

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:

The first step in our project was to explore the dataset to understand the characteristics and distributions of different features. We found that the dataset contained missing values and outliers, which were handled by filling the missing values with the mean and removing the outliers. The dataset was then converted to five classes by mapping the original 23 classes to the new five classes. Feature selection was performed using correlation analysis, which identified the most relevant features for classification.

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 good performance.

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 hyperparameters 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.

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.

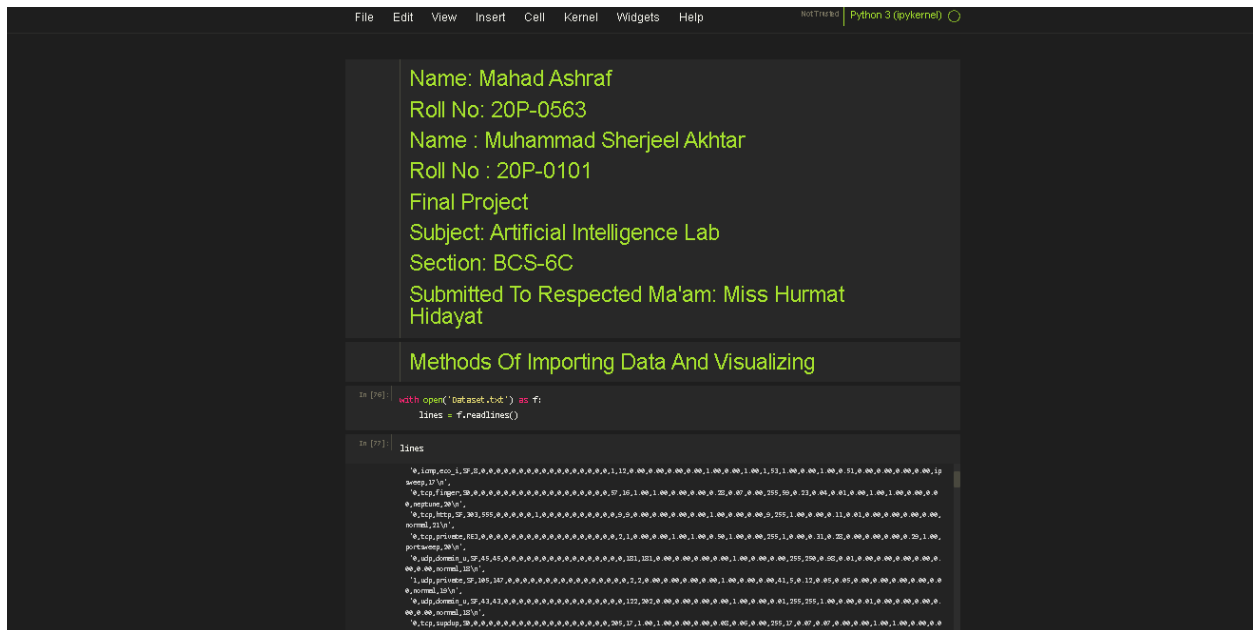
Overall, this project showed the effectiveness of different classification and clustering algorithms for the classification of cyber-attacks in network traffic.

Demonstration Of Our Work:

Importing Data:

Using `.readlines()`:

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



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,1576'.

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

With 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.input_data_rate_limit`.

Current values:
NotebookApp.input_data_rate_limit=10000000 (bytes/sec)
NotebookApp.input_data_rate_limit_window=10 (secs)

Importing DataSet Using Pandas 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())

   dist_hist_bin_center_freq  dist_hist_center_freq  dist_hist_bin_center_freq  \
0                0.00                0.00                0.00                0.00
1                0.00                0.00                0.00                0.00
2                1.00                0.00                0.00                0.00
3                0.01                0.00                0.00                0.01
4                0.00                0.00                0.00                0.00

retail_category    occurrence
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:

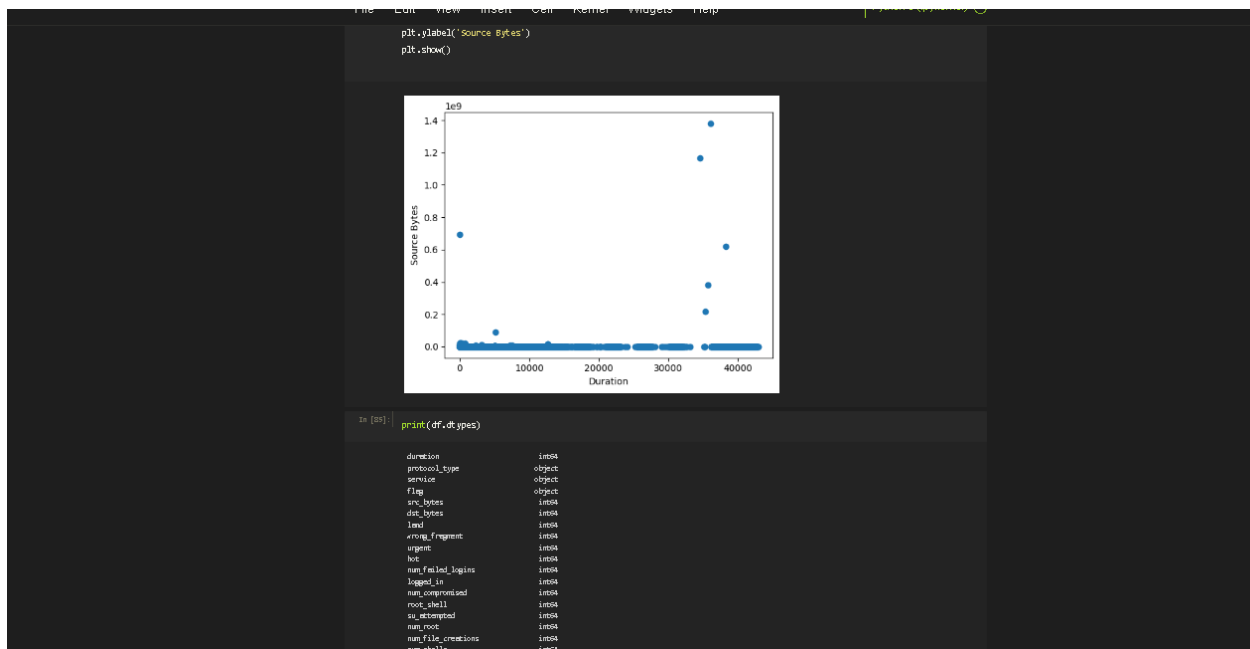
This is an additional thing that we've performed in our project. Here we are displaying the distributions of the features using Histograms.

Code:

```
import matplotlib.pyplot as plt

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

Visual Demonstration:



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:


```

occurrence      imsi4
dtype: object

In [10]: # Check for missing values
print(df.isnull().sum())

duration      0
protocol_type 0
service       0
flag          0
src_bytes     0
dst_bytes     0
lmd           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
server_rate    0
srv_server_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_server_rate 0
dst_host_srv_server_rate 0
dst_host_rerror_rate 0
dst_host_srv_rerror_rate 0
attack_category 0
occurrence     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.

```
File Edit View Insert Cell Kernel Widgets Help
In [20]:
mapping = pd.read_csv('Attack_types.txt', sep='\t+', names=['attack_type', 'class'])
mapping = mapping.drop(0) # remove the first row
mapping = mapping.rename(columns={'attack_type': 'attack_category'}) # rename the column

In [21]: mapping
  attack_category  class
1  apache2       dos
2  back         dos
3  mailbomb      dos
4  processable  dos
5  ransomware    dos
6  warship       dos
7  smurf        dos
8  land         dos
9  kephair       dos
10 pod          dos
11 kqskim       dos
12 pe           dos
13 backdoor      dos
14 perl         dos
15 rootkit      dos
16 backdoor     dos
17 x86         dos
18 sqlmap       dos
19 sqlmap       dos
20 sqlmap       dos
21 sqlmap       dos
22 sqlmap       dos
```

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 [90]: 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 | space2 | dos | 0 |
| 2 | back | dos | 1 |
| 3 | mailbomb | dos | 10 |
| 4 | processflood | dos | 20 |
| 5 | smurfattack | dos | 27 |
| 6 | landrop | dos | 31 |
| 7 | smurf | dos | 35 |
| 8 | land | dos | 8 |
| 9 | seppuku | dos | 14 |
| 10 | pod | dos | 18 |
| 11 | udpstorm | dos | 32 |
| 12 | ps | u2r | 21 |
| 13 | dataflooding | u2r | 2 |
| 14 | port | 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

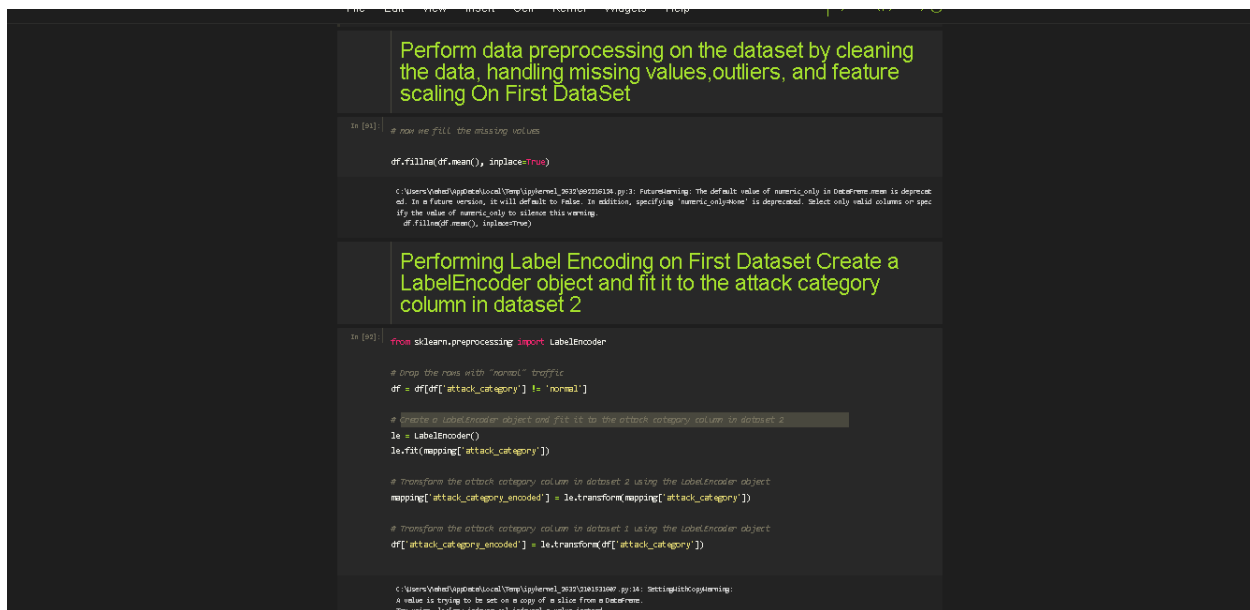
# Drop the rows with "normal" traffic
df = df[df['attack_category'] != 'normal']

# 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:



The screenshot shows a Jupyter Notebook interface with two code cells. The first cell contains a title 'Perform data preprocessing on the dataset by cleaning the data, handling missing values, outliers, and feature scaling On First DataSet' and a code block that uses `df.fillna(df.mean(), inplace=True)`. The second cell contains a title 'Performing Label Encoding on First Dataset Create a LabelEncoder object and fit it to the attack category column in dataset 2' and a code block that imports `LabelEncoder` from `sklearn.preprocessing`, drops rows with 'normal' traffic, fits a `LabelEncoder` to the 'attack_category' column of dataset 2, and then transforms the 'attack_category' column of both dataset 1 and dataset 2. The notebook interface includes a menu bar at the top with options like 'File', 'Edit', 'View', 'Insert', 'Cell', 'Format', 'Help', and a toolbar with icons for running, saving, and other actions.

```
In [91]: # now we fill the missing values
df.fillna(df.mean(), inplace=True)

C:\Users\Yahya\AppData\Local\Temp\ipynb1_3031261331087.py:3: FutureWarning: The default value of numeric_only in DataFrame.mean is deprecated. In a future version, it will default to False. In addition, specifying 'numeric_only=False' is deprecated. Select only valid columns or specify the value of numeric_only to silence this warning.
  df.fillna(df.mean(), inplace=True)

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\Yahya\AppData\Local\Temp\ipynb1_3031261331087.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 Outliers,

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)

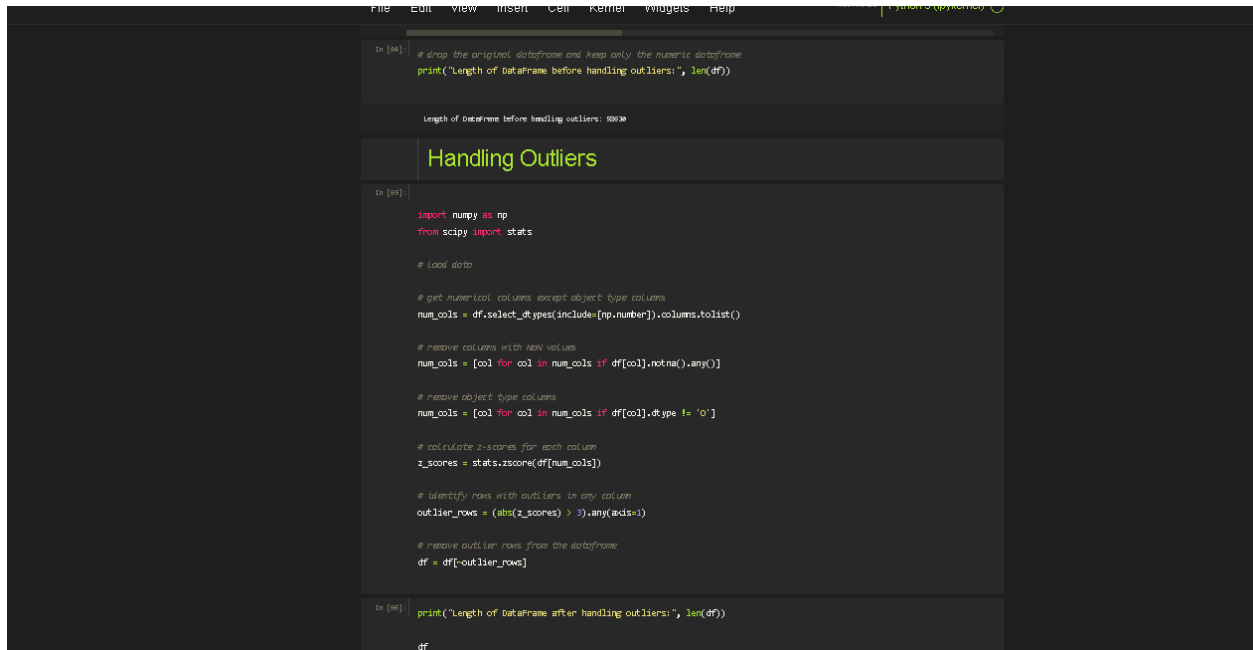
# remove outlier rows from the dataframe
df = df[~outlier_rows]
```

Visualize Using:

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

df
```

Demonstration:



The screenshot shows a Jupyter Notebook interface with a dark theme. The top menu bar includes File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. The notebook has two cells. The first cell, labeled 'In [94]', contains code to drop the original dataframe and keep only the numeric one, followed by a print statement showing the length of the dataframe before handling outliers as 8000. The second cell, labeled 'In [95]', is titled 'Handling Outliers' and contains code to import numpy and scipy, load data, get numerical columns, remove columns with NaN values, remove object type columns, calculate z-scores, identify rows with outliers, and remove those rows from the dataframe. The final print statement shows the length of the dataframe after handling outliers as 7999.

```
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: 8000

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()

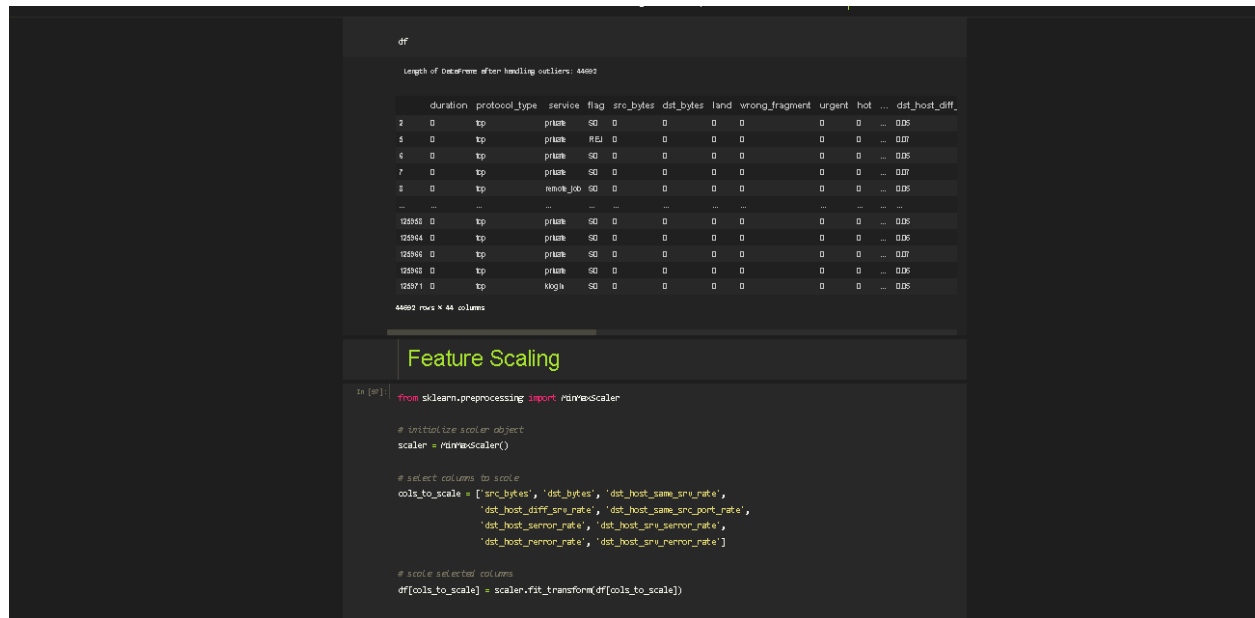
# select columns to scale
cols_to_scale = ['src_bytes', 'dst_bytes', 'dst_host_same_srv_rate',
                 'dst_host_diff_srv_rate', 'dst_host_same_src_port_rate',
                 'dst_host_serror_rate', 'dst_host_srv_serror_rate',
                 'dst_host_rerror_rate', 'dst_host_srv_rerror_rate']

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

For Demonstration Use:

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

Demonstration:



The screenshot shows a Jupyter Notebook interface. The top part displays a DataFrame named 'df' with a message 'Length of dataframe after handling outliers: 44892'. Below this, a preview of the DataFrame is shown with columns: duration, protocol_type, service, flag, src_bytes, dst_bytes, land, wrong_fragment, urgent, hot, ..., dst_host_diff. The bottom part of the screenshot shows a code cell titled 'Feature Scaling' with the following Python code:

```
In [4]: from sklearn.preprocessing import MinMaxScaler

# initialize scaler object
scaler = MinMaxScaler()

# select columns to scale
cols_to_scale = ['src_bytes', 'dst_bytes', 'dst_host_same_src_rate',
                 'dst_host_diff_src_rate', 'dst_host_same_src_port_rate',
                 'dst_host_serror_rate', 'dst_host_sru_serror_rate',
                 'dst_host_rerror_rate', 'dst_host_sru_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

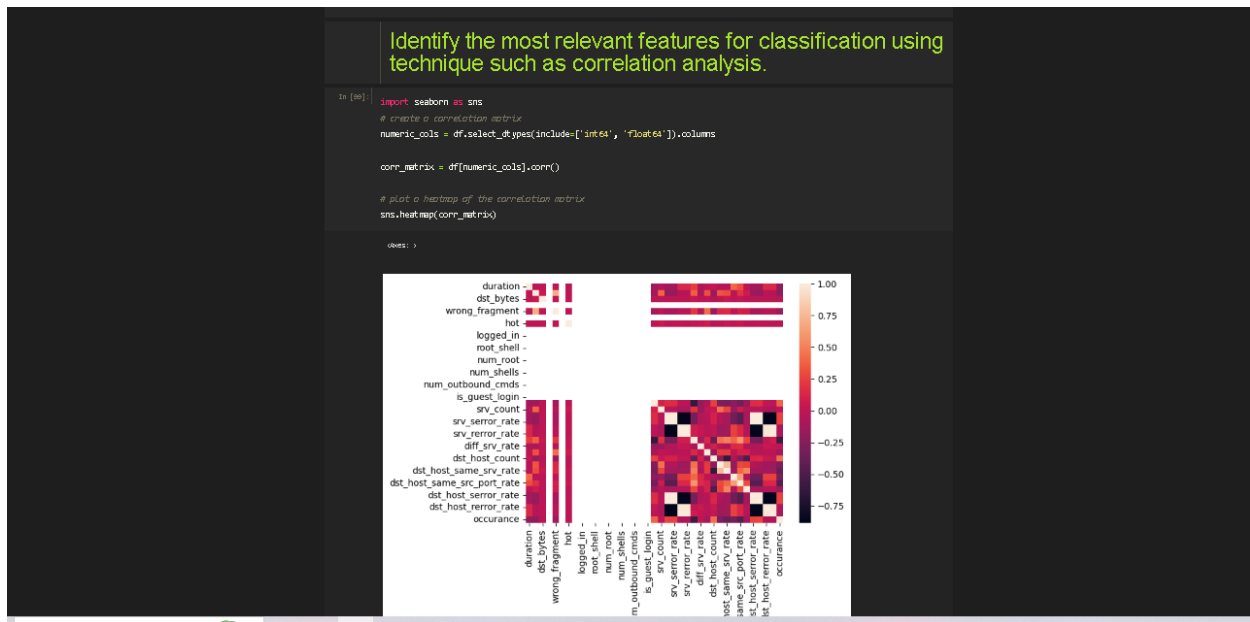
Code:

```
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:



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 to 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=0)

In [343]:
# Train the decision tree classifier
clf = DecisionTreeClassifier(random_state=0)
clf.fit(X_train, y_train)

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

In [344]:
# 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: 0.999552320330467
Precision: 0.999552320330467
Recall: 0.999552320330467
F1 score: 0.999552320330467

C:\Users\Vinay\AppData\Local\Programs\Python\Python38\lib\site-packages\sklearn\metrics\_classification.py:1364: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use 'zero_division' parameter to control this behavior.
  warn_jit(average, modifier, msg_start, len(result))
C:\Users\Vinay\AppData\Local\Programs\Python\Python38\lib\site-packages\sklearn\metrics\_classification.py:1364: UndefinedMetricWarning: Recall is ill-defined and being set to 0.0 in labels with no true samples. Use 'zero_division' parameter to control this behavior.
  warn_jit(average, modifier, msg_start, len(result))

In [345]:
df

```

| | duration | protocol_type | service | flag | src_bytes | dst_bytes | land | wrong_fragment | urgent | hot | ... | dst_host_diff |
|-------|----------|---------------|-----------|------|-----------|-----------|------|----------------|--------|-----|-----|---------------|
| 2 | 0 | tp | prstm | SD | 0.0 | 0.0 | 0 | 0 | 0 | 0 | ... | 0.000000 |
| 5 | 0 | tp | prstm | RST | 0.0 | 0.0 | 0 | 0 | 0 | 0 | ... | 0.000000 |
| 6 | 0 | tp | prstm | SD | 0.0 | 0.0 | 0 | 0 | 0 | 0 | ... | 0.000000 |
| 7 | 0 | tp | prstm | SD | 0.0 | 0.0 | 0 | 0 | 0 | 0 | ... | 0.000000 |
| 8 | 0 | tp | smack_joo | SD | 0.0 | 0.0 | 0 | 0 | 0 | 0 | ... | 0.000000 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 10595 | 0 | tp | prstm | SD | 0.0 | 0.0 | 0 | 0 | 0 | 0 | ... | 0.000000 |
| 10596 | 0 | tp | prstm | SD | 0.0 | 0.0 | 0 | 0 | 0 | 0 | ... | 0.000000 |
| 11000 | 0 | tp | prstm | SD | 0.0 | 0.0 | 0 | 0 | 0 | 0 | ... | 0.000000 |

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 [100]: # 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 [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)
```

For Accuracy Rate:

[illegible]

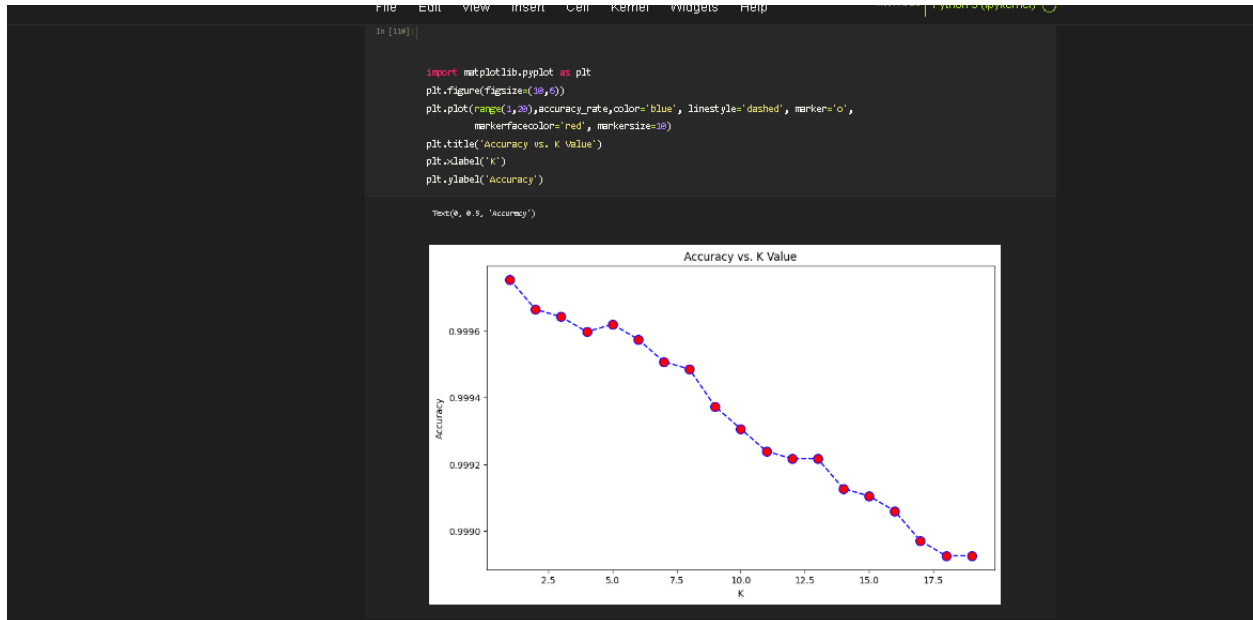
We will use plots in python for this purpose,

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:



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: 0.8992232380811
          Precision: 0.8992232380811
          Recall: 0.8992232380811
          F1-score: 0.8992232380811

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']
```

Splitting 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:

```

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

# 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]

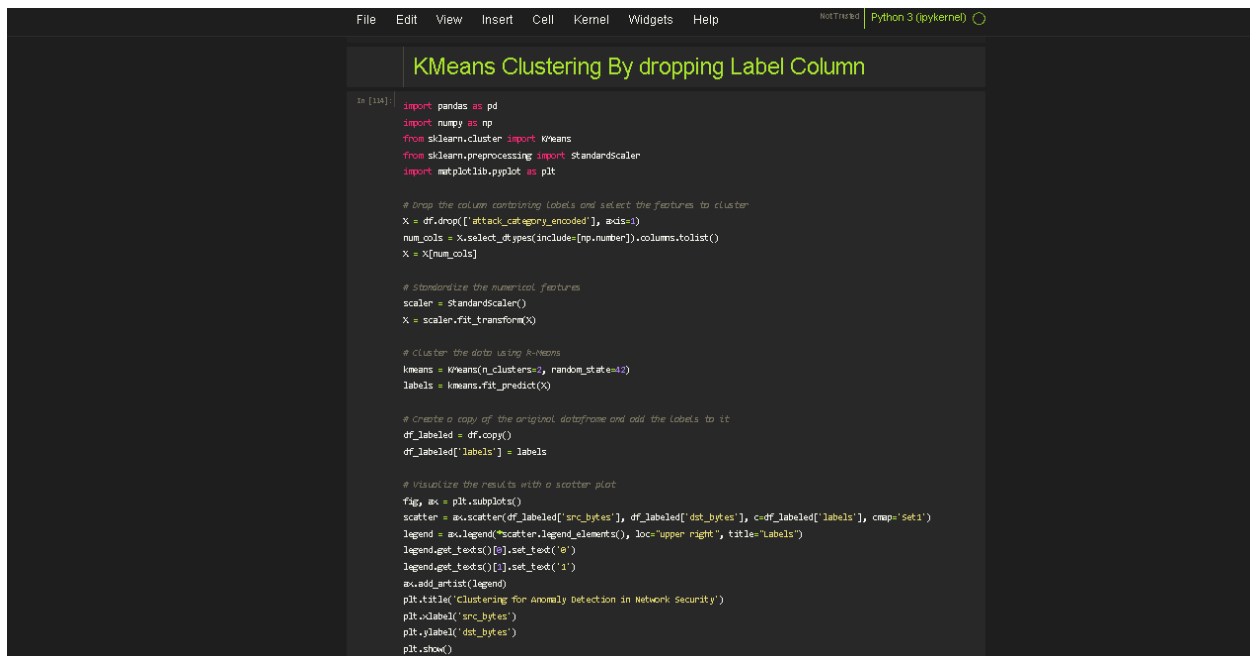
# Standardize the numerical features
scaler = StandardScaler()
X = scaler.fit_transform(X)

# 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 [134]: 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]

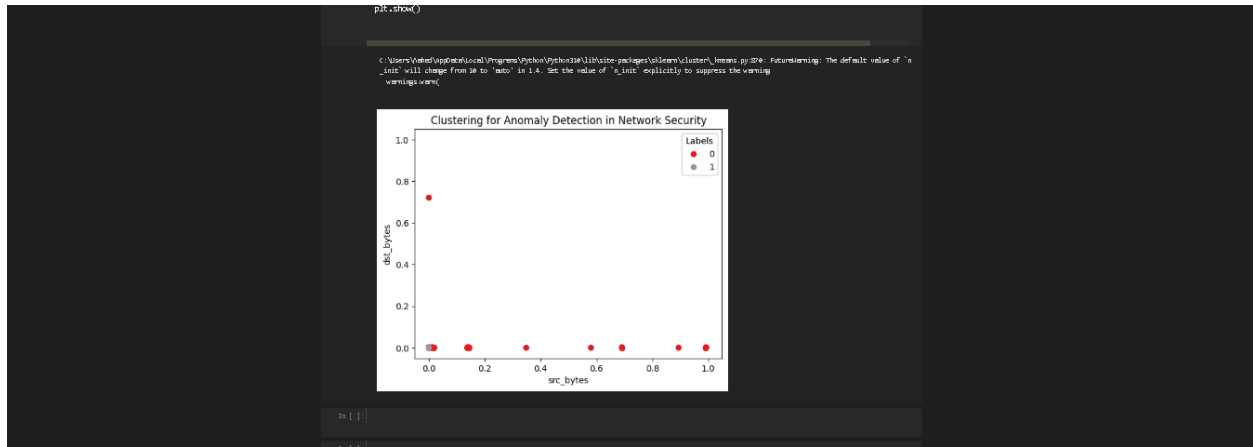
# Standardize the numerical features
scaler = StandardScaler()
X = scaler.fit_transform(X)

# 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:



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

FIN.