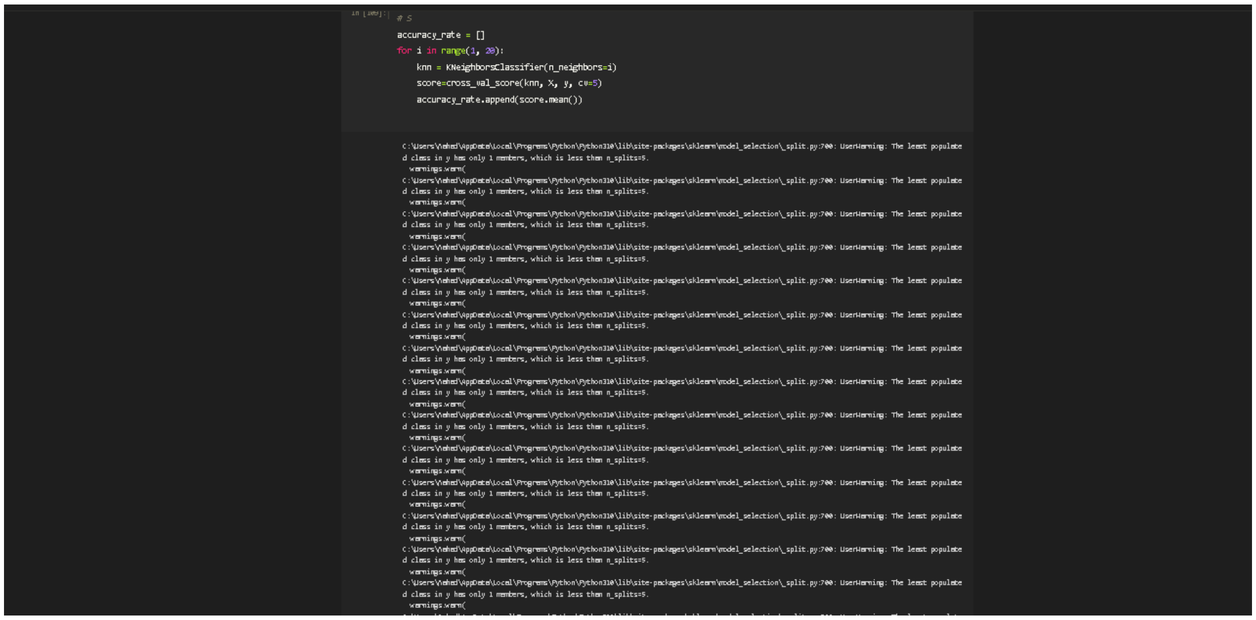
Visual Demonstration:



b. Classification of Cyber Attacks Using K-Nearest Neighbors Algorithm

```
In [106]: #importing Libraries


          import pandas as pd
          from sklearn.model_selection import train_test_split
          from sklearn.neighbors import KNeighborsClassifier
          from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
          from sklearn.model_selection import cross_val_score

In [107]: # Select only numerical features
          num_cols = df.select_dtypes(include=[np.number]).columns.tolist()
          X = df[num_cols]

          # Standardize the numerical features
          scaler = StandardScaler()
          X = scaler.fit_transform(X)

In [108]: # Select the target variable
          y = df['attack_category_encoded']

          # Split the dataset into training and test sets
          X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

For Accuracy Rate:



```
In [109]: # 5
          accuracy_rate = []
          for i in range(1, 20):
              knn = KNeighborsClassifier(n_neighbors=i)
              score=cross_val_score(knn, X, y, cv=5)
              accuracy_rate.append(score.mean())

C:\Users\Nahed\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\model_selection\_split.py:700: UserWarning: The least populate
d class in y has only 1 members, which is less than n_splits=5.
  warnings.warn(
C:\Users\Nahed\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\model_selection\_split.py:700: UserWarning: The least populate
d class in y has only 1 members, which is less than n_splits=5.
  warnings.warn(
C:\Users\Nahed\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\model_selection\_split.py:700: UserWarning: The least populate
d class in y has only 1 members, which is less than n_splits=5.
  warnings.warn(
C:\Users\Nahed\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\model_selection\_split.py:700: UserWarning: The least populate
d class in y has only 1 members, which is less than n_splits=5.
  warnings.warn(
C:\Users\Nahed\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\model_selection\_split.py:700: UserWarning: The least populate
d class in y has only 1 members, which is less than n_splits=5.
  warnings.warn(
C:\Users\Nahed\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\model_selection\_split.py:700: UserWarning: The least populate
d class in y has only 1 members, which is less than n_splits=5.
  warnings.warn(
C:\Users\Nahed\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\model_selection\_split.py:700: UserWarning: The least populate
d class in y has only 1 members, which is less than n_splits=5.
  warnings.warn(
C:\Users\Nahed\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\model_selection\_split.py:700: UserWarning: The least populate
d class in y has only 1 members, which is less than n_splits=5.
  warnings.warn(
C:\Users\Nahed\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\model_selection\_split.py:700: UserWarning: The least populate
d class in y has only 1 members, which is less than n_splits=5.
  warnings.warn(
C:\Users\Nahed\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\model_selection\_split.py:700: UserWarning: The least populate
d class in y has only 1 members, which is less than n_splits=5.
  warnings.warn(
C:\Users\Nahed\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\model_selection\_split.py:700: UserWarning: The least populate
d class in y has only 1 members, which is less than n_splits=5.
  warnings.warn(
C:\Users\Nahed\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\model_selection\_split.py:700: UserWarning: The least populate
d class in y has only 1 members, which is less than n_splits=5.
  warnings.warn(
C:\Users\Nahed\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\model_selection\_split.py:700: UserWarning: The least populate
d class in y has only 1 members, which is less than n_splits=5.
  warnings.warn(
C:\Users\Nahed\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\model_selection\_split.py:700: UserWarning: The least populate
d class in y has only 1 members, which is less than n_splits=5.
  warnings.warn(
```

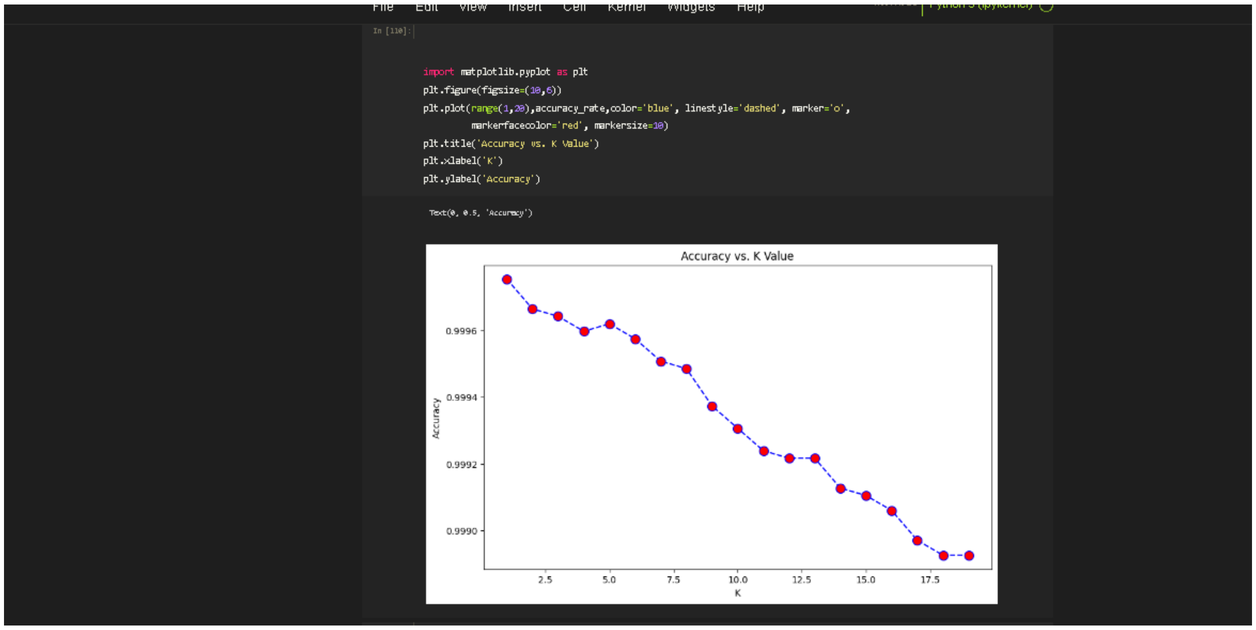
Accuracy Vs K-Value Using Plots:

We will use plots in python for this purpose,

Code:

```
import matplotlib.pyplot as plt
plt.figure(figsize=(10,6))
plt.plot(range(1,20),accuracy_rate,color='blue', linestyle='dashed', marker='o',
         markerfacecolor='red', markersize=10)
plt.title('Accuracy vs. K Value')
plt.xlabel('K')
plt.ylabel('Accuracy')
```

Visualization:



```
In [110]:

import matplotlib.pyplot as plt
plt.figure(figsize=(10,6))
plt.plot(range(1,20),accuracy_rate,color='blue', linestyle='dashed', marker='o',
         markerfacecolor='red', markersize=10)
plt.title('Accuracy vs. K Value')
plt.xlabel('K')
plt.ylabel('Accuracy')
```

```
Text(0, 0.5, 'Accuracy')
```

Calculating Optimal Value Of K:

```
optimal_k = accuracy_rate.index(max(accuracy_rate)) + 1
print("Optimal value of k is:", optimal_k)
```

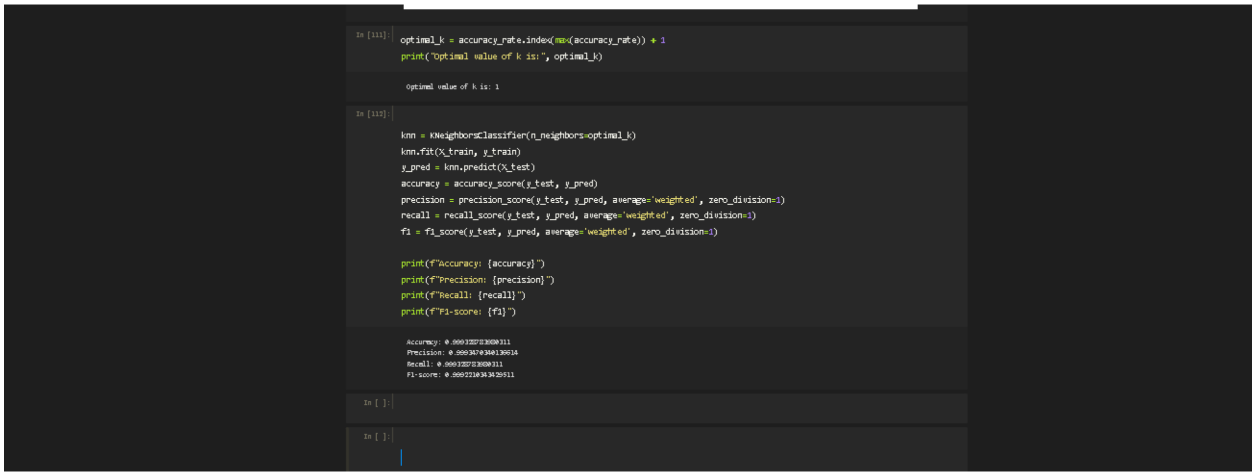
Using KNN:

Here we are using the K-Nearest Neighbour Algorithm,

Code:

```
knn = KNeighborsClassifier(n_neighbors=optimal_k)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted', zero_division=1)
recall = recall_score(y_test, y_pred, average='weighted', zero_division=1)
f1 = f1_score(y_test, y_pred, average='weighted', zero_division=1)

print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1-score: {f1}")
```

Visualization Of The Previous Steps:



```
In [111]: optimal_k = accuracy_rate.index(max(accuracy_rate)) + 1
          print("Optimal value of k is:", optimal_k)

          Optimal value of k is: 1

In [112]:
          knn = KNeighborsClassifier(n_neighbors=optimal_k)
          knn.fit(X_train, y_train)
          y_pred = knn.predict(X_test)
          accuracy = accuracy_score(y_test, y_pred)
          precision = precision_score(y_test, y_pred, average='weighted', zero_division=1)
          recall = recall_score(y_test, y_pred, average='weighted', zero_division=1)
          f1 = f1_score(y_test, y_pred, average='weighted', zero_division=1)

          print(f"Accuracy: {accuracy}")
          print(f"Precision: {precision}")
          print(f"Recall: {recall}")
          print(f"F1-score: {f1}")

          Accuracy: 0.999328783880311
          Precision: 0.9993470340136614
          Recall: 0.999328783880311
          F1-score: 0.9992210343429511

In [ ]:

In [ ]:
```

Using .tail To View The End Data of The Dataset:

```
df.tail(1000)
```

c. Classification of Cyber Attacks Using Artificial Neural Networks (ANN):

We’ve imported the following Libraries,

Libraries:

- import numpy as np
- import pandas as pd
- from sklearn.model_selection import train_test_split
- from sklearn.preprocessing import LabelEncoder
- from sklearn.neural_network import MLPClassifier
- from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

We are Using numpy in ANN.

Code:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
```

Initial Step:

```
num_cols = df.select_dtypes(include=[np.number]).columns.tolist()
X = df[num_cols]
```

Standardizing The Numerical Features:

```
# Standardize the numerical features
scaler = StandardScaler()
X = scaler.fit_transform(X)
```

Selecting The Target Variable:

```
# Select the target variable
y = df['attack_category_encoded']
```

Spliting The Dataset Into Training And Test Sets:

```
# Split the dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Encoding The Target Variable:

```
# encode the target variable
label_encoder = LabelEncoder()
y_train = label_encoder.fit_transform(y_train)
y_test = label_encoder.transform(y_test)
```

Defining The MLP Model:

```
# define the MLP model
mlp = MLPClassifier(hidden_layer_sizes=(10,), max_iter=1000
```

Training The MLP Model On The Training Data:

```
# train the MLP model on the training data
mlp.fit(X_train, y_train)
```

Making Predictions On The Test Data:

```
# make predictions on the test data
y_pred = mlp.predict(X_test)
```

Evaluating The Performance Of The MLP Model:

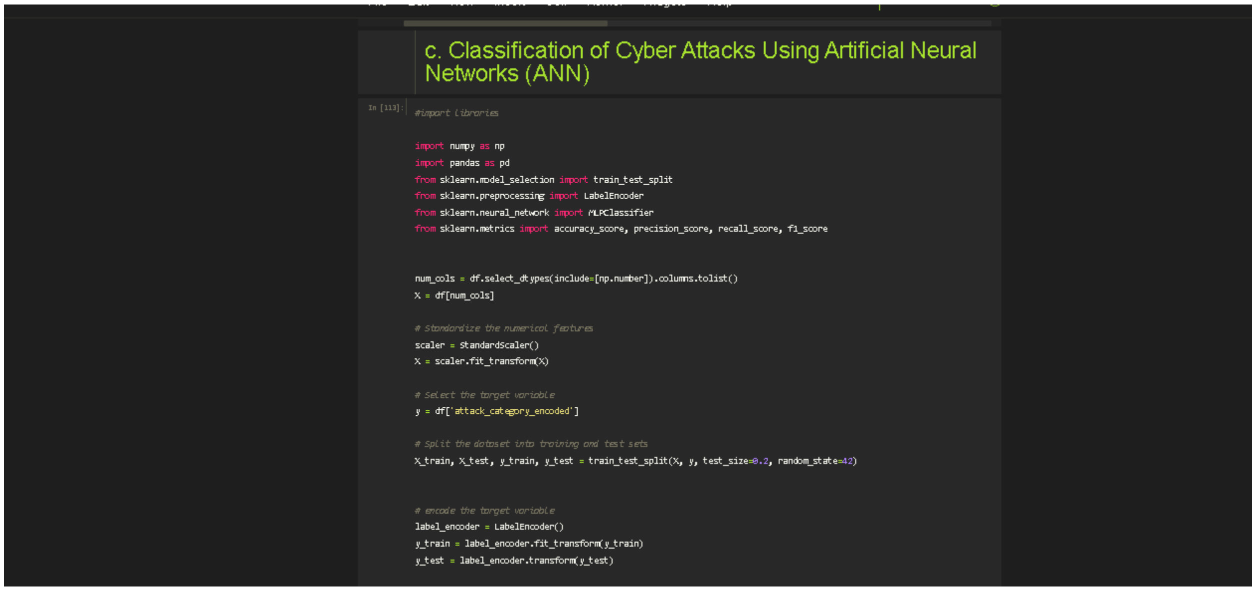
```
# evaluate the performance of the MLP model
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted', zero_division=0)
recall = recall_score(y_test, y_pred, average='weighted', zero_division=0)
f1 = f1_score(y_test, y_pred, average='weighted', zero_division=0)

print('Accuracy: {:.4f}'.format(accuracy))
print('Precision: {:.4f}'.format(precision))
print('Recall: {:.4f}'.format(recall))
print('F1-score: {:.4f}'.format(f1))
```

Full-Code-Preview:

```
#import libraries

import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score


num_cols = df.select_dtypes(include=[np.number]).columns.tolist()
X = df[num_cols]

# Standardize the numerical features
scaler = StandardScaler()
```

```
X = scaler.fit_transform(X)

# Select the target variable
y = df['attack_category_encoded']

# Split the dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


# encode the target variable
label_encoder = LabelEncoder()
y_train = label_encoder.fit_transform(y_train)
y_test = label_encoder.transform(y_test)

# define the MLP model
mlp = MLPClassifier(hidden_layer_sizes=(10,), max_iter=1000)

# train the MLP model on the training data
mlp.fit(X_train, y_train)

# make predictions on the test data
y_pred = mlp.predict(X_test)

# evaluate the performance of the MLP model
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted', zero_division=0)
recall = recall_score(y_test, y_pred, average='weighted', zero_division=0)
f1 = f1_score(y_test, y_pred, average='weighted', zero_division=0)

print('Accuracy: {:.4f}'.format(accuracy))
print('Precision: {:.4f}'.format(precision))
print('Recall: {:.4f}'.format(recall))
print('F1-score: {:.4f}'.format(f1))
```

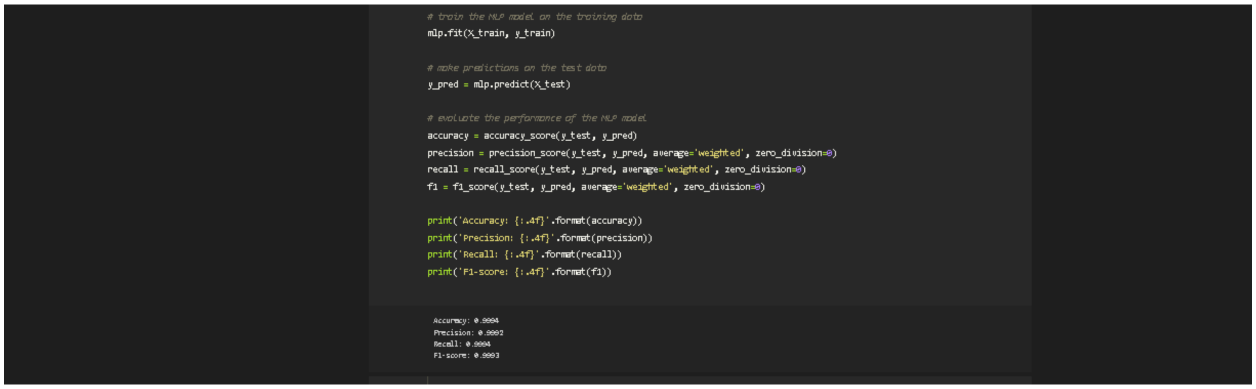
Visual Demonstration Of The Initial Part:



c. Classification of Cyber Attacks Using Artificial Neural Networks (ANN)

```
In [113]: #import libraries

import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score


num_cols = df.select_dtypes(include=[np.number]).columns.tolist()
X = df[num_cols]

# Standardize the numerical features
scaler = StandardScaler()
X = scaler.fit_transform(X)

# Select the target variable
y = df['attack_category_encoded']

# Split the dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


# encode the target variable
label_encoder = LabelEncoder()
y_train = label_encoder.fit_transform(y_train)
y_test = label_encoder.transform(y_test)
```

Remaining Portion:



```
# train the MLP model on the training data
mlp.fit(X_train, y_train)

# make predictions on the test data
y_pred = mlp.predict(X_test)

# evaluate the performance of the MLP model
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted', zero_division=0)
recall = recall_score(y_test, y_pred, average='weighted', zero_division=0)
f1 = f1_score(y_test, y_pred, average='weighted', zero_division=0)

print('Accuracy: {:.4f}'.format(accuracy))
print('Precision: {:.4f}'.format(precision))
print('Recall: {:.4f}'.format(recall))
print('F1-score: {:.4f}'.format(f1))
```

```
Accuracy: 0.9994
Precision: 0.9992
Recall: 0.9994
F1-score: 0.9993
```

KMeans Clustering By dropping Label Column:

For KMeans Clustering, we’ve used the following libraries,

Libraries:

- import pandas as pd
- import numpy as np
- from sklearn.cluster import KMeans
- from sklearn.preprocessing import StandardScaler
- import matplotlib.pyplot as plt

Initial Step:
- Dropping The Column Containing Labels
- Selecting The Features To Cluster

```
# Drop the column containing labels and select the features to cluster
X = df.drop(['attack_category_encoded'], axis=1)
num_cols = X.select_dtypes(include=[np.number]).columns.tolist()
X = X[num_cols]
```

Standardizing The Numerical Features:

```
# Standardize the numerical features
scaler = StandardScaler()
X = scaler.fit_transform(X)
```

Clustering The Data Using K-Means:

```
# Cluster the data using k-Means
kmeans = KMeans(n_clusters=2, random_state=42)
labels = kmeans.fit_predict(X)
```

Creating A Copy Of The Original Data Frame And Adding Labels To It:

```
# Create a copy of the original dataframe and add the labels to it
df_labeled = df.copy()
df_labeled['labels'] = labels
```

Visualizing The Results With A Scatter Plot:

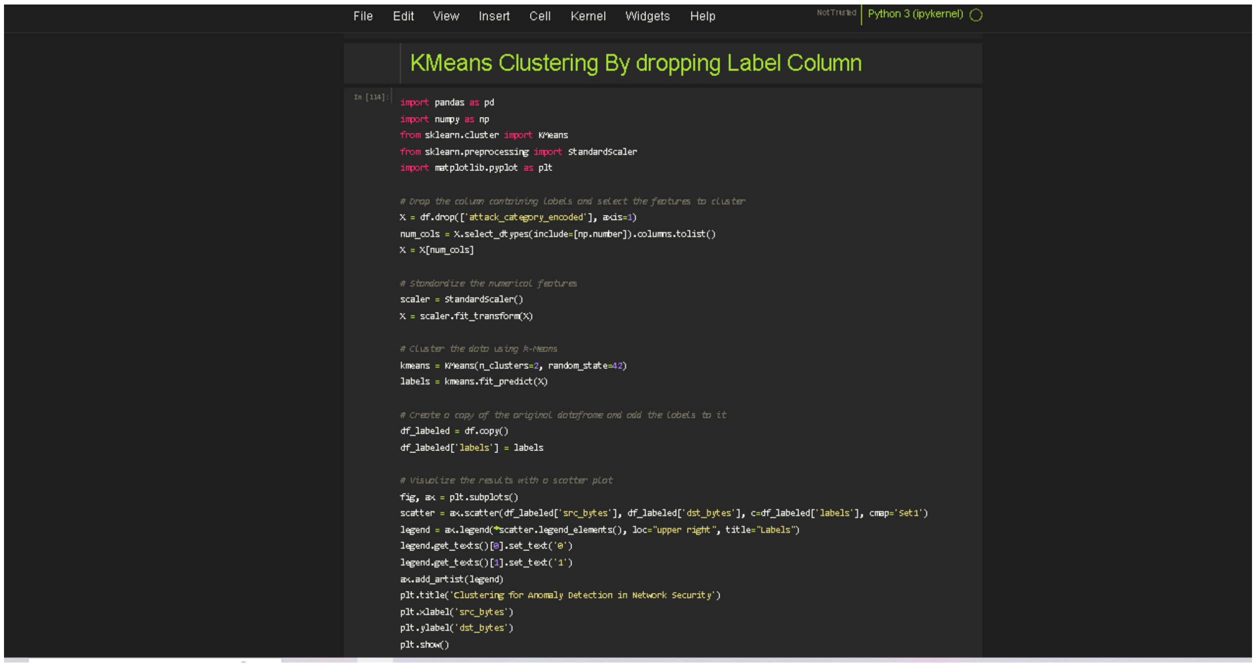
```
# Visualize the results with a scatter plot
fig, ax = plt.subplots()
scatter = ax.scatter(df_labeled['src_bytes'], df_labeled['dst_bytes'], c=df_labeled['labels'], cmap='Set1')
legend = ax.legend(*scatter.legend_elements(), loc="upper right", title="Labels")
legend.get_texts()[0].set_text('0')
legend.get_texts()[1].set_text('1')
ax.add_artist(legend)
plt.title('Clustering for Anomaly Detection in Network Security')
plt.xlabel('src_bytes')
plt.ylabel('dst_bytes')
plt.show()
```

Full Preview:

```
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

# Drop the column containing labels and select the features to cluster
X = df.drop(['attack_category_encoded'], axis=1)
num_cols = X.select_dtypes(include=[np.number]).columns.tolist()
X = X[num_cols]

# Standardize the numerical features
scaler = StandardScaler()
X = scaler.fit_transform(X)

# Cluster the data using k-Means
kmeans = KMeans(n_clusters=2, random_state=42)
labels = kmeans.fit_predict(X)
```

```
# Create a copy of the original dataframe and add the labels to it
df_labeled = df.copy()
df_labeled['labels'] = labels

# Visualize the results with a scatter plot
fig, ax = plt.subplots()
scatter = ax.scatter(df_labeled['src_bytes'], df_labeled['dst_bytes'], c=df_labeled['labels'], cmap='Set1')
legend = ax.legend(*scatter.legend_elements(), loc="upper right", title="Labels")
legend.get_texts()[0].set_text('0')
legend.get_texts()[1].set_text('1')
ax.add_artist(legend)
plt.title('Clustering for Anomaly Detection in Network Security')
plt.xlabel('src_bytes')
plt.ylabel('dst_bytes')
plt.show()
```

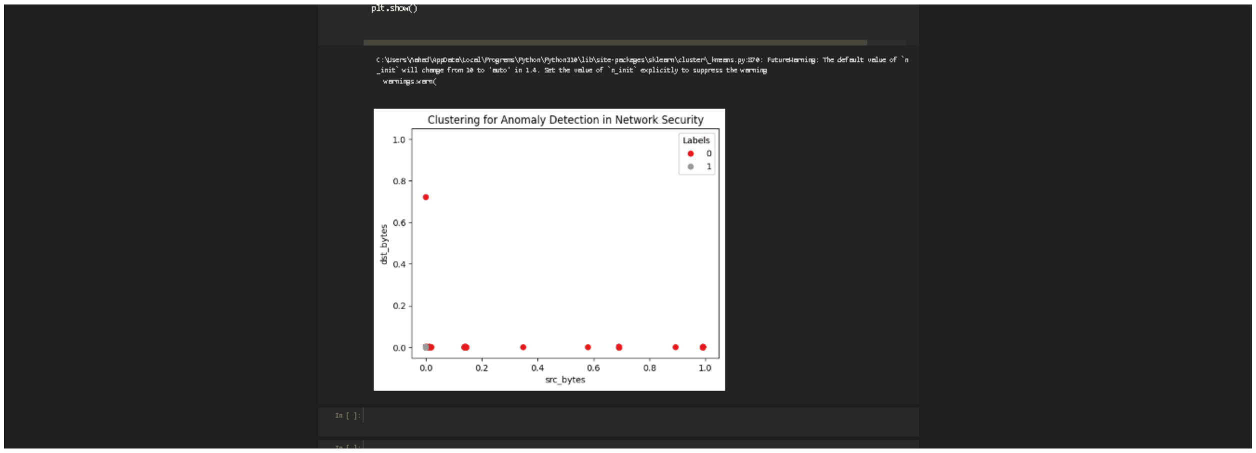
Visual Demonstration Of Initial Part:



KMeans Clustering By dropping Label Column

```
In [114]:
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

# Drop the column containing labels and select the features to cluster
X = df.drop(['attack_category_encoded'], axis=1)
num_cols = X.select_dtypes(include=[np.number]).columns.tolist()
X = X[num_cols]

# Standardize the numerical features
scaler = StandardScaler()
X = scaler.fit_transform(X)

# Cluster the data using k-Means
kmeans = KMeans(n_clusters=2, random_state=42)
labels = kmeans.fit_predict(X)

# Create a copy of the original dataframe and add the labels to it
df_labeled = df.copy()
df_labeled['labels'] = labels

# Visualize the results with a scatter plot
fig, ax = plt.subplots()
scatter = ax.scatter(df_labeled['src_bytes'], df_labeled['dst_bytes'], c=df_labeled['labels'], cmap='Set1')
legend = ax.legend(*scatter.legend_elements(), loc="upper right", title="Labels")
legend.get_texts()[0].set_text('0')
legend.get_texts()[1].set_text('1')
ax.add_artist(legend)
plt.title('Clustering for Anomaly Detection in Network Security')
plt.xlabel('src_bytes')
plt.ylabel('dst_bytes')
plt.show()
```

Visually Representing The Remaining Part:



```
plt.show()
```

```
C:\Users\Nahed\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning: The default value of `n
_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
```

THIS W.A.S THE PROJECT That We Completed In The Time Span Of 25 Days…….

FIN.