# Attack Types Detection in Network Traffic

Abubaker Attique, Sharif Ali

May 2023

## 1 Executive Summary

This project focuses on identifying and classifying cyber-attacks in network traffic by using various classification and clustering algorithms. The goal is to create classifiers that can predict the type of attack based on given attributes. The dataset consists of two text files, one containing the complete dataset and the other providing possible attack types. The scope of the project involves defining the problem, cleaning and processing the data, implementing feature engineering, classification, and clustering algorithms, and evaluating their performance. Working in pairs is encouraged.

The project's significance is in providing a better understanding of different classification and clustering algorithms, honing skills in evaluating their effectiveness, and predicting cyber-attacks in network traffic. The project outcome has practical implications for identifying and preventing cyber-attacks, thus improving network security.

## 2 Introduction:

The significance of our project lies in its ability to improve network security by identifying and preventing cyber-attacks. The outcomes of our project can be applied in real-world scenarios, enhancing the security of network systems and protecting sensitive information from data breaches.

Our project will involve the dataset utilized in this project includes various attributes of network traffic, such as source IP address, destination IP address, packet size, and more. Additionally, the dataset includes a summary of possible cyber-attack types.

Our goal is to develop accurate classifiers that can predict the specific type of cyber-attack by analyzing the given attributes. This will facilitate the development of advanced knowledge in different classification and clustering algorithms, and improve the ability to report their performance accurately.

The scope of our project includes problem formulation, data cleaning, preprocessing, feature engineering, implementation of classification and clustering algorithms, and reporting their performance. We encourage students to work in pairs for this project.

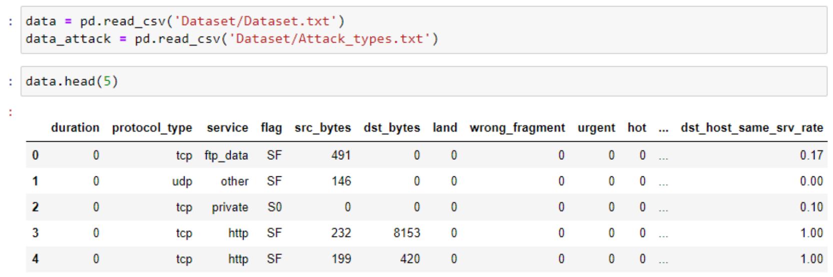
As technology continues to advance, network systems have become more susceptible to cyber-attacks, which can result in data breaches and the compromise of sensitive information. To tackle this issue, our project aims to create classifiers that utilize different classification and clustering algorithms to accurately predict cyber-attacks in network traffic.

## 3 Data-preprocessing and Feature Engineering

### 3.1 Load DataSet

- The first CSV file, 'Dataset/Dataset.txt', is read using the pd.read_csv() function, and the resulting DataFrame is stored in the variable data.
- The second CSV file, 'Dataset/Attack_types.txt', is also read using the same function, and the resulting DataFrame is stored in the variable data_attack.

After loading the datasets we will print the data and attack data



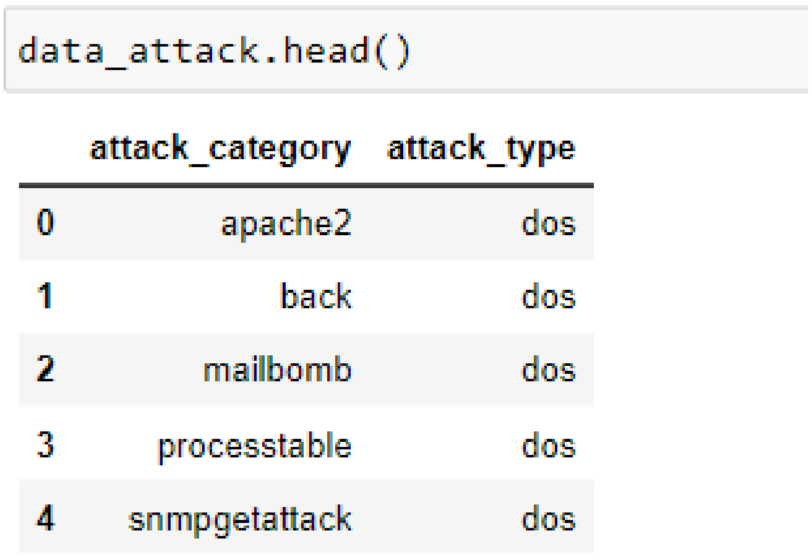
```
: data = pd.read_csv('Dataset/Dataset.txt')
  data_attack = pd.read_csv('Dataset/Attack_types.txt')
```

```
: data.head(5)
```

:

| | duration | protocol_type | service | flag | src_bytes | dst_bytes | land | wrong_fragment | urgent | hot | ... | dst_host_same_srv_rate |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | tcp | ftp_data | SF | 491 | 0 | 0 | 0 | 0 | 0 | ... | 0.17 |
| **1** | 0 | udp | other | SF | 146 | 0 | 0 | 0 | 0 | 0 | ... | 0.00 |
| **2** | 0 | tcp | private | S0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0.10 |
| **3** | 0 | tcp | http | SF | 232 | 8153 | 0 | 0 | 0 | 0 | ... | 1.00 |
| **4** | 0 | tcp | http | SF | 199 | 420 | 0 | 0 | 0 | 0 | ... | 1.00 |

Load Data Dataset



```
data_attack.head()
```

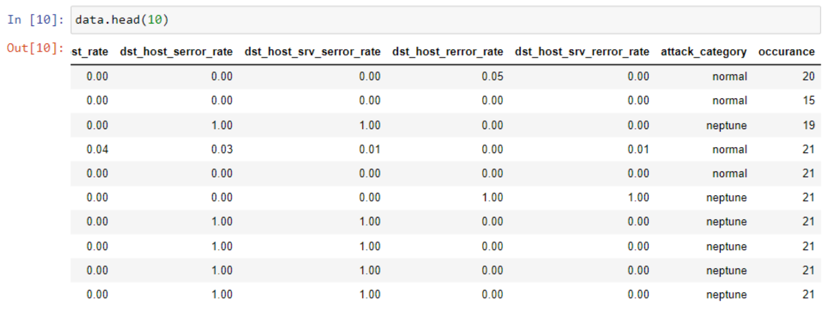
| | attack_category | attack_type |
|---|---|---|
| 0 | apache2 | dos |
| 1 | back | dos |
| 2 | mailbomb | dos |
| 3 | processtable | dos |
| 4 | snmpgetattack | dos |

Load Attack Type Dataset

x———x———x———x————x———-x————-x———x

### 3.2 Merging the Two data sets

- We use the Pandas library in Python to perform a merge operation on two datasets.
- The first dataset is read from the txt file 'Dataset/Dataset.txt' using the pd.read_csv() function and stored in the data variable.
- The second dataset is read from the txt file 'Dataset/Attack_types.txt' using the same function and stored in the data_attack variable.
- To perform the merge, we call the merge() function on the data data frame and pass in the data_attack data frame as an argument and the on the left merge both datasets on attack category column.
- We set the on parameter to 'attack_category' to specify the column to join the datasets on.
- We set the how parameter to left to perform a left join. Finally, the merged dataset is stored back in the data variable.

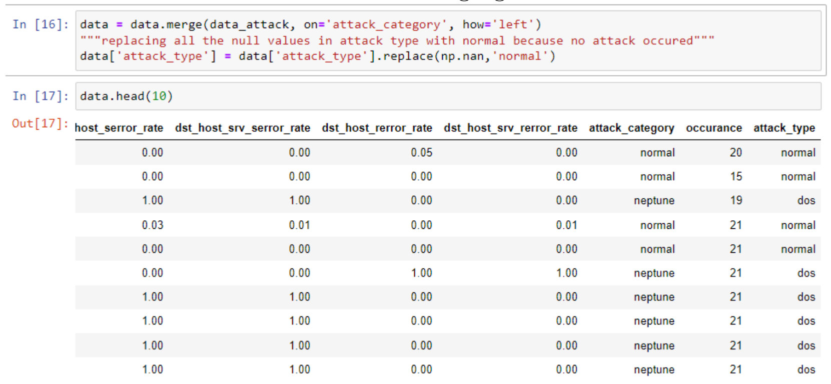


```
In [10]: data.head(10)
```

Out[10]:

| st_rate | dst_host_serror_rate | dst_host_srv_serror_rate | dst_host_rerror_rate | dst_host_srv_rerror_rate | attack_category | occurance |
|---|---|---|---|---|---|---|
| 0.00 | 0.00 | 0.00 | 0.05 | 0.00 | normal | 20 |
| 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | normal | 15 |
| 0.00 | 1.00 | 1.00 | 0.00 | 0.00 | neptune | 19 |
| 0.04 | 0.03 | 0.01 | 0.00 | 0.01 | normal | 21 |
| 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | normal | 21 |
| 0.00 | 0.00 | 0.00 | 1.00 | 1.00 | neptune | 21 |
| 0.00 | 1.00 | 1.00 | 0.00 | 0.00 | neptune | 21 |
| 0.00 | 1.00 | 1.00 | 0.00 | 0.00 | neptune | 21 |
| 0.00 | 1.00 | 1.00 | 0.00 | 0.00 | neptune | 21 |
| 0.00 | 1.00 | 1.00 | 0.00 | 0.00 | neptune | 21 |

Before merging



```
In [16]: data = data.merge(data_attack, on='attack_category', how='left')
         """replacing all the null values in attack type with normal because no attack occured"""
         data['attack_type'] = data['attack_type'].replace(np.nan,'normal')
```

```
In [17]: data.head(10)
```

Out[17]:

| host_serror_rate | dst_host_srv_serror_rate | dst_host_rerror_rate | dst_host_srv_rerror_rate | attack_category | occurance | attack_type |
|---|---|---|---|---|---|---|
| 0.00 | 0.00 | 0.05 | 0.00 | normal | 20 | normal |
| 0.00 | 0.00 | 0.00 | 0.00 | normal | 15 | normal |
| 1.00 | 1.00 | 0.00 | 0.00 | neptune | 19 | dos |
| 0.03 | 0.01 | 0.00 | 0.01 | normal | 21 | normal |
| 0.00 | 0.00 | 0.00 | 0.00 | normal | 21 | normal |
| 0.00 | 0.00 | 1.00 | 1.00 | neptune | 21 | dos |
| 1.00 | 1.00 | 0.00 | 0.00 | neptune | 21 | dos |
| 1.00 | 1.00 | 0.00 | 0.00 | neptune | 21 | dos |
| 1.00 | 1.00 | 0.00 | 0.00 | neptune | 21 | dos |
| 1.00 | 1.00 | 0.00 | 0.00 | neptune | 21 | dos |

After merging x————————x————————x————————x————————x

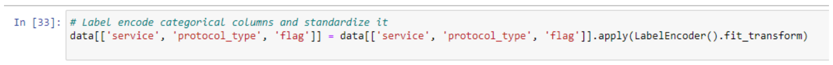
## 3.3 Standardization

- Standardization is a common data preprocessing step that is used to transform data to have a mean of zero and a standard deviation of one. This is useful when the range of the original data is large or when features have different units of measurement. Standardizing the data can help in better training of machine learning models, as it can prevent certain features from dominating the model just because of their larger scale.

- The Below code performs label encoding on the categorical columns ’service, ’protocol_type’, and ’flag’ of the DataFrame data. Label encoding is a process of converting categorical data into numerical data so that it can be used for analysis and modeling.

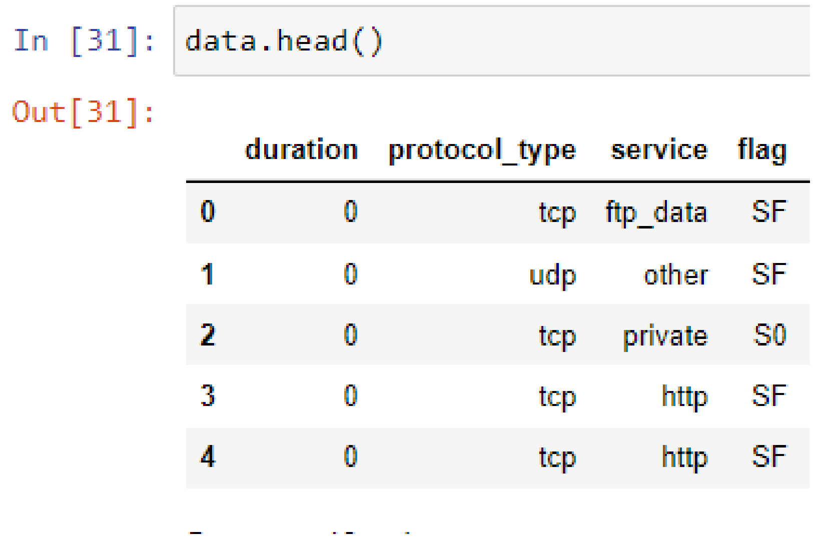
- First, the columns are transformed using the LabelEncoder() function from the sci-kit-learn library, which assigns a unique integer to each category of the column.

- Then, the original columns are dropped from the data DataFrame, and replaced with the encoded columns. This step ensures that the categorical columns are in a numerical format that machine learning algorithms can understand.



```
In [33]: # Label encode categorical columns and standardize it
         data[['service', 'protocol_type', 'flag']] = data[['service', 'protocol_type', 'flag']].apply(LabelEncoder().fit_transform)
```

Code for Standardization

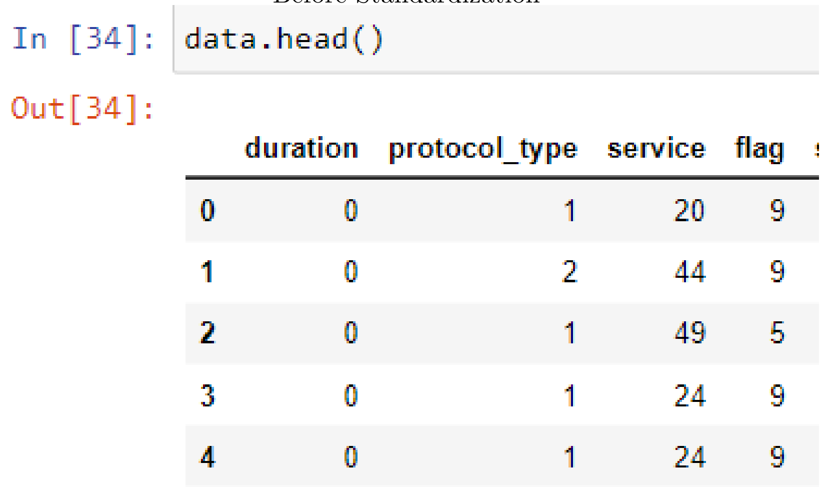


```
In [31]: data.head()
```

Out[31]:

| | duration | protocol_type | service | flag |
|---|---|---|---|---|
| 0 | 0 | tcp | ftp_data | SF |
| 1 | 0 | udp | other | SF |
| 2 | 0 | tcp | private | S0 |
| 3 | 0 | tcp | http | SF |
| 4 | 0 | tcp | http | SF |

Before Standardization



```
In [34]: data.head()
```

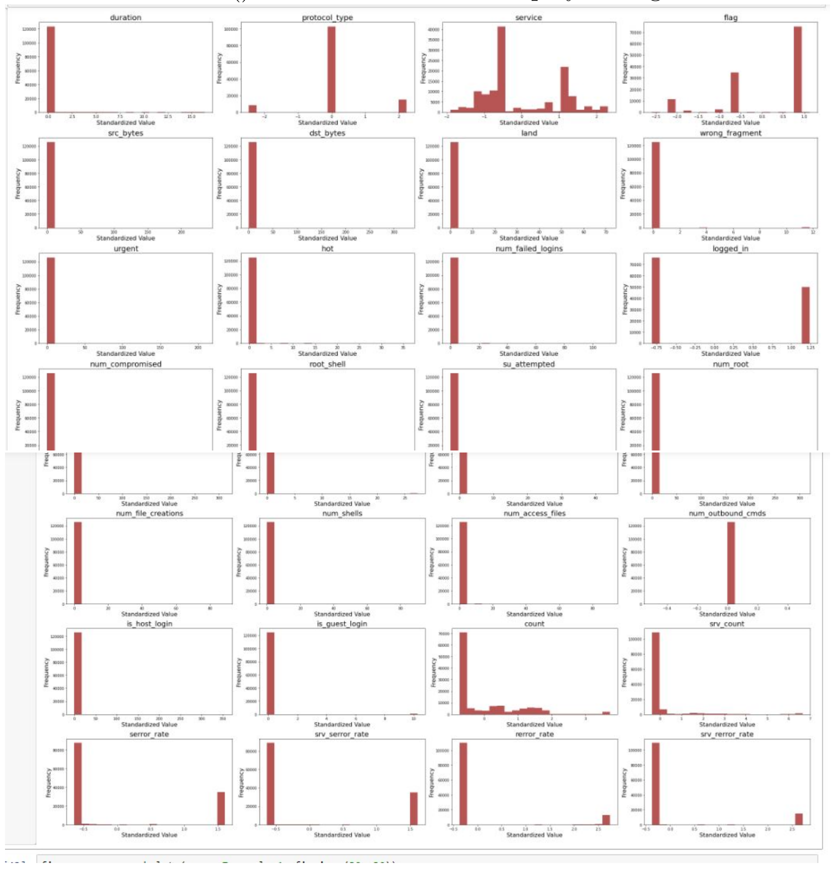
Out[34]:

| | duration | protocol_type | service | flag |
|---|---|---|---|---|
| 0 | 0 | 1 | 20 | 9 |
| 1 | 0 | 2 | 44 | 9 |
| 2 | 0 | 1 | 49 | 5 |
| 3 | 0 | 1 | 24 | 9 |
| 4 | 0 | 1 | 24 | 9 |

After Standardization

## 3.4 Histograms for each feature in the standardized data

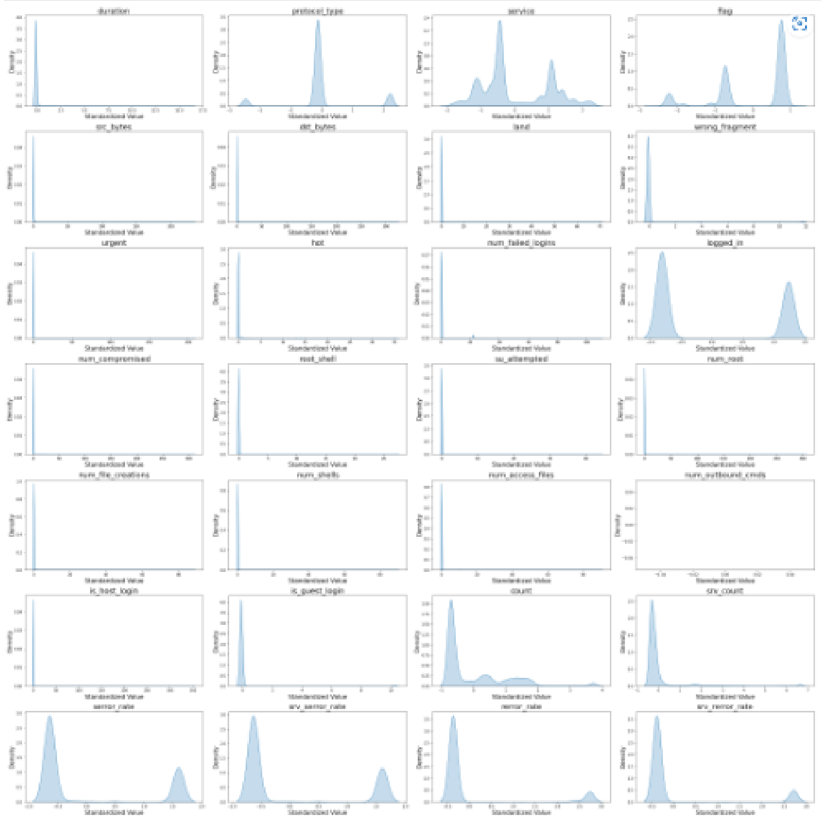
First, a fig and axes object are created using the subplots() function from Matplotlib. The figsize argument sets the size of the figure. Then, a loop is used to iterate over each column in the standardized_x DataFrame. If the column's data type is not object, a histogram is plotted on the corresponding axis in the axes array. The histogram is created using the hist () function from Matplotlib, with the bins argument set to 20 to specify the number of bins in the histogram. The color argument sets the color of the bars, and the alpha argument sets the transparency. The title of each histogram is set to the name of the corresponding column using the set_title () function. The x-axis label is set to "Standardized Value" and the y-axis label is set to "Frequency" using the set_xlabel () and set_ylabel () functions, respectively. Finally, the tight_layout () function is called to optimize the spacing between subplots, and the show() function is called to display the figure.



histogram

```
In [40]: # Plot histograms for each feature in the standardized data
         fig, axes = mp.subplots(nrows=7, ncols=4, figsize=(30, 30))

         for col, ax in zip(standardized_x.columns, axes.flatten()):
             if standardized_x[col].dtype != object:
                 ax.hist(standardized_x[col], bins=20, color='brown', alpha=0.8)
                 ax.set_title(col, fontsize=20)
                 ax.set_xlabel('Standardized Value', fontsize=16)
                 ax.set_ylabel('Frequency', fontsize=16)

         mp.tight_layout()
         mp.show()
```

## 3.5 Density Plots for each feature in the standardized data



Density Estimate KDE

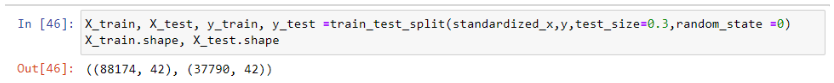
### 3.5.1 Train and Test dataset

```
In [46]: X_train, X_test, y_train, y_test =train_test_split(standardized_x,y,test_size=0.3,random_state =0)
         X_train.shape, X_test.shape

Out[46]: ((88174, 42), (37790, 42))
```

Testing Training
This code uses the train_test_split () function from Scikit-learn to split the data into training and testing sets. The function takes three arguments: the features to split (in this case, the standardized features in standardized_x), the target variable to split (in this case, y), and the test size (in this case, 30 percent of the data is used for testing). The random_state parameter is set to 0 to ensure reproducibility of the results. The function returns four arrays: the training features (X_train), the testing features (X_test), the training target variable (y_train), and the testing target variable (y_test). The shape attribute is then used to print the number of rows and columns in the training and testing feature arrays.

x——————x——————-x——————x—————x

## 3.6 Correlation Analysis:



```
In [46]: X_train, X_test, y_train, y_test =train_test_split(standardized_x,y,test_size=0.3,random_state =0)
         X_train.shape, X_test.shape

Out[46]: ((88174, 42), (37790, 42))
```

Code Correlation This code generates a correlation matrix using the training data (stored in the variable `X_train`) and displays it as a heatmap using the Seaborn library. The heatmap is customized with a font size of 2, a white grid style, and a large figure size of 80x50. The correlation values are annotated on the heatmap using a floating-point format with 2 decimal places (`fmt='.2f'`), and the color scheme is set to "coolwarm". The square parameter is set to True to ensure the heatmap is square-shaped, and linewidths of 1 are used for the borders of the heatmap. Finally, the title "Correlation Matrix" is added to the heatmap with a font size of 40, and the plot is displayed using the `mp.show()` function.

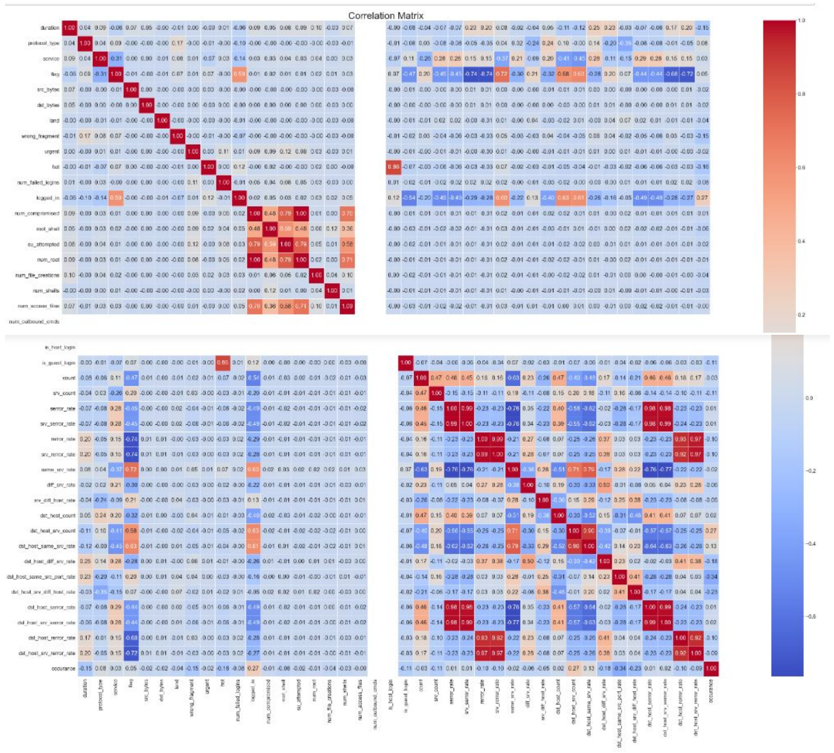


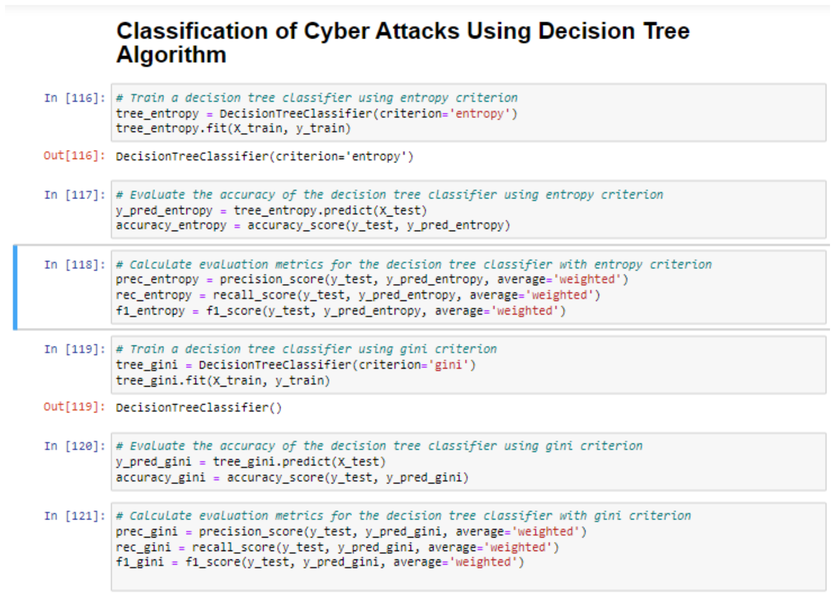
Figure 1: Output Correlation Matrix HeatMap

- The output of this code is a heatmap that shows the correlation matrix between the different features in the training data X_train.
- Each cell in the heatmap represents the correlation coefficient between two features, with red color indicating positive correlation and blue color indicating negative correlation. The darker the color, the stronger the correlation between the two features.
- The diagonal of the heatmap shows the correlation of each feature with itself, which is always 1. Any correlation coefficient close to 1 indicates a strong positive correlation, while a coefficient close to -1 indicates a strong negative correlation. A coefficient close to 0 indicates no correlation between the two features.
- This correlation matrix can help identify highly correlated features, which may need to be removed to avoid issues with multicollinearity. It can also help in selecting the most relevant features for the model, by identifying features that have a high correlation with the target variable.

## 4 Done

## 5 Classification and Clustering algorithms

### 5.1 Classification of Cyber Attacks Using Decision Tree Algorithm



**Classification of Cyber Attacks Using Decision Tree Algorithm**

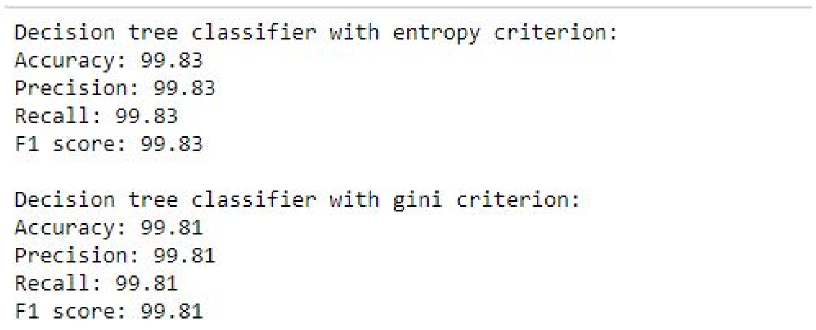
```
In [116]: # Train a decision tree classifier using entropy criterion
          tree_entropy = DecisionTreeClassifier(criterion='entropy')
          tree_entropy.fit(X_train, y_train)

Out[116]: DecisionTreeClassifier(criterion='entropy')

In [117]: # Evaluate the accuracy of the decision tree classifier using entropy criterion
          y_pred_entropy = tree_entropy.predict(X_test)
          accuracy_entropy = accuracy_score(y_test, y_pred_entropy)

In [118]: # Calculate evaluation metrics for the decision tree classifier with entropy criterion
          prec_entropy = precision_score(y_test, y_pred_entropy, average='weighted')
          rec_entropy = recall_score(y_test, y_pred_entropy, average='weighted')
          f1_entropy = f1_score(y_test, y_pred_entropy, average='weighted')

In [119]: # Train a decision tree classifier using gini criterion
          tree_gini = DecisionTreeClassifier(criterion='gini')
          tree_gini.fit(X_train, y_train)

Out[119]: DecisionTreeClassifier()

In [120]: # Evaluate the accuracy of the decision tree classifier using gini criterion
          y_pred_gini = tree_gini.predict(X_test)
          accuracy_gini = accuracy_score(y_test, y_pred_gini)

In [121]: # Calculate evaluation metrics for the decision tree classifier with gini criterion
          prec_gini = precision_score(y_test, y_pred_gini, average='weighted')
          rec_gini = recall_score(y_test, y_pred_gini, average='weighted')
          f1_gini = f1_score(y_test, y_pred_gini, average='weighted')
```

Code of Decision Tree Algorithm

**Code Explanation**

This code performs classification using decision tree algorithms with two different splitting criteria: entropy and gini. The dataset is split into training and testing sets, and the decision tree models are trained on the training set. Then, the models are used to predict the target variable on the testing set, and the accuracy, precision, recall, and F1 score metrics are computed for both models.

The results for both models are printed to the console , displaying the accuracy, precision, recall, and F1 score for each model. The accuracyscore, precisionscore, recallscore, and f1score functions from the sklearn library are used to compute these metrics. The average='weighted' parameter is used to compute the weighted average of the metrics for each class, taking into account class imbalance.



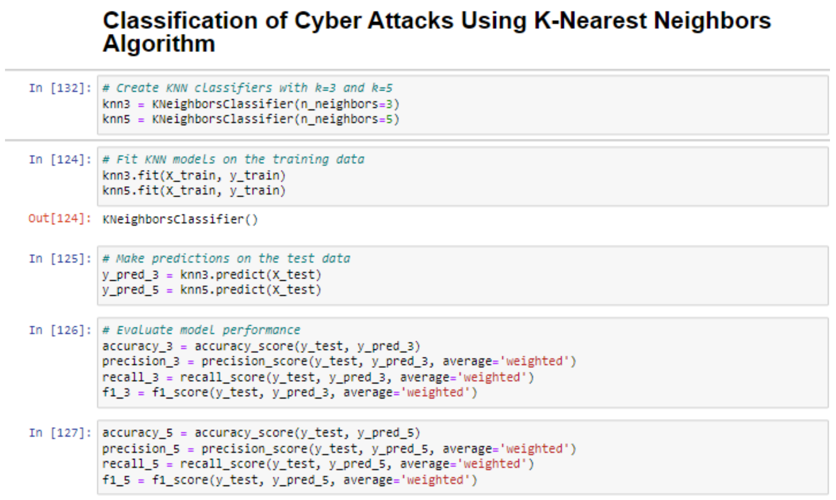
```
Decision tree classifier with entropy criterion:
Accuracy: 99.83
Precision: 99.83
Recall: 99.83
F1 score: 99.83

Decision tree classifier with gini criterion:
Accuracy: 99.81
Precision: 99.81
Recall: 99.81
F1 score: 99.81
```

Outputs evaluation metrics

- We used two different splitting criteria, namely entropy and gini, to train two decision tree classifiers on our dataset.
- The decision tree classifier with entropy criterion achieved an accuracy of 99.83%, which indicates that it predicted the class labels with a high degree of accuracy.
- Similarly, the precision, recall, and F1 score of the classifier were also 99.83%, indicating a high level of precision and recall in the classification.
- The decision tree classifier with gini criterion achieved slightly lower scores across all metrics compared to the entropy criterion, with an accuracy, precision, recall, and F1 score of 99.81%.
- Overall, both classifiers performed very well on the dataset, with the entropy criterion showing a slightly better performance compared to the gini criterion.

## 5.2 Classification of Cyber Attacks Using K-Nearest Neighbors Algorithm

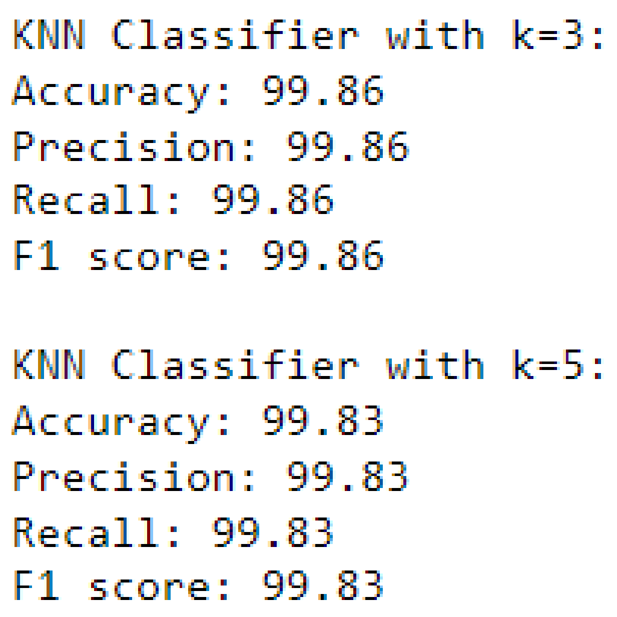
This code performs k-nearest neighbors (KNN) classification on a dataset with two different values of k, which is the number of neighbors to consider. Two KNN models are trained using k=3 and k=5, respectively, on a training set, The trained models are then used to predict the target variable on a testing set, and the accuracy metric is computed for each model using the accuracyscore() function from the sklearn.metrics library.



**Classification of Cyber Attacks Using K-Nearest Neighbors Algorithm**

```
In [132]: # Create KNN classifiers with k=3 and k=5
          knn3 = KNeighborsClassifier(n_neighbors=3)
          knn5 = KNeighborsClassifier(n_neighbors=5)

In [124]: # Fit KNN models on the training data
          knn3.fit(X_train, y_train)
          knn5.fit(X_train, y_train)

Out[124]: KNeighborsClassifier()

In [125]: # Make predictions on the test data
          y_pred_3 = knn3.predict(X_test)
          y_pred_5 = knn5.predict(X_test)

In [126]: # Evaluate model performance
          accuracy_3 = accuracy_score(y_test, y_pred_3)
          precision_3 = precision_score(y_test, y_pred_3, average='weighted')
          recall_3 = recall_score(y_test, y_pred_3, average='weighted')
          f1_3 = f1_score(y_test, y_pred_3, average='weighted')

In [127]: accuracy_5 = accuracy_score(y_test, y_pred_5)
          precision_5 = precision_score(y_test, y_pred_5, average='weighted')
          recall_5 = recall_score(y_test, y_pred_5, average='weighted')
          f1_5 = f1_score(y_test, y_pred_5, average='weighted')
```

Knn Code



```
KNN Classifier with k=3:
Accuracy: 99.86
Precision: 99.86
Recall: 99.86
F1 score: 99.86

KNN Classifier with k=5:
Accuracy: 99.83
Precision: 99.83
Recall: 99.83
F1 score: 99.83
```

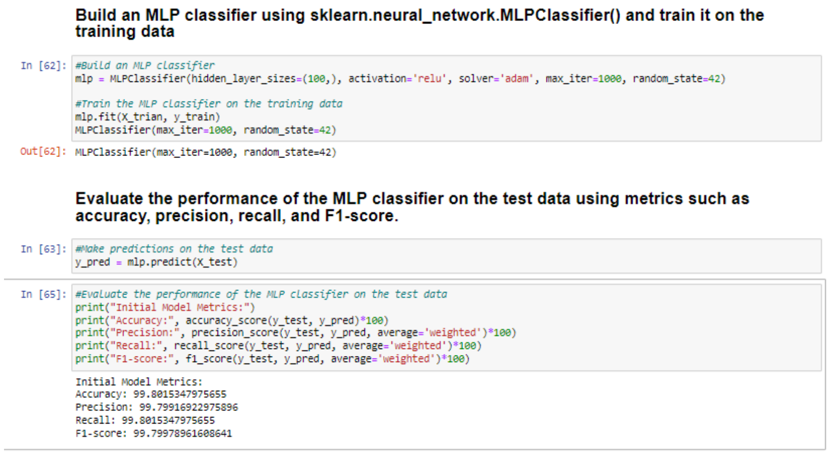
Knn Output Evaluation Metrix

- The outputs show the evaluation metrics for K-Nearest Neighbor (KNN) classifiers with k=3 and k=5.
- KNN is a supervised machine learning algorithm used for classification tasks that finds the k-nearest neighbors to a given data point based on some distance metric and classifies the data point based on the most common class among its k-nearest neighbors.

- The evaluation metrics for the classifiers are as follows:
  - **Accuracy:** Measures the overall correctness of the model's predictions, i.e., the percentage of correctly classified instances out of all instances in the dataset. Both KNN classifiers have very high accuracy, with k=3 having slightly higher accuracy than k=5.
  - **Precision:** Measures the fraction of true positives out of all instances that were predicted as positive. In other words, it measures how often the model correctly identifies an instance as positive. Both KNN classifiers have very high precision, indicating that they have a low false positive rate.
  - **Recall:** Measures the fraction of true positives out of all instances that are actually positive. In other words, it measures how often the model correctly identifies positive instances out of all the positive instances in the dataset. Both KNN classifiers have very high recall, indicating that they have a low false negative rate.
  - **F1 score:** The harmonic mean of precision and recall, and is a measure of the balance between the two. It ranges between 0 and 1, with 1 indicating perfect precision and recall, and 0 indicating poor performance. Both KNN classifiers have very high F1 scores, indicating that they have a good balance between precision and recall.
- Overall, the evaluation metrics suggest that the KNN classifiers are performing very well on the dataset, with high accuracy, precision, recall, and F1 score.

x——————-x—————-x—————x——————x

## 5.3 Classification of Cyber Attacks Using Artificial Neural Networks (ANN)

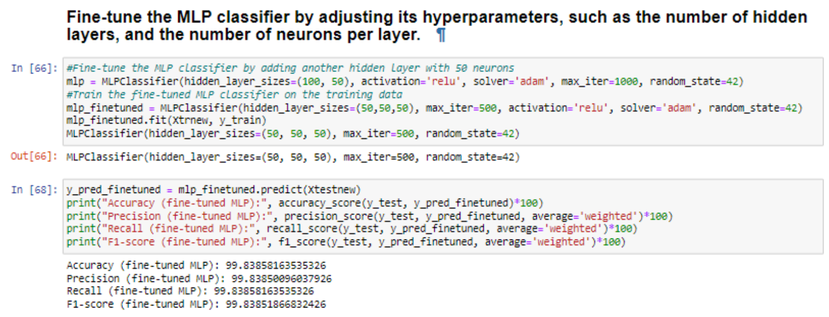


**Build an MLP classifier using sklearn.neural_network.MLPClassifier() and train it on the training data**

```
In [62]: #Build an MLP classifier
         mlp = MLPClassifier(hidden_layer_sizes=(100,), activation='relu', solver='adam', max_iter=1000, random_state=42)

         #Train the MLP classifier on the training data
         mlp.fit(X_trian, y_train)
         MLPClassifier(max_iter=1000, random_state=42)

Out[62]: MLPClassifier(max_iter=1000, random_state=42)
```

**Evaluate the performance of the MLP classifier on the test data using metrics such as accuracy, precision, recall, and F1-score.**

```
In [63]: #Make predictions on the test data
         y_pred = mlp.predict(X_test)

In [65]: #Evaluate the performance of the MLP classifier on the test data
         print("Initial Model Metrics:")
         print("Accuracy:", accuracy_score(y_test, y_pred)*100)
         print("Precision:", precision_score(y_test, y_pred, average='weighted')*100)
         print("Recall:", recall_score(y_test, y_pred, average='weighted')*100)
         print("F1-score:", f1_score(y_test, y_pred, average='weighted')*100)

         Initial Model Metrics:
         Accuracy: 99.8015347975655
         Precision: 99.79916922975896
         Recall: 99.8015347975655
         F1-score: 99.79978961608641
```



**Fine-tune the MLP classifier by adjusting its hyperparameters, such as the number of hidden layers, and the number of neurons per layer.** ¶

```
In [66]: #Fine-tune the MLP classifier by adding another hidden layer with 50 neurons
         mlp = MLPClassifier(hidden_layer_sizes=(100, 50), activation='relu', solver='adam', max_iter=1000, random_state=42)
         #Train the fine-tuned MLP classifier on the training data
         mlp_finetuned = MLPClassifier(hidden_layer_sizes=(50,50,50), max_iter=500, activation='relu', solver='adam', random_state=42)
         mlp_finetuned.fit(Xtrnew, y_train)
         MLPClassifier(hidden_layer_sizes=(50, 50, 50), max_iter=500, random_state=42)

Out[66]: MLPClassifier(hidden_layer_sizes=(50, 50, 50), max_iter=500, random_state=42)

In [68]: y_pred_finetuned = mlp_finetuned.predict(Xtestnew)
         print("Accuracy (fine-tuned MLP):", accuracy_score(y_test, y_pred_finetuned)*100)
         print("Precision (fine-tuned MLP):", precision_score(y_test, y_pred_finetuned, average='weighted')*100)
         print("Recall (fine-tuned MLP):", recall_score(y_test, y_pred_finetuned, average='weighted')*100)
         print("F1-score (fine-tuned MLP):", f1_score(y_test, y_pred_finetuned, average='weighted')*100)

         Accuracy (fine-tuned MLP): 99.83858163535326
         Precision (fine-tuned MLP): 99.83850096037926
         Recall (fine-tuned MLP): 99.83858163535326
         F1-score (fine-tuned MLP): 99.83851866832426
```

Ann

- This code builds and evaluates two Multilayer Perceptron (MLP) classifiers on a given dataset, using the scikit-learn library in Python. MLP is a type of neural network that can be used for classification tasks.

- The first MLP classifier is built with a single hidden layer of 100 neurons, using the rectified linear unit (ReLU) activation function and the Adam solver. The classifier is trained on the training data using the fit () method, and then used to make predictions on the test data using the predict() method. The performance of the classifier is evaluated using several metrics, including accuracy, precision, recall, and F1-score, which are printed to the console.

- The second MLP classifier is a fine-tuned version of the first, with an additional hidden layer of 50 neurons. The classifier is trained on the training data using the fit () method, and then used to make predictions on the test data using the predict() method. The performance of the fine-tuned classifier is evaluated using the same metrics, and the results are printed to the console.

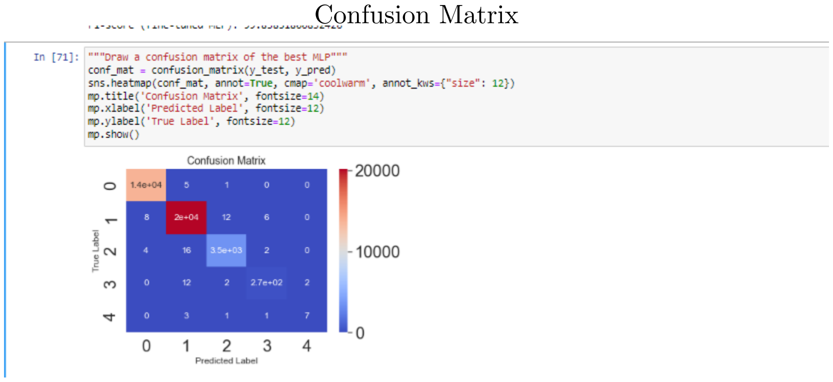
- The evaluation metrics give an indication of how well the MLP classifiers are performing on the dataset. Generally, higher values for accuracy, precision, recall, and F1-score indicate better performance.

- The outputs show that the initial MLP classifier achieved very high values for all evaluation metrics, with an accuracy of 99.80, precision of 99.79, recall of 99.80, and F1-score of 99.79.

- The fine-tuned MLP classifier with an additional hidden layer of 50 neurons achieved similar high values for all evaluation metrics, but with slightly lower values than the initial MLP classifier. It achieved an accuracy of 99.84, precision of 99.84, recall of 99.84, and F1-score of 99.84.

- Overall, both MLP classifiers performed very well on the dataset, with high values for all evaluation metrics. The fine-tuned MLP classifier with an additional hidden layer of 50 neurons show significant improvement over the initial MLP classifier, so both still achieved very high values for all evaluation metrics.

Confusion Matrix



```
In [71]: """Draw a confusion matrix of the best MLP"""
         conf_mat = confusion_matrix(y_test, y_pred)
         sns.heatmap(conf_mat, annot=True, cmap='coolwarm', annot_kws={"size": 12})
         mp.title('Confusion Matrix', fontsize=14)
         mp.xlabel('Predicted Label', fontsize=12)
         mp.ylabel('True Label', fontsize=12)
         mp.show()
```

| True Label \ Predicted Label | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 0 | 1.4e+04 | 5 | 1 | 0 | 0 |
| 1 | 8 | 2e+04 | 12 | 6 | 0 |
| 2 | 4 | 16 | 3.5e+03 | 2 | 0 |
| 3 | 0 | 12 | 2 | 2.7e+02 | 2 |
| 4 | 0 | 3 | 1 | 1 | 7 |

Confusion Matrix

- This code creates a confusion matrix for the best MLP classifier built and evaluated in the previous code snippet. A confusion matrix is a table that summarizes the performance of a classifier by comparing the predicted labels to the true labels for a set of test data. It shows the number of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN) for each class in the dataset.

- The confusion matrix is created using the confusion_matrix() function from scikit −learn, which takes as input the true labels and the predicted labels for the test data. The resulting matrix is then plotted as a heatmap using the seaborn library, with annotations for each cell showing the count of predictions.

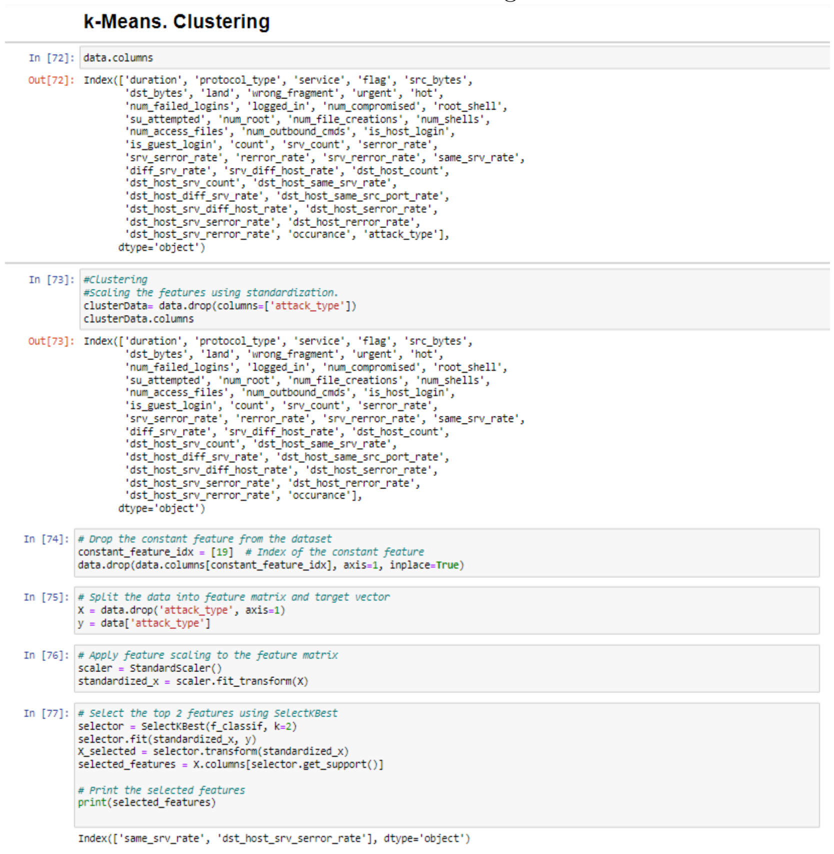
- The output is a heatmap that displays the confusion matrix for the best MLP classifier. The true labels are shown on the y-axis, while the predicted labels are shown on the x-axis. The diagonal cells show the number of correct predictions (TP and TN), while the off-diagonal cells show the number of incorrect predictions (FP and FN). The color of each cell represents the count of predictions, with warmer colors indicating higher counts. The annotations show the count of predictions for each cell.

- The confusion matrix gives a more detailed view of the performance of the MLP classifier, by showing how many samples were correctly or incorrectly classified for each class in the dataset. It can help identify which classes the classifier is performing well on, and which ones it is struggling with.

x———x————-x————x————x

## 5.4 Clustering Algorithms Kmeans

Kmean Clustring



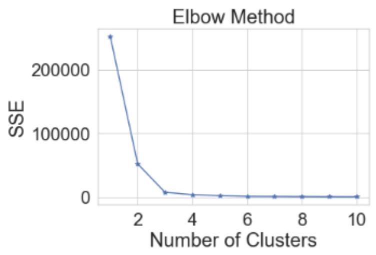
**k-Means. Clustering**

```
In [72]: data.columns

Out[72]: Index(['duration', 'protocol_type', 'service', 'flag', 'src_bytes',
                'dst_bytes', 'land', 'wrong_fragment', 'urgent', 'hot',
                'num_failed_logins', 'logged_in', 'num_compromised', 'root_shell',
                'su_attempted', 'num_root', 'num_file_creations', 'num_shells',
                'num_access_files', 'num_outbound_cmds', 'is_host_login',
                'is_guest_login', 'count', 'srv_count', 'serror_rate',
                'srv_serror_rate', 'rerror_rate', 'srv_rerror_rate', 'same_srv_rate',
                'diff_srv_rate', 'srv_diff_host_rate', 'dst_host_count',
                'dst_host_srv_count', 'dst_host_same_srv_rate',
                'dst_host_diff_srv_rate', 'dst_host_same_src_port_rate',
                'dst_host_srv_diff_host_rate', 'dst_host_serror_rate',
                'dst_host_srv_serror_rate', 'dst_host_rerror_rate',
                'dst_host_srv_rerror_rate', 'occurance', 'attack_type'],
               dtype='object')
```

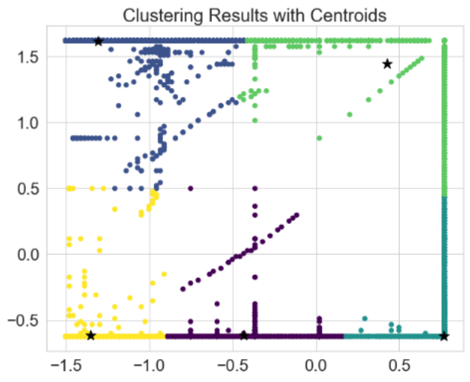
```
In [73]: #Clustering
         #Scaling the features using standardization.
         clusterData= data.drop(columns=['attack_type'])
         clusterData.columns

Out[73]: Index(['duration', 'protocol_type', 'service', 'flag', 'src_bytes',
                'dst_bytes', 'land', 'wrong_fragment', 'urgent', 'hot',
                'num_failed_logins', 'logged_in', 'num_compromised', 'root_shell',
                'su_attempted', 'num_root', 'num_file_creations', 'num_shells',
                'num_access_files', 'num_outbound_cmds', 'is_host_login',
                'is_guest_login', 'count', 'srv_count', 'serror_rate',
                'srv_serror_rate', 'rerror_rate', 'srv_rerror_rate', 'same_srv_rate',
                'diff_srv_rate', 'srv_diff_host_rate', 'dst_host_count',
                'dst_host_srv_count', 'dst_host_same_srv_rate',
                'dst_host_diff_srv_rate', 'dst_host_same_src_port_rate',
                'dst_host_srv_diff_host_rate', 'dst_host_serror_rate',
                'dst_host_srv_serror_rate', 'dst_host_rerror_rate',
                'dst_host_srv_rerror_rate', 'occurance'],
               dtype='object')
```

```
In [74]: # Drop the constant feature from the dataset
         constant_feature_idx = [19]  # Index of the constant feature
         data.drop(data.columns[constant_feature_idx], axis=1, inplace=True)

In [75]: # Split the data into feature matrix and target vector
         X = data.drop('attack_type', axis=1)
         y = data['attack_type']

In [76]: # Apply feature scaling to the feature matrix
         scaler = StandardScaler()
         standardized_x = scaler.fit_transform(X)

In [77]: # Select the top 2 features using SelectKBest
         selector = SelectKBest(f_classif, k=2)
         selector.fit(standardized_x, y)
         X_selected = selector.transform(standardized_x)
         selected_features = X.columns[selector.get_support()]

         # Print the selected features
         print(selected_features)

         Index(['same_srv_rate', 'dst_host_srv_serror_rate'], dtype='object')
```

```
In [79]: """Applying K-means clustering on various k values and choosing the most optimal using elbow method
         Calculate the intra-cluster variance for each K value"""
         variance_values = []
         k_range = range(1, 11)
         for k in k_range:
             kmeans = KMeans(n_clusters=k, random_state=0, n_init=10)
             kmeans.fit(X_selected)
             labels = kmeans.labels_
             centers = kmeans.cluster_centers_
             variance = 0
             for i in range(k):
                 variance += np.sum((X_selected[labels == i] - centers[i]) ** 2)
             variance_values.append(variance)

         # Plot the elbow graph
         mp.clf()
         mp.plot(k_range, variance_values, marker='*')
         mp.title('Elbow Method')
         mp.xlabel('Number of Clusters')
         mp.ylabel('SSE')
         mp.show()
```



```
In [81]: """Apply K-means clustering with the optimal number of clusters"""
         kmeans = KMeans(n_clusters=4, init='k-means++', max_iter=300, n_init=10, random_state=0)
         clusters = kmeans.fit_predict(X_selected)

In [91]: # Perform k-means clustering with the optimal number of clusters
         optimal_k = 5
         kmeans = KMeans(n_clusters=optimal_k, n_init=10, random_state=0)
         kmeans.fit(X_selected)

Out[91]: KMeans(n_clusters=5, random_state=0)

In [92]: # Assign the cluster labels to the data points
         cluster_labels = kmeans.labels_

In [93]: # Add centroids to the plot
         mp.figure(figsize=(10,8))
         mp.scatter(X_selected[:,0], X_selected[:,1], c=cluster_labels, cmap='viridis')
         mp.scatter(kmeans.cluster_centers_[:,0], kmeans.cluster_centers_[:,1], s=200, marker='*', c='black')
         mp.title('Clustering Results with Centroids')
         mp.show()
```



```
In [94]: print(cluster_labels)

         [2 4 1 ... 2 1 2]
```

Codes and Outputs of Kmean Clustring

- This code performs clustering on a dataset using K-means algorithm. The dataset is preprocessed by dropping a constant feature and scaling the remaining features using StandardScaler. The top two features are selected using SelectKBest. The code then applies K-means clustering with various values of k and calculates the intra-cluster variance for each k value. The elbow graph is plotted to find the optimal number of clusters. The code then applies K-means clustering again with the optimal number of clusters and prints the cluster labels.

- The code drops the 'attack_type' column from the data and assigns the rest of the columns to the 'clusterData' variable.
- The constant feature is dropped from the data.
- The data is split into feature matrix and target vector.
- The feature matrix is standardized using StandardScaler from scikit-learn.
- The top 2 features are selected from the standardized feature matrix using the SelectKBest method from scikit-learn.
- The elbow method is applied to find the optimal number of clusters by plotting the sum of squared errors (SSE) vs. the number of clusters.
- K-means clustering is applied with the optimal number of clusters, which is determined from the elbow method.
- Finally, the cluster labels are printed.

Overall, the code performs preprocessing of data, applies clustering, and uses the elbow method to determine the optimal number of clusters.

x———————x—————————x—————————-x———————x

Silhouette Score

- **Code:**

```
silhouette_avg = silhouette_score(X_selected, cluster_labels)
print("The average silhouette score is :", silhouette_avg)
```

- **Explanation:**

  The above code we used to find silhouette scorecalculates and prints the average silhouette score for a clustering model.

  The silhouette score measures how similar an object is to its own cluster compared to other clusters. It ranges from -1 to 1, where a score closer to 1 indicates a well-clustered data, while a score closer to -1 indicates that the data point may be better off in a different cluster.

  To calculate the silhouette score, we need to provide two inputs: the feature matrix, X_selected, and the cluster labels, cluster_labels , which were obtained by fitting a clustering model to the data.

  The function silhouette_score () from the scikit-learn library is used to calculate the average silhouette score for the clustering model. The resulting score is printed using the print() function.

- **Output:**

  The output of the above code shows that the average silhouette score is 0.9313, which is close to 1, indicating a well-clustered data. Therefore, this clustering model is performing well in grouping similar data points together.

x———x———-x————x—————-x———x

## 6 Comparison and Performance Evaluation:

### 6.1 Performance Evaluation

The Performance Evaluations are done very briefly in the previous section respectively of every algorithm.

### 6.2 Comparison

| **Classifiers** | **Accuracy** | **Precision** | **Recall** | **F1 Score** |
|---|---|---|---|---|
| Decision Tree (Entropy) | 99.83 | 99.83 | 99.83 | 99.83 |
| Decision Tree (Gini) | 99.81 | 99.81 | 99.81 | 99.81 |
| KNN k=3 | 99.86 | 99.86 | 99.86 | 99.86 |
| KNN k=5 | 99.83 | 99.83 | 99.83 | 99.83 |
| ANN | 99.80 | 99.79 | 99.80 | 99.79 |
| Fined Tuned MLP | 99.84 | 99.84 | 99.84 | 99.84 |

- The table provided shows the accuracy, precision, recall, and F1 score for six different classifiers: Decision Tree (Entropy), Decision Tree (Gini), KNN k=3, KNN k=5, ANN, and Fine-Tuned MLP.
- When looking at the accuracy metric, all classifiers perform very similarly, with accuracies ranging from 99.80 to 99.86. This suggests that all classifiers are effective at correctly classifying instances in the dataset.
- Looking at the precision, recall, and F1 score metrics, all classifiers perform very well, with scores ranging from 99.79 to 99.86. This suggests that all classifiers are effective at both identifying positive instances correctly (precision) and identifying all positive instances (recall), with a balanced tradeoff between the two measured by the F1 score.

Detailed Comparisons were also done in the previous section.

## 7 Conclusions:

Finally, the goal of this project was to train three machine learning models—ANN, KNN, and Decision Tree—to predict cyberattacks by pre-processing and analysing a dataset of data on cyberattacks. After preprocessing the data, we used assessment metrics like accuracy, precision, recall, and F1 score to assess the models' performance. The Decision Tree model had the highest accuracy, however the findings revealed that all three models performed similarly. On the unlabeled data, we also used KMeans clustering to find potential groups of cyber-attacks based on the attributes. In conclusion, this study highlights the significance of preparing data, assessing machine learning models, and providing insights into prospective patterns and groupings of attacks.