NETWORK ANOMALY DETECTION USING MACHINE LEARNING TECHNIQUES

PROJECT REPORT

Submitted in the partial fulfillment of the

requirementsfor award of the

Six Months Online Certificate Course

in

Cyber Security

Course Duration: [22-01-2024 to 21-07-2024]

By

Bangari Shanmukh(Ht.No.2406CYS14)

Under the Esteemed Guidance

Dr. Shivarama Krishna T.JNTUK

DIRECTORATE OF INNOVATIVE LEARNING & TEACHING(DILT)

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY HYDERABAD

(Formerly SCDE_SCHOOL OF CONTINUING AND DISTANCE EDUCATION)
Kukatpally, Hyderabad, Telangana State, INDIA- 500085

JULY 2024

ABSTRACT

In the evolving landscape of network security, the identification and mitigation of anomalies have become paramount to safeguarding data integrity and ensuring reliable communication systems. This project, "Network Anomaly Detection Using Machine Learning Techniques," aims to develop a robust framework capable of identifying and responding to network anomalies in real-time. The core objective is to leverage advanced machine learning algorithms to detect deviations from normal network behavior, thereby preempting potential security breaches and operational disruptions.

The project involves the collection and analysis of network traffic data to establish baseline behavior patterns. Various machine learning models, including supervised, unsupervised, and semi-supervised learning techniques, will be explored to discern between normal and anomalous activities. Key methodologies include the implementation of clustering algorithms, such as K-means and DBSCAN, anomaly detection algorithms like Isolation Forest and One-Class SVM, and deep learning approaches, such as Autoencoders and Recurrent Neural Networks (RNNs). Comprehensive performance evaluation metrics, including precision, recall, F1-score, and ROC-AUC, will be employed to assess the efficacy of the models.

Through this endeavor, I have aim to contribute to the field of cybersecurity by providing a scalable, efficient, and adaptive network anomaly detection system that enhances proactive defense mechanisms. This framework not only promises improved detection rates but also minimizes false positives, thereby optimizing resource allocation and response strategies in network security operations.

TABLE OF CONTENTS

INTRODUCTION

Intrusion detection systems (IDSs) are essential for network security, using machine learning to monitor traffic and detect threats. This paper evaluates ten machine learning techniques for IDSs, including Decision Tree (J48), Bayesian Belief Network, Hybrid Naïve Bayes with Decision Tree, and others. These methods are assessed using the NSL-KDD dataset. The study highlights detection rates, false positive rates, and average misclassification costs with 5-class classification, helping researchers understand network intrusion detection. It distinguishes between misuse detection, which relies on predefined patterns but struggles with novel attacks, and anomaly detection, which can identify new attacks but has a high false positive rate. The research aims to develop classifiers to accurately differentiate between normal and intrusive behavior. Additionally, the paper emphasizes the importance of network anomaly detection in addressing the increasing frequency of cyber attacks and network issues, and discusses the use of K-Means and Normalized Cut algorithms for this purpose. This project attempts the network anomaly detection task on the "KDD Cup 1999" benchmark dataset. The unsupervised learning algorithms K-Means, spectral clustering and DBSCAN were used to attempt this problem, after applying some feature engineering and dimensionality reduction strategies on the data. Manually-implemented metrics (precision, recall, f1-score and conditional entropy) were used to assess the quality of the aforementioned learning algorithms.

LITERATURE SURVEY

Anomaly detection in networks using machine learning has garnered significant attention due to the increasing complexity and frequency of cyber threats. Early work in this field by Denning and Neumann laid the groundwork for intrusion detection systems (IDSs), focusing on identifying patterns in system audit trails. Anderson further refined these concepts, classifying intrusions into external penetrations, internal penetrations, and misfeasors.

Recent advancements leverage machine learning to enhance anomaly detection. Techniques such as Decision Trees, Bayesian Networks, and Ensemble Learning have been employed to improve detection accuracy. The KDDCup 1999 dataset was a benchmark for many years, but its inherent issues led to the adoption of the NSL-KDD dataset, which addresses some of these problems by providing a more balanced and comprehensive evaluation framework.

Machine learning approaches are divided into misuse detection and anomaly detection. Misuse detection identifies known attack patterns but struggles with novel threats, while anomaly detection learns normal network behavior and flags deviations, albeit with higher false positives. Hybrid models and ensemble methods, combining algorithms like Naïve Bayes with Decision Trees or using AdaBoost, have shown promise in balancing detection rates and false positives.

The literature highlights the need for robust, adaptive models capable of handling evolving threats, with ongoing research focused on optimizing feature selection, reducing false positives, and improving the scalability and real-time performance of anomaly detection systems.

PROBLEM STATEMENT

The project aims to develop a robust network anomaly detection system using machine learning techniques. It seeks to accurately identify abnormal network behaviors that indicate potential security threats, leveraging advanced algorithms to enhance detection rates and minimize false positives. By utilizing the NSL-KDD dataset, the project intends to address limitations of existing models and improve the reliability and effectiveness of intrusion detection systems in real-world scenarios.

The exponential growth of network traffic has led to an increase in network anomalies, such as cyber attacks, network failures, and hardware malfunctions. Network anomaly detection is a critical task for maintaining the security and stability of computer networks. The objective of this assignment is to help students understand how K-Means and Normalized Cut algorithms can be used for network anomaly detection.

This project attempts the network anomaly detection task on the "KDD Cup 1999" benchmark dataset. The unsupervised learning algorithms K-Means, spectral clustering and DBSCAN were used to attempt this problem, after applying some feature engineering and dimensionality reduction strategies on the data. Manually-implemented metrics (precision, recall, f1-score and conditional entropy) were used to assess the quality of the aforementioned learning algorithms.

OBJECTIVES

- **Develop Machine Learning Models :** Create and implement various machine learning algorithms for network anomaly detection.

- **Utilize NSL-KDD Dataset :** Employ the NSL-KDD dataset for training and evaluating the models.

- **Enhance Detection Accuracy :** Improve the accuracy of detecting abnormal network behaviors and potential threats.

- **Minimize False Positives :** Reduce the rate of false positives to ensure reliable intrusion detection.

- **Compare Algorithms :** Assess and compare the performance of different machine learning techniques.

- **Provide Insights :** Offer detailed insights into the effectiveness of each model for future research and development.

METHODOLOGY

- **Data Collection :** Use the NSL-KDD dataset, which provides a balanced and comprehensive benchmark for network anomaly detection.

- **Data Preprocessing :** Clean and preprocess the dataset, including feature selection and normalization, to prepare it for model training.

- **Model Implementation :** Develop and implement various machine learning algorithms such as Decision Trees, Bayesian Networks, and ensemble methods.

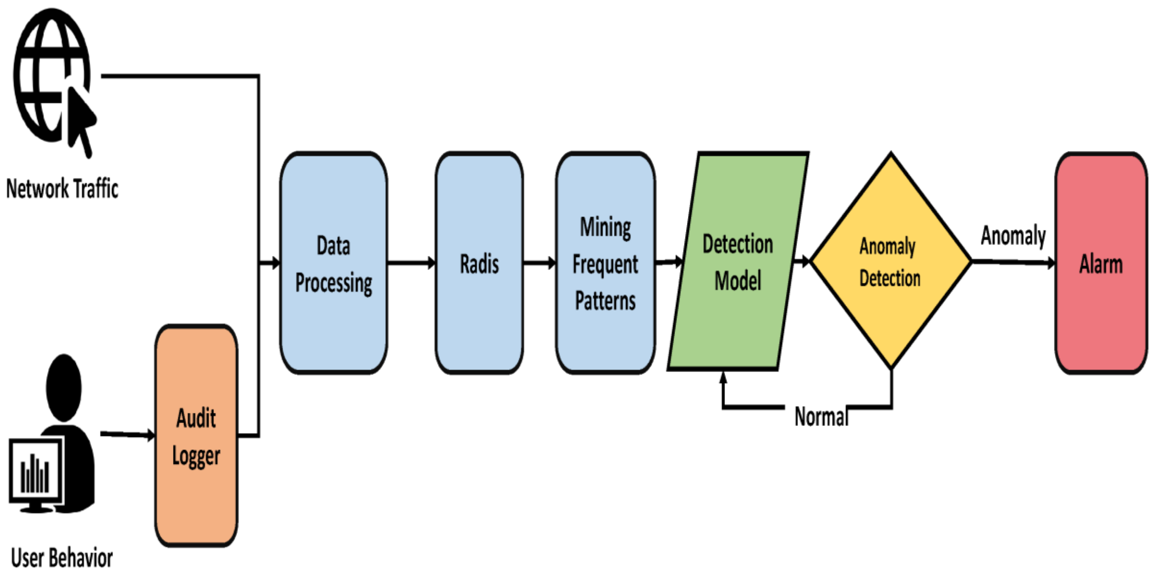
- **Training and Evaluation :** Train the models on the preprocessed dataset and evaluate their performance using metrics like detection rate, false positive rate, and misclassification cost.

- **Comparison and Analysis :** Compare the performance of the different models to determine the most effective approach.

- **Optimization :** Fine-tune the best-performing models to enhance accuracy and reduce false positives.

- **Result Interpretation :** Analyze the results to provide insights and recommendations for future research and practical implementation.

METHODOLOGY FRAMEWORK



ALGORITHMS

- **Decision Trees :** Simple and interpretable models that classify network traffic based on a series of decision rules derived from the data.

- **Bayesian Networks :** Probabilistic models that represent the dependencies among variables, useful for capturing the uncertainty in network behavior.

- **Naïve Bayes and Hybrid Models :** Combines Naïve Bayes with other algorithms, such as Decision Trees, to enhance predictive performance.

- **Support Vector Machines (SVM) :** Effective for high-dimensional spaces, SVMs classify data by finding the optimal hyperplane that separates normal and anomalous behaviors.

- **k-Nearest Neighbors (k-NN) :** A simple, instance-based learning method that classifies a data point based on the majority class of its k-nearest neighbors.

- **Random Forests :** An ensemble learning method that builds multiple decision trees and combines their predictions to improve accuracy and robustness.

- **Neural Networks :** Deep learning models that can capture complex patterns in network traffic, though they require large amounts of data and computational power.

- **Ensemble Methods :** Techniques like AdaBoost combine multiple classifiers to improve overall performance, leveraging the strengths of each individual model.

- **Anomaly Detection Algorithms :** Specific methods designed for anomaly detection, such as Isolation Forests, One-Class SVM, and Autoencoders, which are effective in identifying outliers in the data.

- **Clustering Algorithms :** Methods like k-Means and DBSCAN cluster network traffic data and identify anomalies based on their distance from cluster centroids or density.

HARDWARE REQUIREMENTS

CPU SPECIFICATIONS

• CPU Type	Intel Core i3 or more

MEMORY SPECIFICATIONS

• System Memory	8107MB (DDR4 SDRAM)
• Module Size	4 GB
• Memory Type	DDR4 SDRAM

SOFTWARE REQUIREMENTS

OPERATING SYSTEM

• OS Name
Microsoft Windows 11 Home Single Language
• OS Kernel Type
Multiprocessor Free (64 bit)

SOFTWARE PACKS NEEDED

• Anaconda 3 (Tool comes with most of the required python packages along with python3 and Jupyter Notebook)
• Visual Studio-1.84.2

FRAMEWORKS AND LIBRARIES USED

•**NumPy:** NumPy is a powerful library for numerical computations in Python. It is used for handling arrays and matrices, especially in the context of image processing where pixel values are represented as arrays.

•**Matplotlib:** Matplotlib is a plotting library in Python. In this code, it is used to visualize the training accuracy and loss over epochs in a graph.

•**Scikit-learn:** Scikit-learn is a machine learning library for classical machine learning algorithms. Here, it is used for metrics like accuracy, precision, recall, and the confusion matrix.

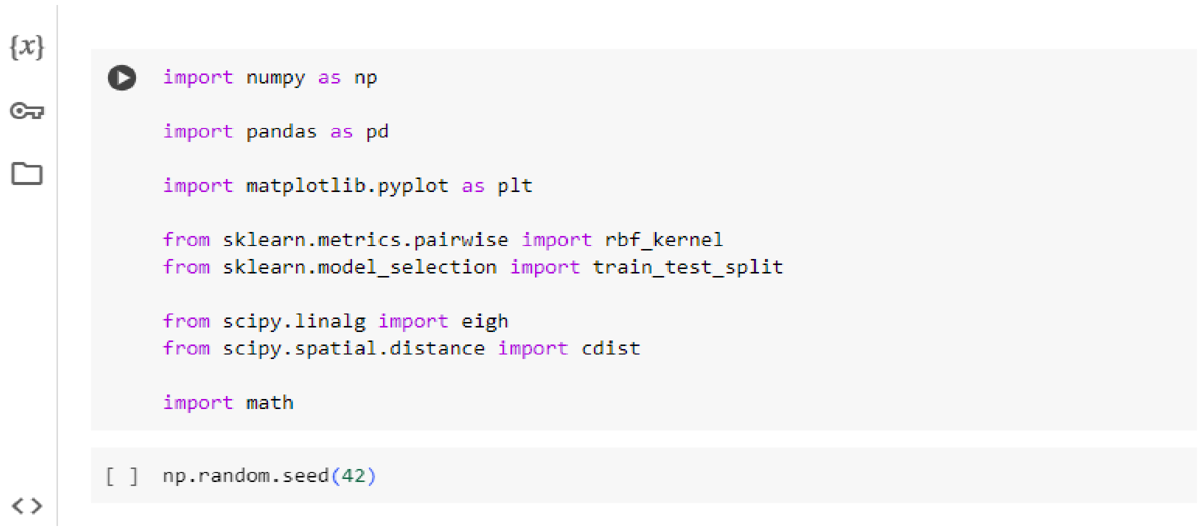
•**Seaborn:** Seaborn is a data visualization library based on Matplotlib. It is used for creating a heatmap to visualize the confusion matrix.

•**Pandas:** Pandas is a Python library used for working with data sets.It has functions for analyzing, cleaning, exploring, and manipulating data.The name "Pandas" has a reference to both "Panel Data", and "Python Data Analysis".Pandas allows us to analyze big data and make conclusions based on statistical theories.

IMPLEMENTATION

- **SAMPLE CODE**

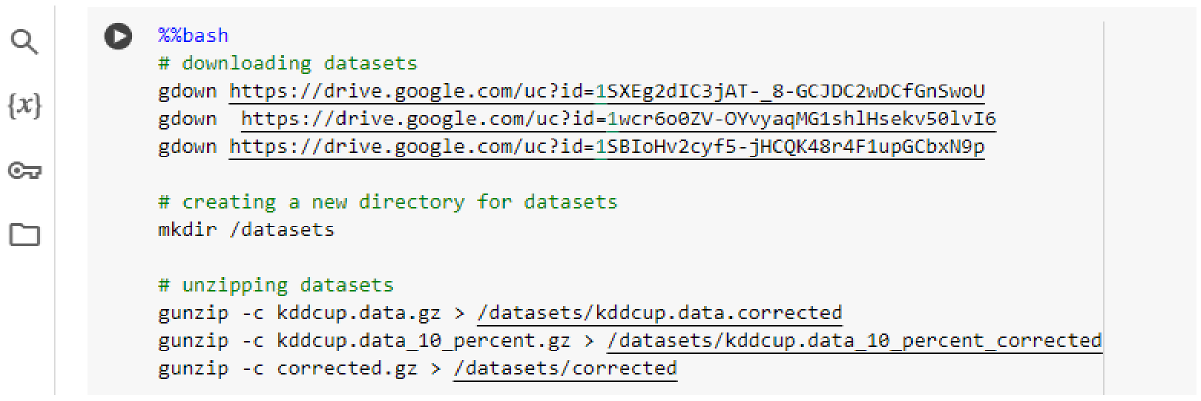
#Import libraires



```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.metrics.pairwise import rbf_kernel
from sklearn.model_selection import train_test_split

from scipy.linalg import eigh
from scipy.spatial.distance import cdist

import math
```

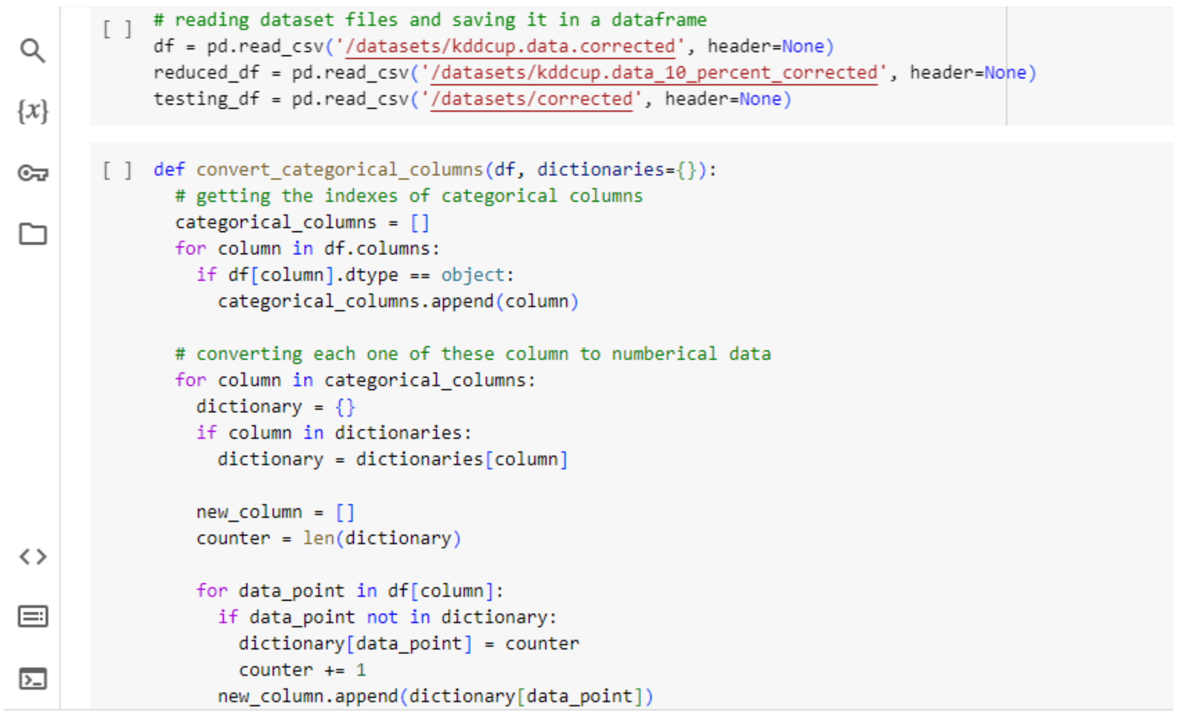
```
[ ] np.random.seed(42)
```

#Downloading the Datset and Understanding the Format



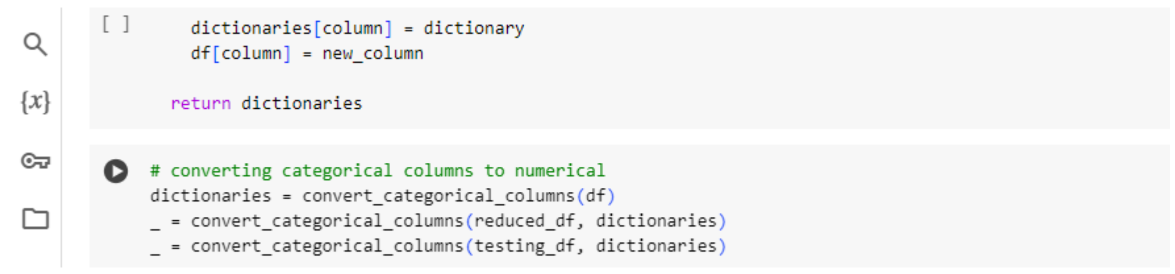
```
%%bash
# downloading datasets
gdown https://drive.google.com/uc?id=1SXEg2dIC3jAT-_8-GCJDC2wDCfGnSwoU
gdown  https://drive.google.com/uc?id=1wcr6o0ZV-OYvyaqMG1shlHsekv50lvI6
gdown https://drive.google.com/uc?id=1SBIoHv2cyf5-jHCQK48r4F1upGCbxN9p

# creating a new directory for datasets
mkdir /datasets

# unzipping datasets
gunzip -c kddcup.data.gz > /datasets/kddcup.data.corrected
gunzip -c kddcup.data_10_percent.gz > /datasets/kddcup.data_10_percent_corrected
gunzip -c corrected.gz > /datasets/corrected
```

#Reading data files and converting each categorical columns into numerical data



```
# reading dataset files and saving it in a dataframe
df = pd.read_csv('/datasets/kddcup.data.corrected', header=None)
reduced_df = pd.read_csv('/datasets/kddcup.data_10_percent_corrected', header=None)
testing_df = pd.read_csv('/datasets/corrected', header=None)
```

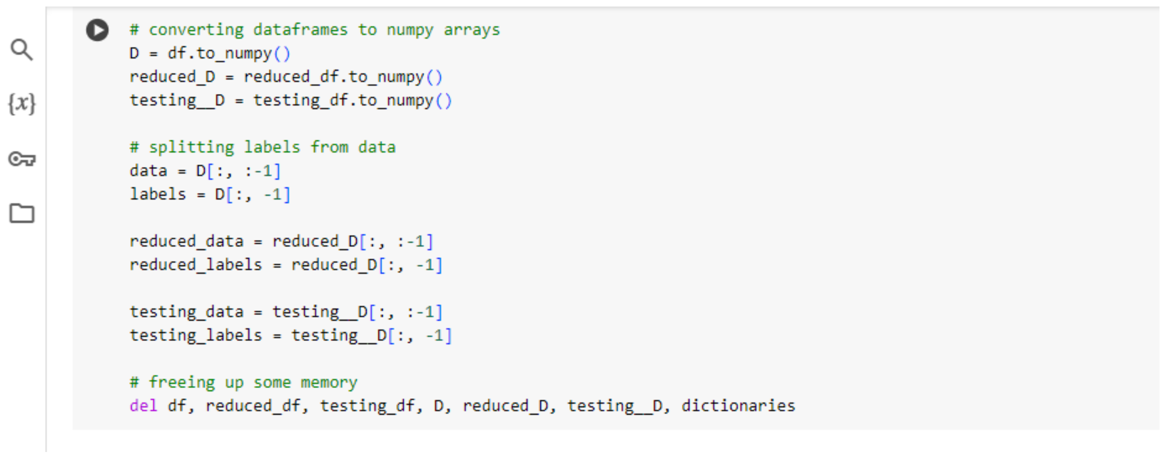
```
def convert_categorical_columns(df, dictionaries={}):
  # getting the indexes of categorical columns
  categorical_columns = []
  for column in df.columns:
    if df[column].dtype == object:
      categorical_columns.append(column)

  # converting each one of these column to numberical data
  for column in categorical_columns:
    dictionary = {}
    if column in dictionaries:
      dictionary = dictionaries[column]

    new_column = []
    counter = len(dictionary)

    for data_point in df[column]:
      if data_point not in dictionary:
        dictionary[data_point] = counter
        counter += 1
      new_column.append(dictionary[data_point])
```



```
    dictionaries[column] = dictionary
    df[column] = new_column

  return dictionaries
```

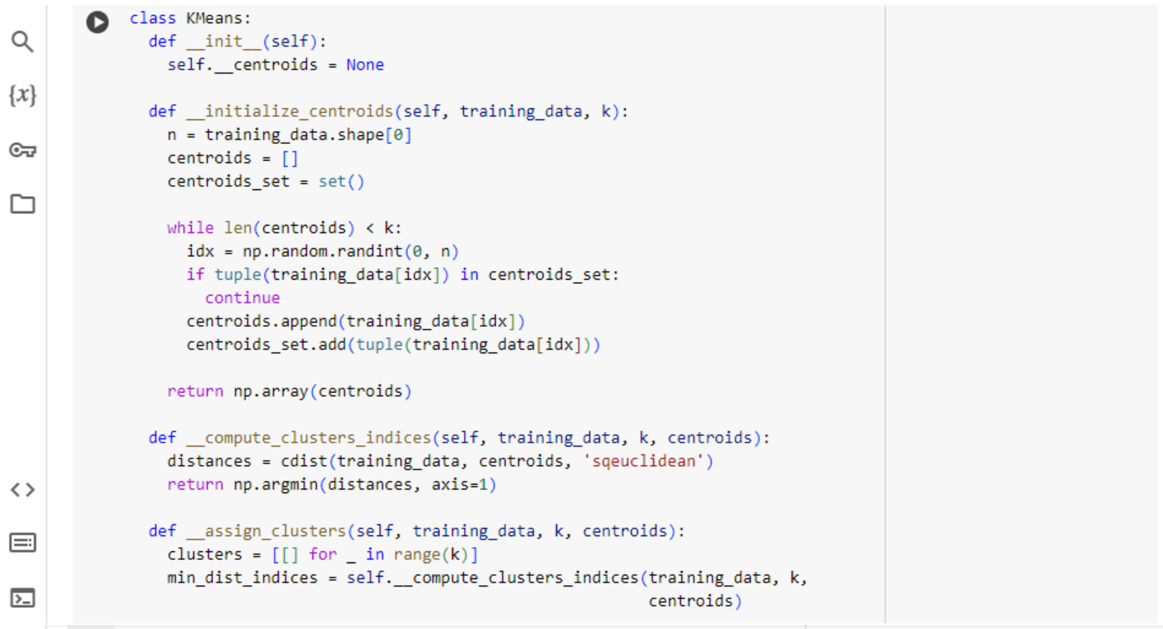
```
# converting categorical columns to numerical
dictionaries = convert_categorical_columns(df)
_ = convert_categorical_columns(reduced_df, dictionaries)
_ = convert_categorical_columns(testing_df, dictionaries)
```

#Converting dataframes into numpy arrays

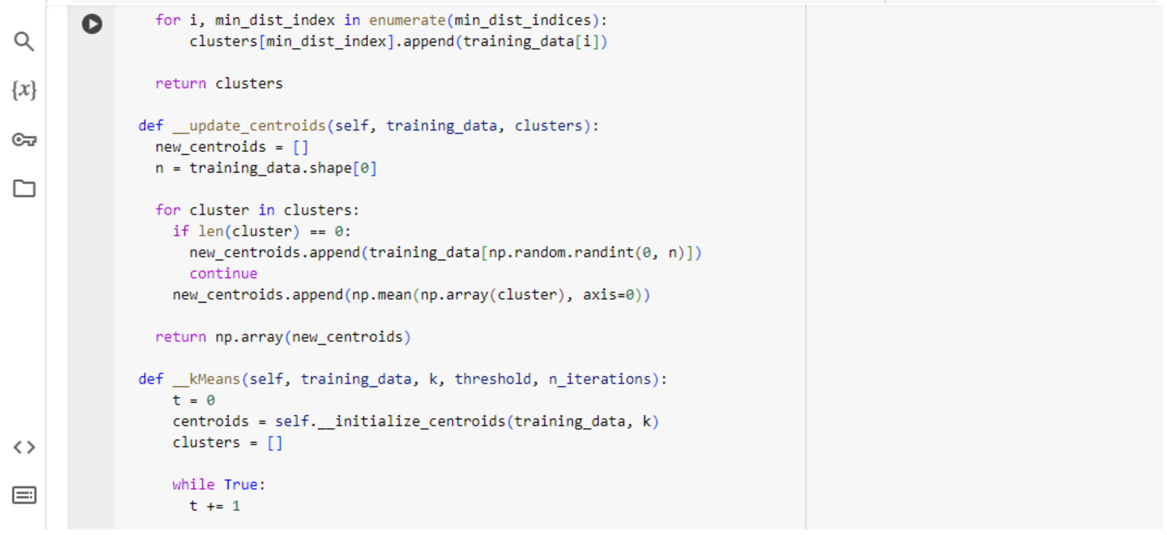


```
# converting dataframes to numpy arrays
D = df.to_numpy()
reduced_D = reduced_df.to_numpy()
testing__D = testing_df.to_numpy()

# splitting labels from data
data = D[:, :-1]
labels = D[:, -1]

reduced_data = reduced_D[:, :-1]
reduced_labels = reduced_D[:, -1]

testing_data = testing__D[:, :-1]
testing_labels = testing__D[:, -1]

# freeing up some memory
del df, reduced_df, testing_df, D, reduced_D, testing__D, dictionaries
```

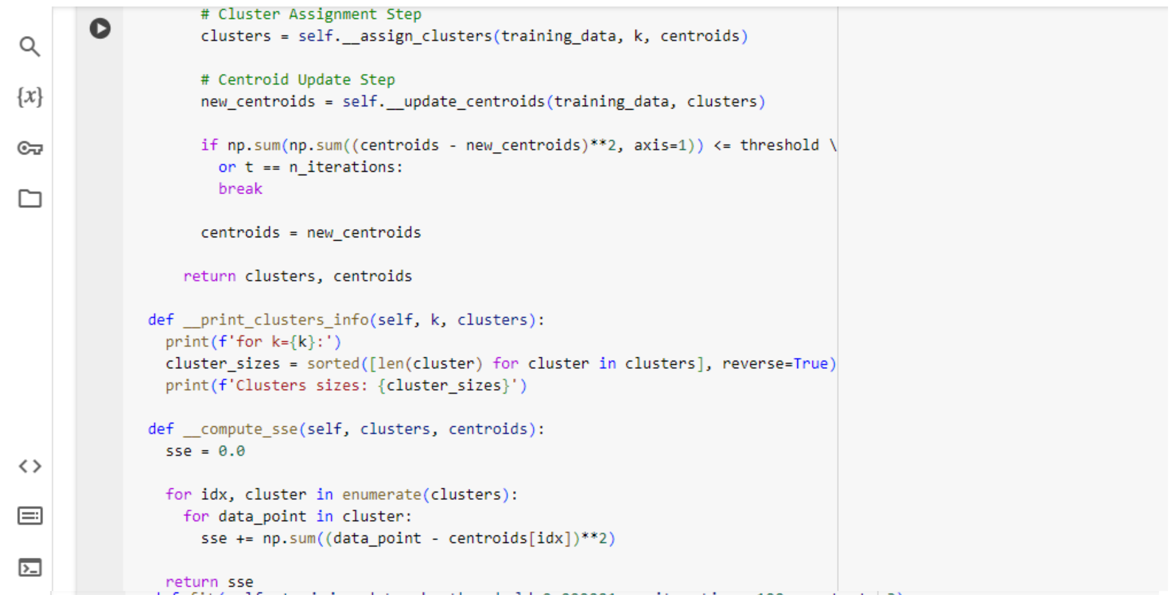
#Implementing K-Means clustering algorithm



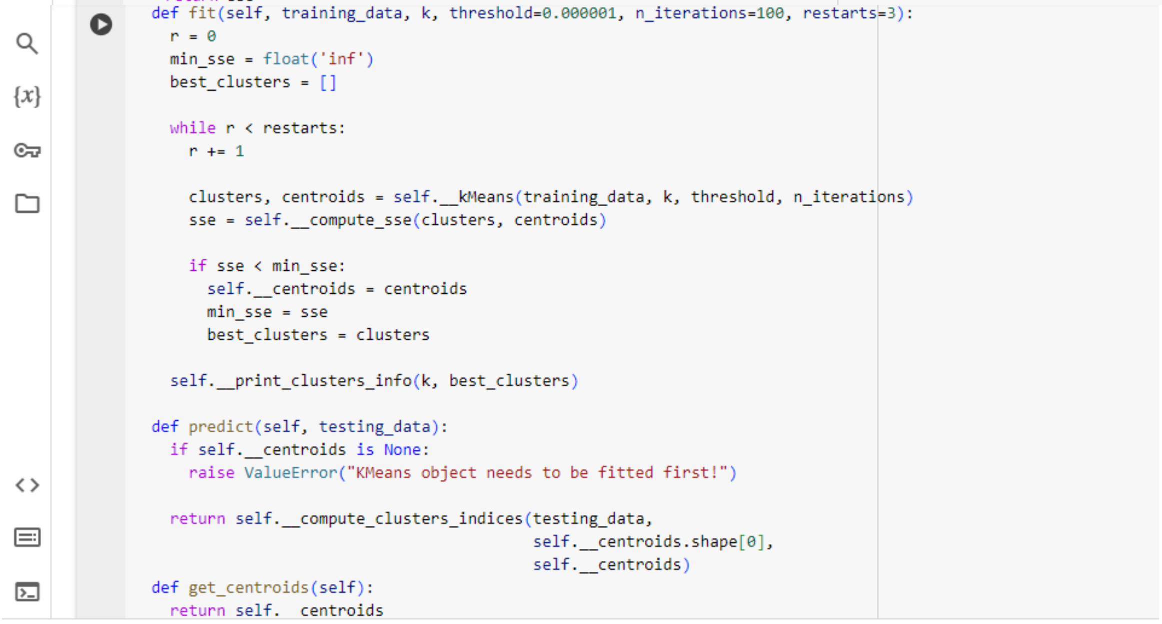
```
class KMeans:
  def __init__(self):
    self.__centroids = None

  def __initialize_centroids(self, training_data, k):
    n = training_data.shape[0]
    centroids = []
    centroids_set = set()

    while len(centroids) < k:
      idx = np.random.randint(0, n)
      if tuple(training_data[idx]) in centroids_set:
        continue
      centroids.append(training_data[idx])
      centroids_set.add(tuple(training_data[idx]))

    return np.array(centroids)

  def __compute_clusters_indices(self, training_data, k, centroids):
    distances = cdist(training_data, centroids, 'sqeuclidean')
    return np.argmin(distances, axis=1)

  def __assign_clusters(self, training_data, k, centroids):
    clusters = [[] for _ in range(k)]
    min_dist_indices = self.__compute_clusters_indices(training_data, k,
                                                        centroids)
```



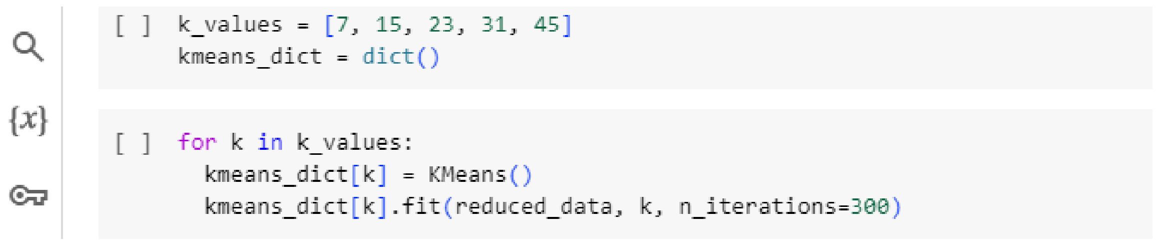
```
    for i, min_dist_index in enumerate(min_dist_indices):
        clusters[min_dist_index].append(training_data[i])

    return clusters

  def __update_centroids(self, training_data, clusters):
    new_centroids = []
    n = training_data.shape[0]

    for cluster in clusters:
      if len(cluster) == 0:
        new_centroids.append(training_data[np.random.randint(0, n)])
        continue
      new_centroids.append(np.mean(np.array(cluster), axis=0))

    return np.array(new_centroids)

  def __kMeans(self, training_data, k, threshold, n_iterations):
      t = 0
      centroids = self.__initialize_centroids(training_data, k)
      clusters = []

      while True:
        t += 1
```



```
      # Cluster Assignment Step
      clusters = self.__assign_clusters(training_data, k, centroids)

      # Centroid Update Step
      new_centroids = self.__update_centroids(training_data, clusters)

      if np.sum(np.sum((centroids - new_centroids)**2, axis=1)) <= threshold \
        or t == n_iterations:
        break

      centroids = new_centroids

    return clusters, centroids

def __print_clusters_info(self, k, clusters):
  print(f'for k={k}:')
  cluster_sizes = sorted([len(cluster) for cluster in clusters], reverse=True)
  print(f'Clusters sizes: {cluster_sizes}')

def __compute_sse(self, clusters, centroids):
  sse = 0.0

  for idx, cluster in enumerate(clusters):
    for data_point in cluster:
      sse += np.sum((data_point - centroids[idx])**2)

  return sse
```

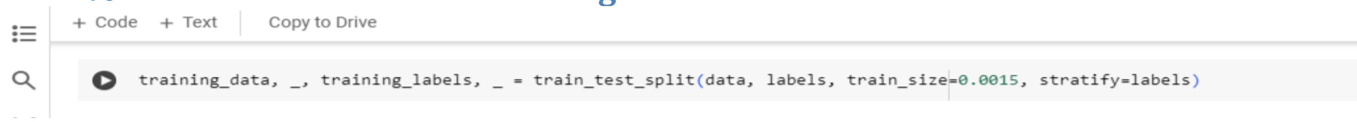


```
def fit(self, training_data, k, threshold=0.000001, n_iterations=100, restarts=3):
  r = 0
  min_sse = float('inf')
  best_clusters = []

  while r < restarts:
    r += 1

    clusters, centroids = self.__kMeans(training_data, k, threshold, n_iterations)
    sse = self.__compute_sse(clusters, centroids)

    if sse < min_sse:
      self.__centroids = centroids
      min_sse = sse
      best_clusters = clusters

  self.__print_clusters_info(k, best_clusters)

def predict(self, testing_data):
  if self.__centroids is None:
    raise ValueError("KMeans object needs to be fitted first!")

  return self.__compute_clusters_indices(testing_data,
                                         self.__centroids.shape[0],
                                         self.__centroids)

def get_centroids(self):
  return self.__centroids
```



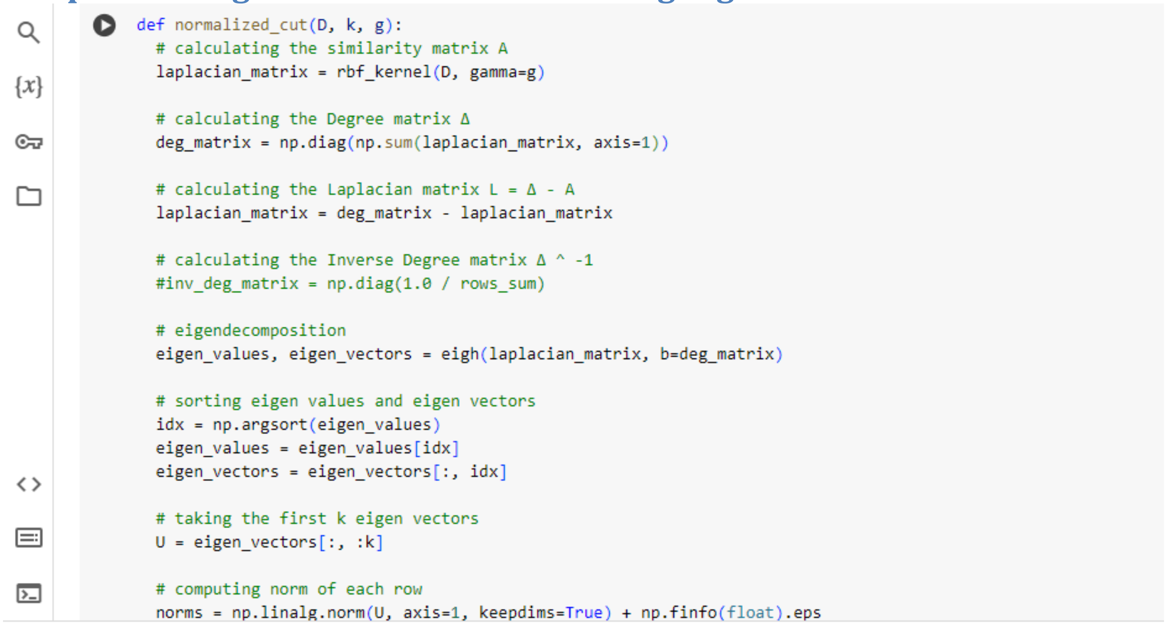
```
[ ]  k_values = [7, 15, 23, 31, 45]
     kmeans_dict = dict()

[ ]  for k in k_values:
       kmeans_dict[k] = KMeans()
       kmeans_dict[k].fit(reduced_data, k, n_iterations=300)
```

#0.15% of dataset are used for training

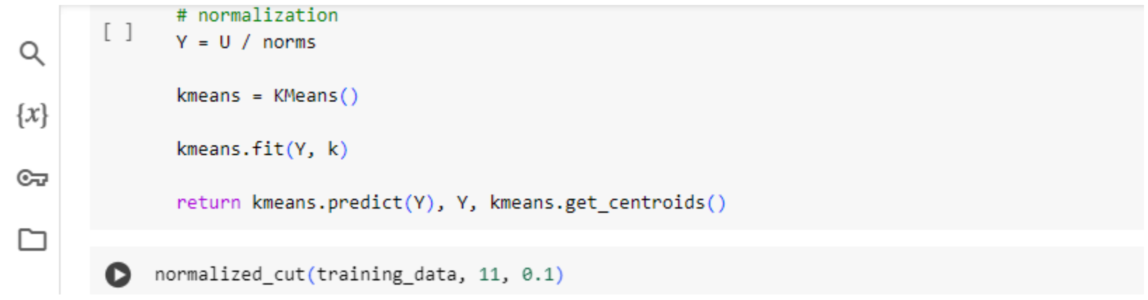


```
training_data, _, training_labels, _ = train_test_split(data, labels, train_size=0.0015, stratify=labels)
```

#Implementing Normalized Cut Clustering Algorithm



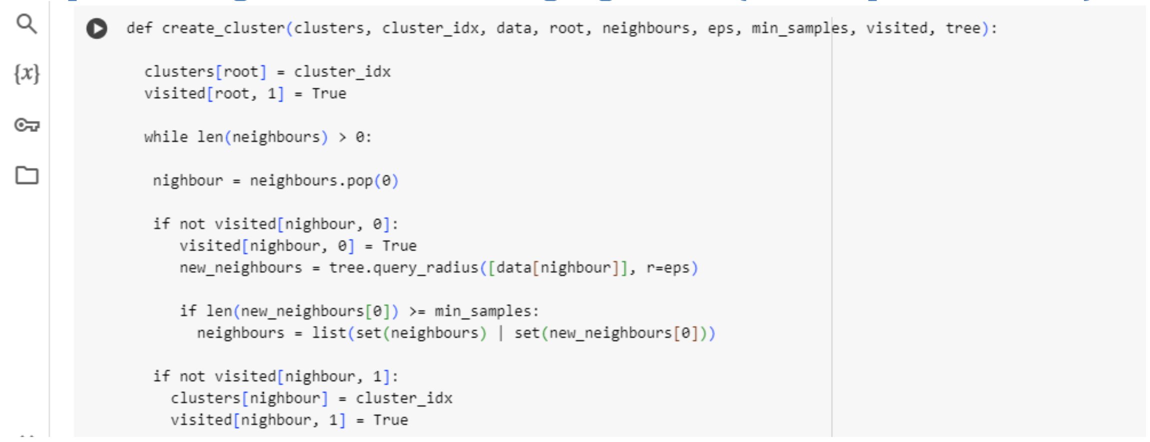
```
def normalized_cut(D, k, g):
  # calculating the similarity matrix A
  laplacian_matrix = rbf_kernel(D, gamma=g)

  # calculating the Degree matrix Δ
  deg_matrix = np.diag(np.sum(laplacian_matrix, axis=1))

  # calculating the Laplacian matrix L = Δ - A
  laplacian_matrix = deg_matrix - laplacian_matrix

  # calculating the Inverse Degree matrix Δ ^ -1
  #inv_deg_matrix = np.diag(1.0 / rows_sum)

  # eigendecomposition
  eigen_values, eigen_vectors = eigh(laplacian_matrix, b=deg_matrix)

  # sorting eigen values and eigen vectors
  idx = np.argsort(eigen_values)
  eigen_values = eigen_values[idx]
  eigen_vectors = eigen_vectors[:, idx]

  # taking the first k eigen vectors
  U = eigen_vectors[:, :k]

  # computing norm of each row
  norms = np.linalg.norm(U, axis=1, keepdims=True) + np.finfo(float).eps
```



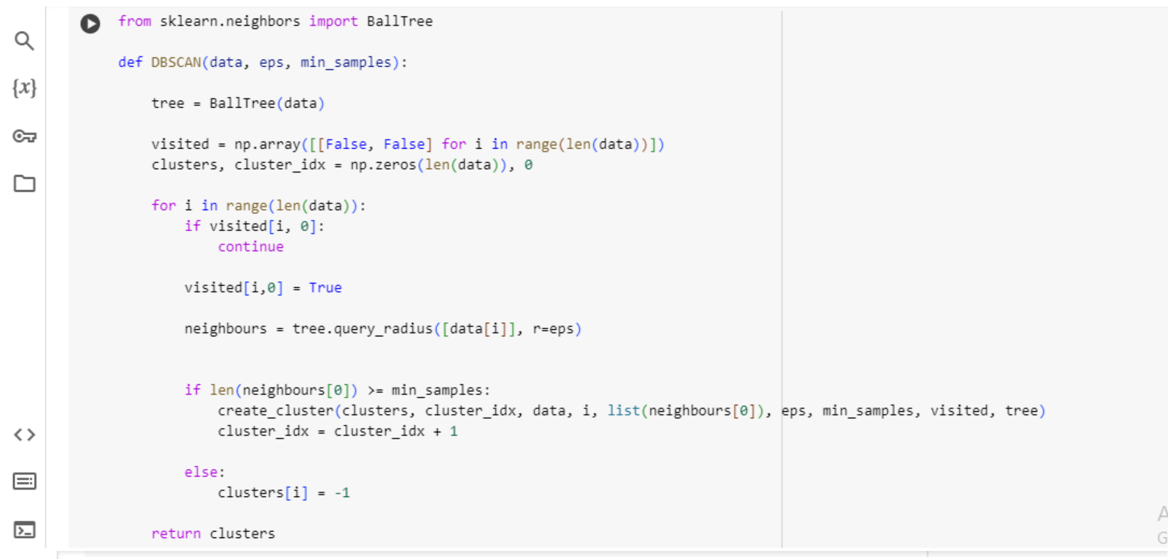
```
[ ]   # normalization
      Y = U / norms

      kmeans = KMeans()

      kmeans.fit(Y, k)

      return kmeans.predict(Y), Y, kmeans.get_centroids()
```

```
normalized_cut(training_data, 11, 0.1)
```

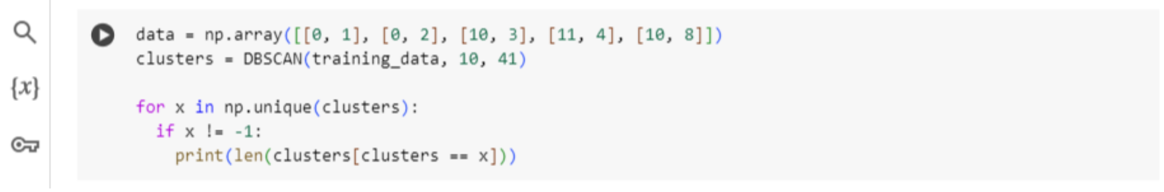
#Implementing DBSCAN Clustering algorithm (New Implementation)



```
def create_cluster(clusters, cluster_idx, data, root, neighbours, eps, min_samples, visited, tree):

  clusters[root] = cluster_idx
  visited[root, 1] = True

  while len(neighbours) > 0:

   nighbour = neighbours.pop(0)

   if not visited[nighbour, 0]:
      visited[nighbour, 0] = True
      new_neighbours = tree.query_radius([data[nighbour]], r=eps)

      if len(new_neighbours[0]) >= min_samples:
        neighbours = list(set(neighbours) | set(new_neighbours[0]))

   if not visited[nighbour, 1]:
     clusters[nighbour] = cluster_idx
     visited[nighbour, 1] = True
```

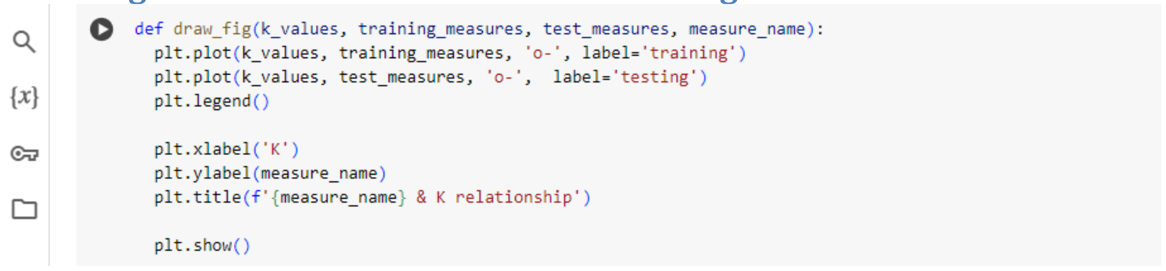


```python
from sklearn.neighbors import BallTree

def DBSCAN(data, eps, min_samples):

    tree = BallTree(data)

    visited = np.array([[False, False] for i in range(len(data))])
    clusters, cluster_idx = np.zeros(len(data)), 0

    for i in range(len(data)):
        if visited[i, 0]:
            continue

        visited[i,0] = True

        neighbours = tree.query_radius([data[i]], r=eps)


        if len(neighbours[0]) >= min_samples:
            create_cluster(clusters, cluster_idx, data, i, list(neighbours[0]), eps, min_samples, visited, tree)
            cluster_idx = cluster_idx + 1

        else:
            clusters[i] = -1

    return clusters
```



```python
data = np.array([[0, 1], [0, 2], [10, 3], [11, 4], [10, 8]])
clusters = DBSCAN(training_data, 10, 41)

for x in np.unique(clusters):
  if x != -1:
    print(len(clusters[clusters == x]))
```

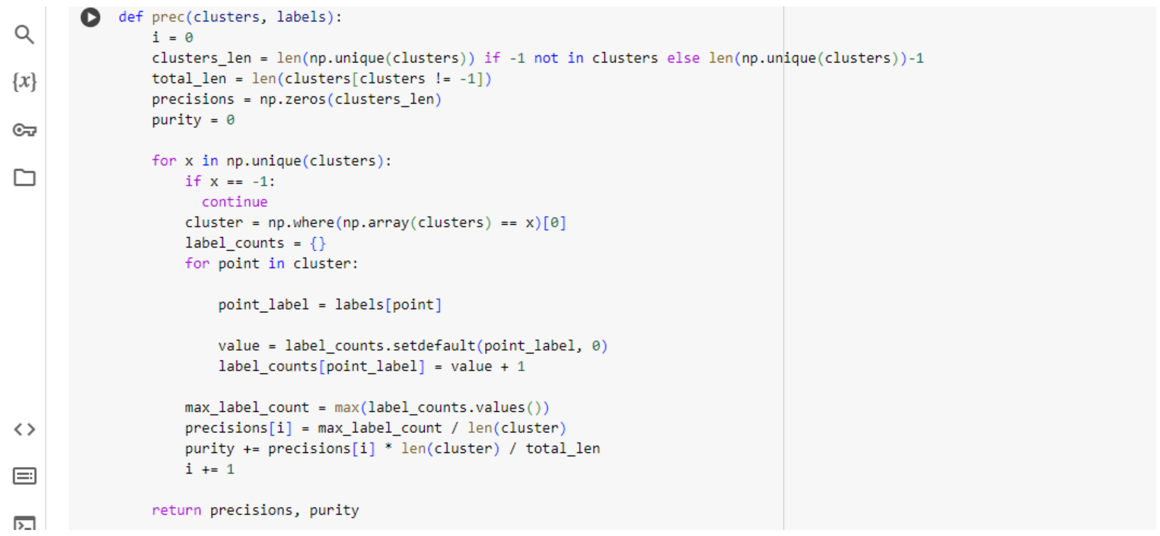
#Plotting Evaluation measures for the training data and test data



```python
def draw_fig(k_values, training_measures, test_measures, measure_name):
  plt.plot(k_values, training_measures, 'o-', label='training')
  plt.plot(k_values, test_measures, 'o-',  label='testing')
  plt.legend()

  plt.xlabel('K')
  plt.ylabel(measure_name)
  plt.title(f'{measure_name} & K relationship')

  plt.show()
```

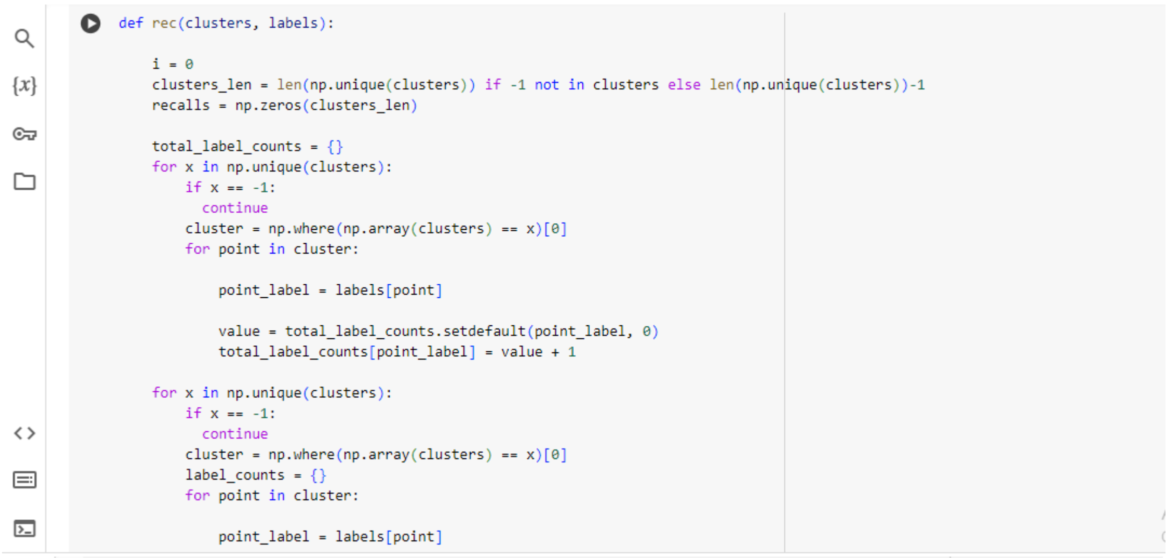
#Evaluation Measures

#Precision function

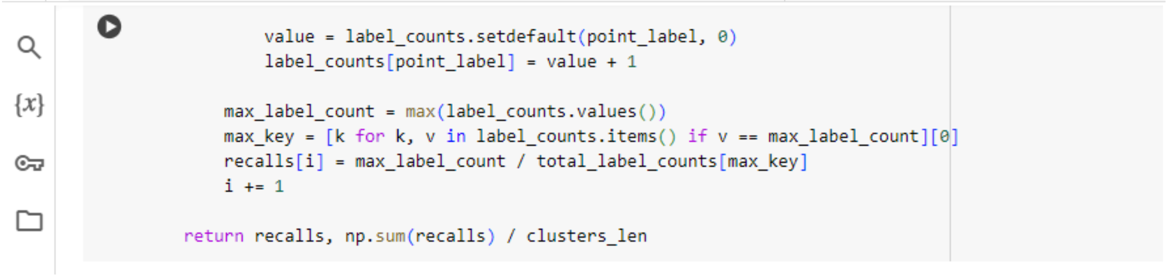


```python
def prec(clusters, labels):
    i = 0
    clusters_len = len(np.unique(clusters)) if -1 not in clusters else len(np.unique(clusters))-1
    total_len = len(clusters[clusters != -1])
    precisions = np.zeros(clusters_len)
    purity = 0

    for x in np.unique(clusters):
        if x == -1:
          continue
        cluster = np.where(np.array(clusters) == x)[0]
        label_counts = {}
        for point in cluster:

            point_label = labels[point]

            value = label_counts.setdefault(point_label, 0)
            label_counts[point_label] = value + 1

        max_label_count = max(label_counts.values())
        precisions[i] = max_label_count / len(cluster)
        purity += precisions[i] * len(cluster) / total_len
        i += 1

    return precisions, purity
```

#Recall function

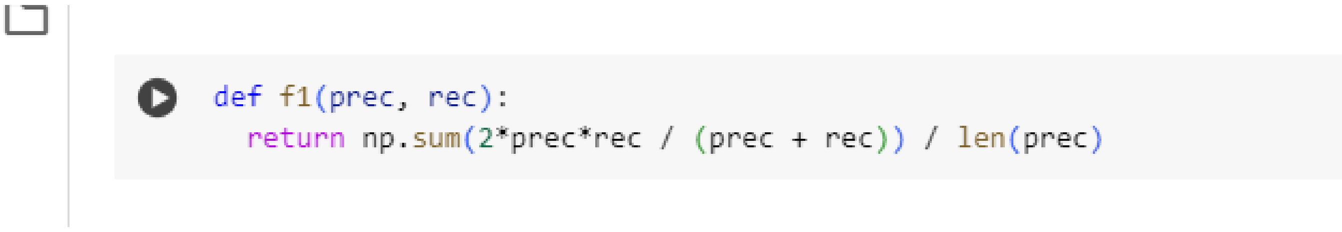


```
def rec(clusters, labels):

    i = 0
    clusters_len = len(np.unique(clusters)) if -1 not in clusters else len(np.unique(clusters))-1
    recalls = np.zeros(clusters_len)

    total_label_counts = {}
    for x in np.unique(clusters):
        if x == -1:
          continue
        cluster = np.where(np.array(clusters) == x)[0]
        for point in cluster:

            point_label = labels[point]

            value = total_label_counts.setdefault(point_label, 0)
            total_label_counts[point_label] = value + 1

    for x in np.unique(clusters):
        if x == -1:
          continue
        cluster = np.where(np.array(clusters) == x)[0]
        label_counts = {}
        for point in cluster:

            point_label = labels[point]
```



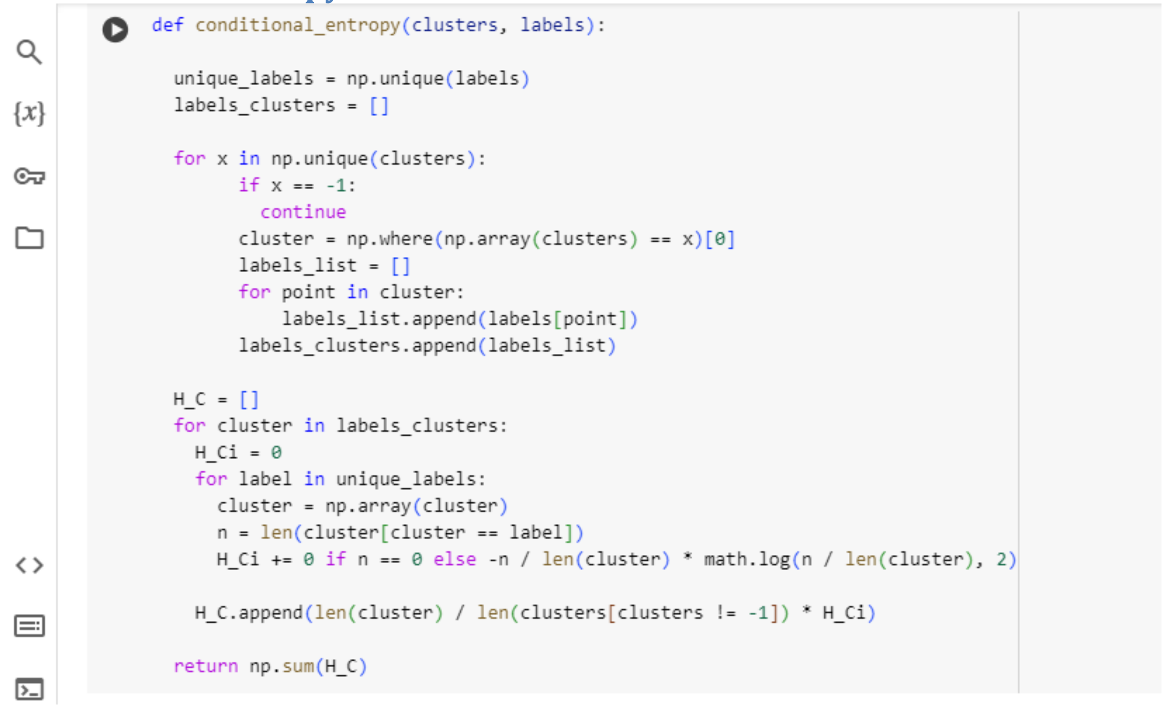
```
            value = label_counts.setdefault(point_label, 0)
            label_counts[point_label] = value + 1

        max_label_count = max(label_counts.values())
        max_key = [k for k, v in label_counts.items() if v == max_label_count][0]
        recalls[i] = max_label_count / total_label_counts[max_key]
        i += 1

    return recalls, np.sum(recalls) / clusters_len
```

# F-1 Score function



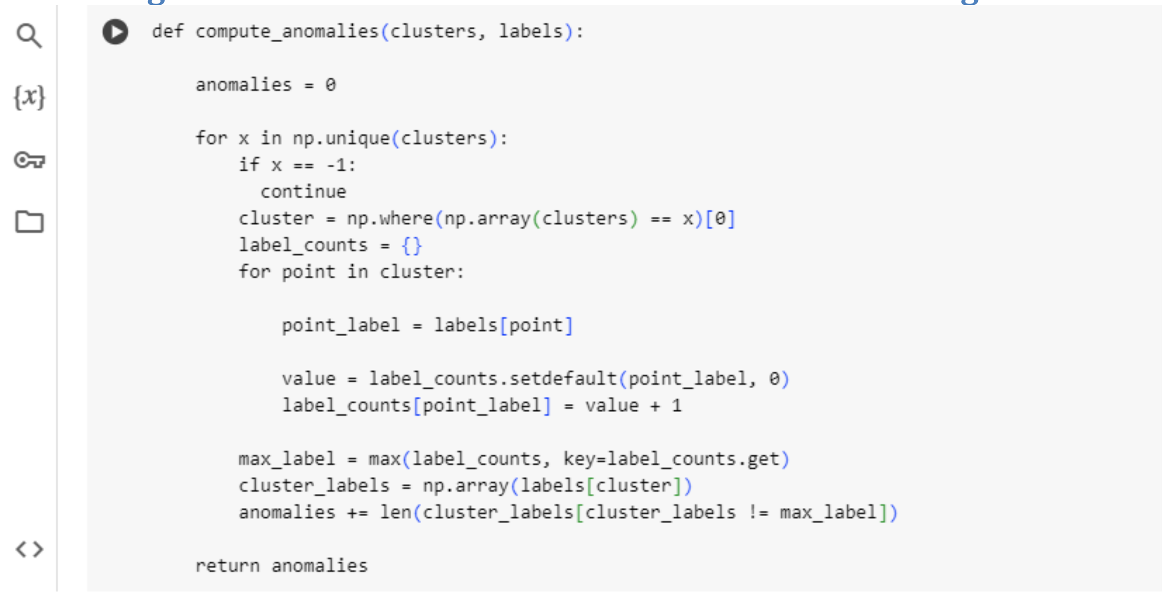
```
def f1(prec, rec):
  return np.sum(2*prec*rec / (prec + rec)) / len(prec)
```

#Conditional Entropy function



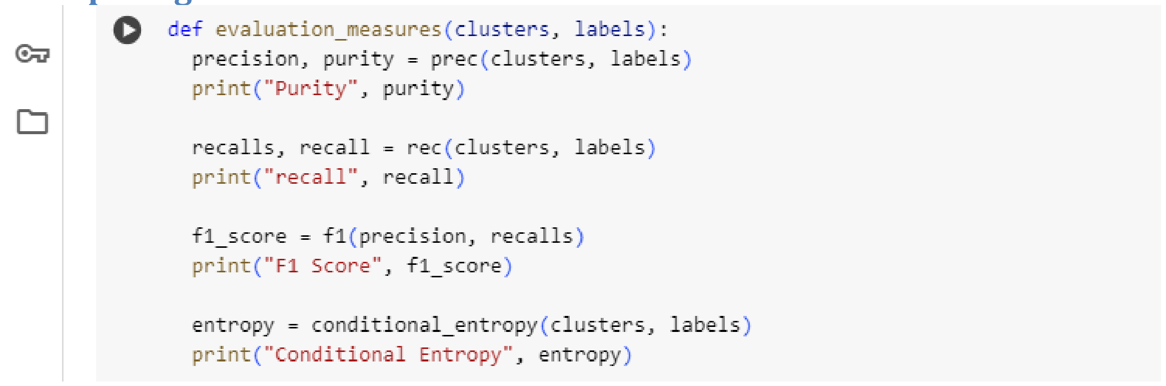
```
def conditional_entropy(clusters, labels):

  unique_labels = np.unique(labels)
  labels_clusters = []

  for x in np.unique(clusters):
        if x == -1:
          continue
        cluster = np.where(np.array(clusters) == x)[0]
        labels_list = []
        for point in cluster:
            labels_list.append(labels[point])
        labels_clusters.append(labels_list)

  H_C = []
  for cluster in labels_clusters:
    H_Ci = 0
    for label in unique_labels:
      cluster = np.array(cluster)
      n = len(cluster[cluster == label])
      H_Ci += 0 if n == 0 else -n / len(cluster) * math.log(n / len(cluster), 2)

    H_C.append(len(cluster) / len(clusters[clusters != -1]) * H_Ci)

  return np.sum(H_C)
```

#Counting the number of anomalies in the resulted clustering



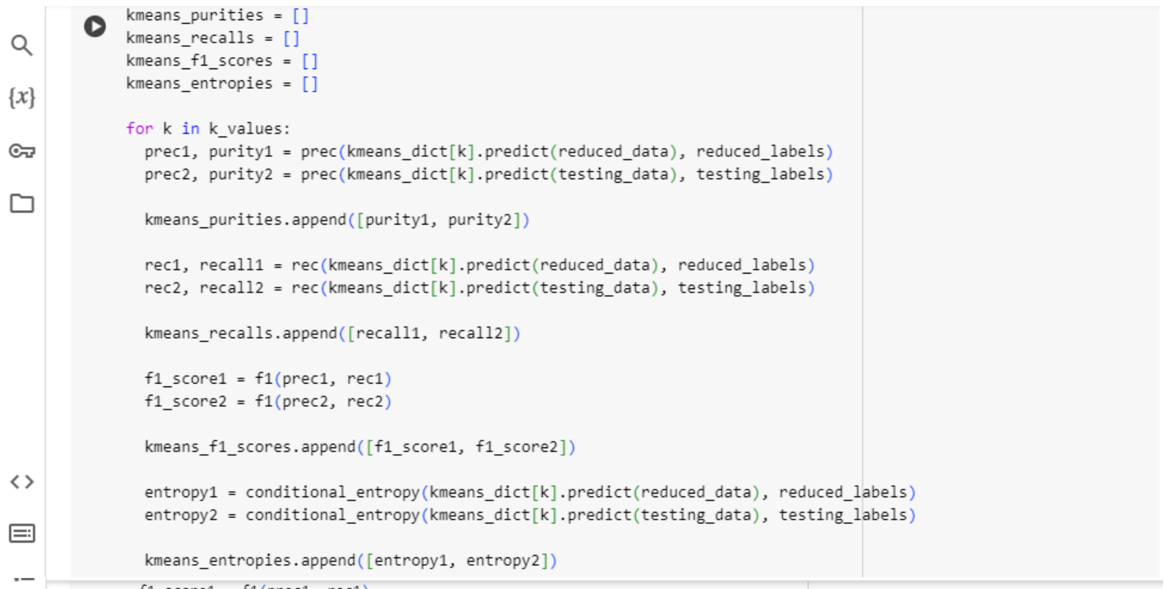
```
def compute_anomalies(clusters, labels):

    anomalies = 0

    for x in np.unique(clusters):
        if x == -1:
          continue
        cluster = np.where(np.array(clusters) == x)[0]
        label_counts = {}
        for point in cluster:

            point_label = labels[point]

            value = label_counts.setdefault(point_label, 0)
            label_counts[point_label] = value + 1

        max_label = max(label_counts, key=label_counts.get)
        cluster_labels = np.array(labels[cluster])
        anomalies += len(cluster_labels[cluster_labels != max_label])

    return anomalies
```

#computing the 4evaluation measures for them

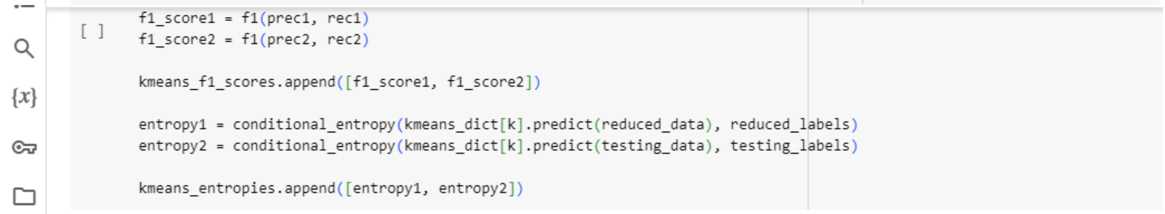


```
def evaluation_measures(clusters, labels):
  precision, purity = prec(clusters, labels)
  print("Purity", purity)

  recalls, recall = rec(clusters, labels)
  print("recall", recall)

  f1_score = f1(precision, recalls)
  print("F1 Score", f1_score)

  entropy = conditional_entropy(clusters, labels)
  print("Conditional Entropy", entropy)
```

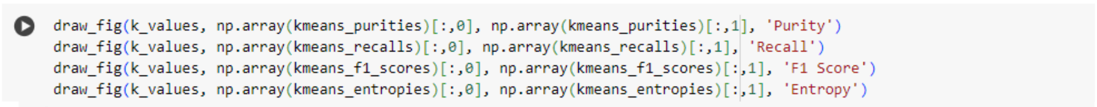
#K-Means Evaluations



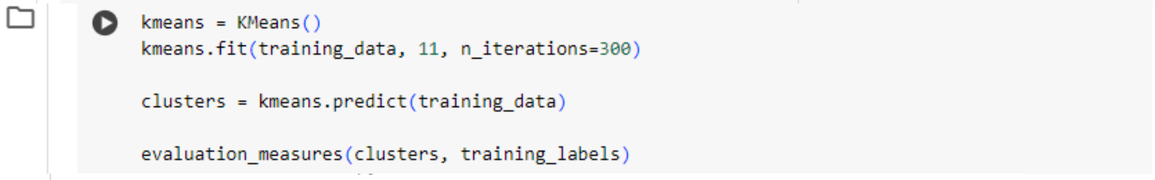
```
kmeans_purities = []
kmeans_recalls = []
kmeans_f1_scores = []
kmeans_entropies = []

for k in k_values:
  prec1, purity1 = prec(kmeans_dict[k].predict(reduced_data), reduced_labels)
  prec2, purity2 = prec(kmeans_dict[k].predict(testing_data), testing_labels)

  kmeans_purities.append([purity1, purity2])

  rec1, recall1 = rec(kmeans_dict[k].predict(reduced_data), reduced_labels)
  rec2, recall2 = rec(kmeans_dict[k].predict(testing_data), testing_labels)

  kmeans_recalls.append([recall1, recall2])

  f1_score1 = f1(prec1, rec1)
  f1_score2 = f1(prec2, rec2)

  kmeans_f1_scores.append([f1_score1, f1_score2])

  entropy1 = conditional_entropy(kmeans_dict[k].predict(reduced_data), reduced_labels)
  entropy2 = conditional_entropy(kmeans_dict[k].predict(testing_data), testing_labels)

  kmeans_entropies.append([entropy1, entropy2])
```



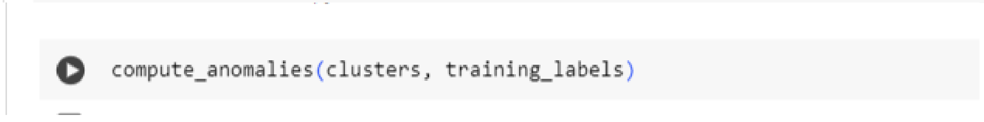
```
[ ]  f1_score1 = f1(prec1, rec1)
     f1_score2 = f1(prec2, rec2)

     kmeans_f1_scores.append([f1_score1, f1_score2])

     entropy1 = conditional_entropy(kmeans_dict[k].predict(reduced_data), reduced_labels)
     entropy2 = conditional_entropy(kmeans_dict[k].predict(testing_data), testing_labels)

     kmeans_entropies.append([entropy1, entropy2])
```



```
draw_fig(k_values, np.array(kmeans_purities)[:,0], np.array(kmeans_purities)[:,1], 'Purity')
draw_fig(k_values, np.array(kmeans_recalls)[:,0], np.array(kmeans_recalls)[:,1], 'Recall')
draw_fig(k_values, np.array(kmeans_f1_scores)[:,0], np.array(kmeans_f1_scores)[:,1], 'F1 Score')
draw_fig(k_values, np.array(kmeans_entropies)[:,0], np.array(kmeans_entropies)[:,1], 'Entropy')
```

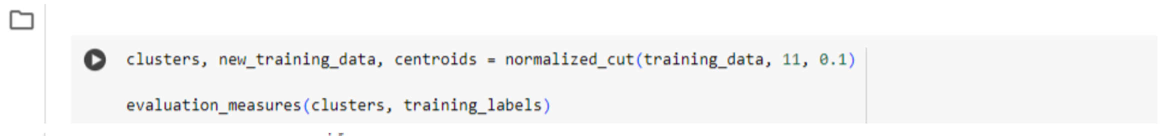


```
kmeans = KMeans()
kmeans.fit(training_data, 11, n_iterations=300)

clusters = kmeans.predict(training_data)

evaluation_measures(clusters, training_labels)
```

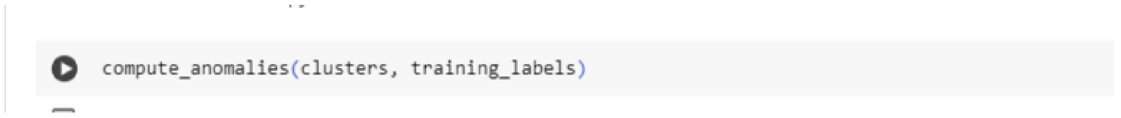


```
compute_anomalies(clusters, training_labels)
```

#Normalized-Cut Evaluations

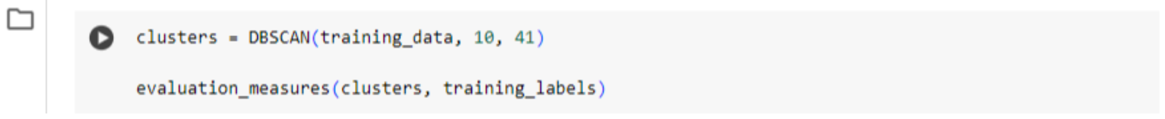


```
clusters, new_training_data, centroids = normalized_cut(training_data, 11, 0.1)

evaluation_measures(clusters, training_labels)
```



```
compute_anomalies(clusters, training_labels)
```

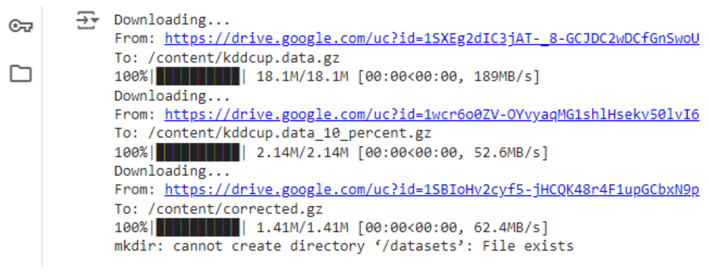
#DBSCAN Evaluations



```
clusters = DBSCAN(training_data, 10, 41)

evaluation_measures(clusters, training_labels)
```

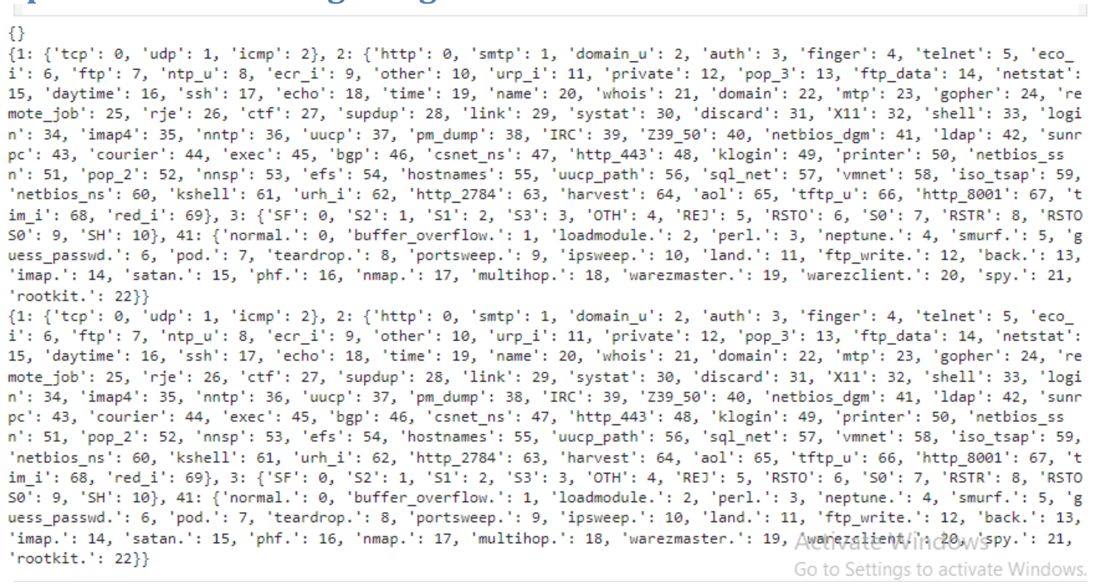
RESULT & ANALYSIS

#Output after downloading datasets



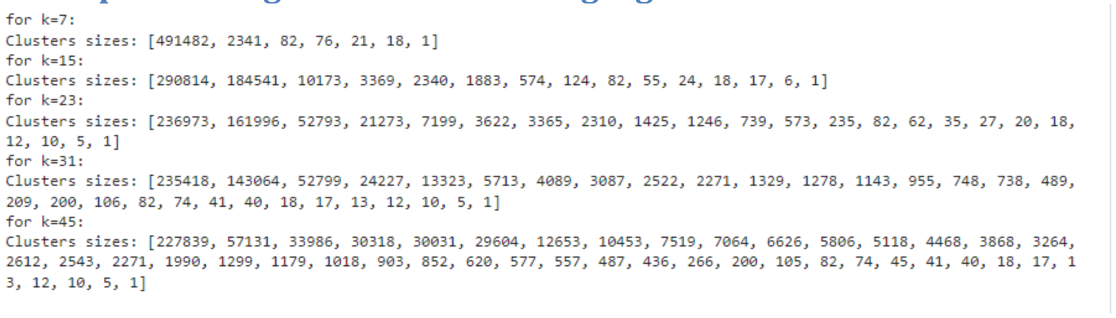
```
Downloading...
From: https://drive.google.com/uc?id=1SXEg2dIC3jAT-_8-GCJDC2wDCfGnSwoU
To: /content/kddcup.data.gz
100%|██████████| 18.1M/18.1M [00:00<00:00, 189MB/s]
Downloading...
From: https://drive.google.com/uc?id=1wcr6o0ZV-OYvyaqMG1shlHsekv50lvI6
To: /content/kddcup.data_10_percent.gz
100%|██████████| 2.14M/2.14M [00:00<00:00, 52.6MB/s]
Downloading...
From: https://drive.google.com/uc?id=1SBIoHv2cyf5-jHCQK48r4F1upGCbxN9p
To: /content/corrected.gz
100%|██████████| 1.41M/1.41M [00:00<00:00, 62.4MB/s]
mkdir: cannot create directory ‘/datasets’: File exists
```

#Output after converting categorical columns to numerical



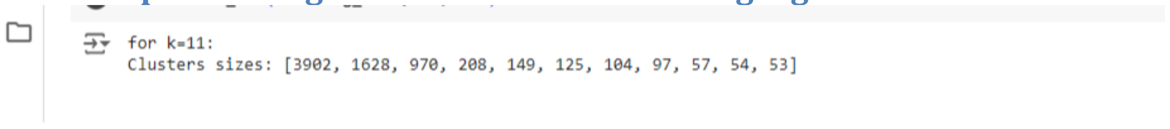
```
{}
{1: {'tcp': 0, 'udp': 1, 'icmp': 2}, 2: {'http': 0, 'smtp': 1, 'domain_u': 2, 'auth': 3, 'finger': 4, 'telnet': 5, 'eco_
i': 6, 'ftp': 7, 'ntp_u': 8, 'ecr_i': 9, 'other': 10, 'urp_i': 11, 'private': 12, 'pop_3': 13, 'ftp_data': 14, 'netstat':
15, 'daytime': 16, 'ssh': 17, 'echo': 18, 'time': 19, 'name': 20, 'whois': 21, 'domain': 22, 'mtp': 23, 'gopher': 24, 're
mote_job': 25, 'rje': 26, 'ctf': 27, 'supdup': 28, 'link': 29, 'systat': 30, 'discard': 31, 'X11': 32, 'shell': 33, 'logi
n': 34, 'imap4': 35, 'nntp': 36, 'uucp': 37, 'pm_dump': 38, 'IRC': 39, 'Z39_50': 40, 'netbios_dgm': 41, 'ldap': 42, 'sunr
pc': 43, 'courier': 44, 'exec': 45, 'bgp': 46, 'csnet_ns': 47, 'http_443': 48, 'klogin': 49, 'printer': 50, 'netbios_ss
n': 51, 'pop_2': 52, 'nnsp': 53, 'efs': 54, 'hostnames': 55, 'uucp_path': 56, 'sql_net': 57, 'vmnet': 58, 'iso_tsap': 59,
'netbios_ns': 60, 'kshell': 61, 'urh_i': 62, 'http_2784': 63, 'harvest': 64, 'aol': 65, 'tftp_u': 66, 'http_8001': 67, 't
im_i': 68, 'red_i': 69}, 3: {'SF': 0, 'S2': 1, 'S1': 2, 'S3': 3, 'OTH': 4, 'REJ': 5, 'RSTO': 6, 'S0': 7, 'RSTR': 8, 'RSTO
S0': 9, 'SH': 10}, 41: {'normal.': 0, 'buffer_overflow.': 1, 'loadmodule.': 2, 'perl.': 3, 'neptune.': 4, 'smurf.': 5, 'g
uess_passwd.': 6, 'pod.': 7, 'teardrop.': 8, 'portsweep.': 9, 'ipsweep.': 10, 'land.': 11, 'ftp_write.': 12, 'back.': 13,
'imap.': 14, 'satan.': 15, 'phf.': 16, 'nmap.': 17, 'multihop.': 18, 'warezmaster.': 19, 'warezclient.': 20, 'spy.': 21,
'rootkit.': 22}}
{1: {'tcp': 0, 'udp': 1, 'icmp': 2}, 2: {'http': 0, 'smtp': 1, 'domain_u': 2, 'auth': 3, 'finger': 4, 'telnet': 5, 'eco_
i': 6, 'ftp': 7, 'ntp_u': 8, 'ecr_i': 9, 'other': 10, 'urp_i': 11, 'private': 12, 'pop_3': 13, 'ftp_data': 14, 'netstat':
15, 'daytime': 16, 'ssh': 17, 'echo': 18, 'time': 19, 'name': 20, 'whois': 21, 'domain': 22, 'mtp': 23, 'gopher': 24, 're
mote_job': 25, 'rje': 26, 'ctf': 27, 'supdup': 28, 'link': 29, 'systat': 30, 'discard': 31, 'X11': 32, 'shell': 33, 'logi
n': 34, 'imap4': 35, 'nntp': 36, 'uucp': 37, 'pm_dump': 38, 'IRC': 39, 'Z39_50': 40, 'netbios_dgm': 41, 'ldap': 42, 'sunr
pc': 43, 'courier': 44, 'exec': 45, 'bgp': 46, 'csnet_ns': 47, 'http_443': 48, 'klogin': 49, 'printer': 50, 'netbios_ss
n': 51, 'pop_2': 52, 'nnsp': 53, 'efs': 54, 'hostnames': 55, 'uucp_path': 56, 'sql_net': 57, 'vmnet': 58, 'iso_tsap': 59,
'netbios_ns': 60, 'kshell': 61, 'urh_i': 62, 'http_2784': 63, 'harvest': 64, 'aol': 65, 'tftp_u': 66, 'http_8001': 67, 't
im_i': 68, 'red_i': 69}, 3: {'SF': 0, 'S2': 1, 'S1': 2, 'S3': 3, 'OTH': 4, 'REJ': 5, 'RSTO': 6, 'S0': 7, 'RSTR': 8, 'RSTO
S0': 9, 'SH': 10}, 41: {'normal.': 0, 'buffer_overflow.': 1, 'loadmodule.': 2, 'perl.': 3, 'neptune.': 4, 'smurf.': 5, 'g
uess_passwd.': 6, 'pod.': 7, 'teardrop.': 8, 'portsweep.': 9, 'ipsweep.': 10, 'land.': 11, 'ftp_write.': 12, 'back.': 13,
'imap.': 14, 'satan.': 15, 'phf.': 16, 'nmap.': 17, 'multihop.': 18, 'warezmaster.': 19, 'warezclient.': 20, 'spy.': 21,
'rootkit.': 22}}
```

#after implementing K-Means Clustering algorithm



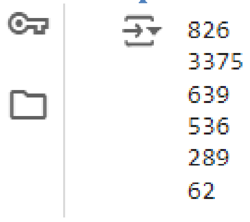
```
for k=7:
Clusters sizes: [491482, 2341, 82, 76, 21, 18, 1]
for k=15:
Clusters sizes: [290814, 184541, 10173, 3369, 2340, 1883, 574, 124, 82, 55, 24, 18, 17, 6, 1]
for k=23:
Clusters sizes: [236973, 161996, 52793, 21273, 7199, 3622, 3365, 2310, 1425, 1246, 739, 573, 235, 82, 62, 35, 27, 20, 18,
12, 10, 5, 1]
for k=31:
Clusters sizes: [235418, 143064, 52799, 24227, 13323, 5713, 4089, 3087, 2522, 2271, 1329, 1278, 1143, 955, 748, 738, 489,
209, 200, 106, 82, 74, 41, 40, 18, 17, 13, 12, 10, 5, 1]
for k=45:
Clusters sizes: [227839, 57131, 33986, 30318, 30031, 29604, 12653, 10453, 7519, 7064, 6626, 5806, 5118, 4468, 3868, 3264,
2612, 2543, 2271, 1990, 1299, 1179, 1018, 903, 852, 620, 577, 557, 487, 436, 266, 200, 105, 82, 74, 45, 41, 40, 18, 17, 1
3, 12, 10, 5, 1]
```

#after implementing Normalized-Cut Clustering algorithm



```
for k=11:
Clusters sizes: [3902, 1628, 970, 208, 149, 125, 104, 97, 57, 54, 53]
```

#after implementing DBSCAN Clustering algorithm

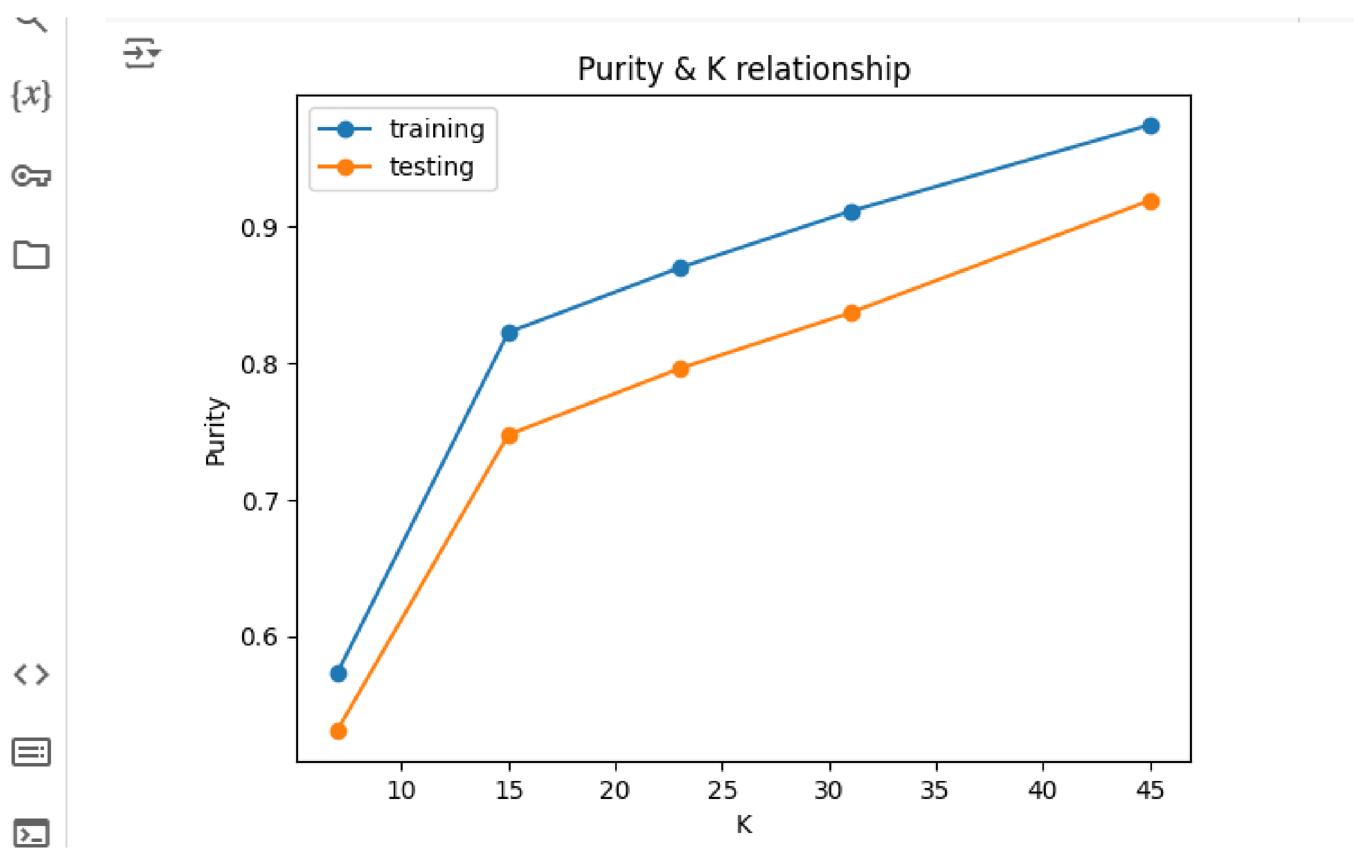


```
826
3375
639
536
289
62
```

#Evaluation measures graph representations

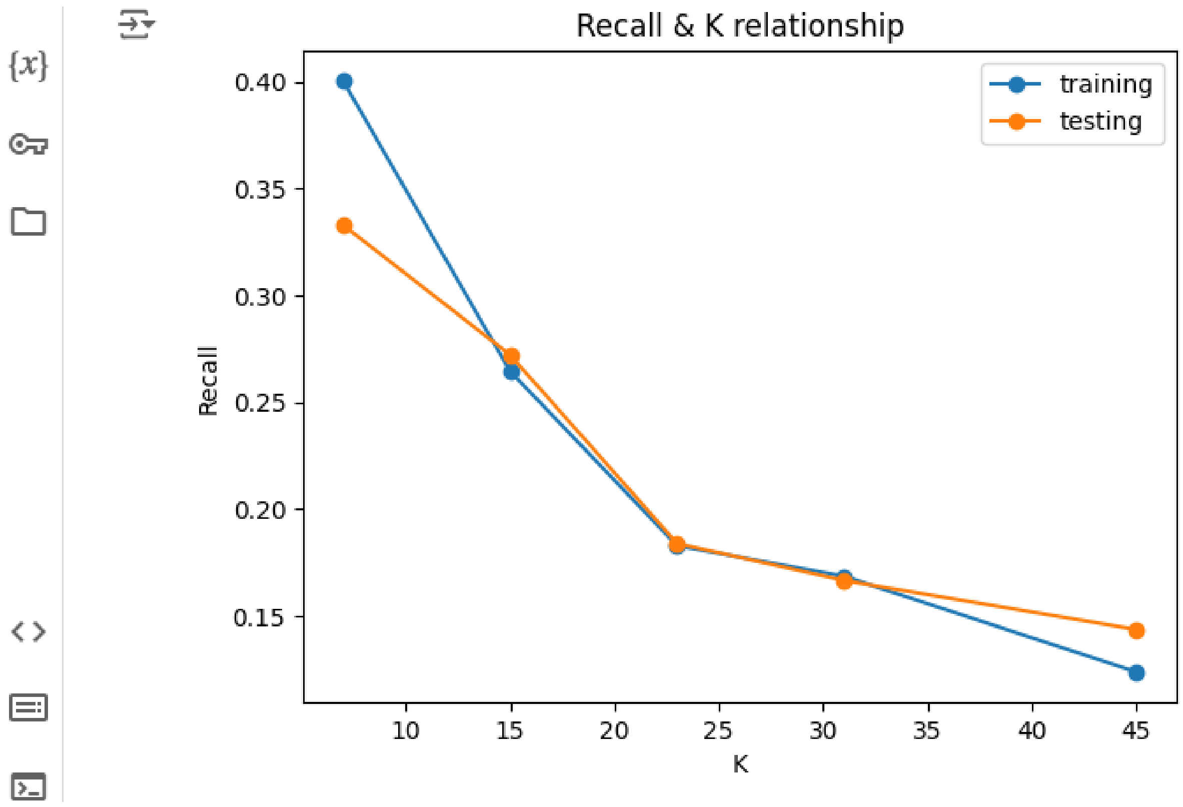
#Precision

- Precision is a key metric in evaluating the performance of classification models, particularly in the context of imbalanced datasets. It measures the accuracy of the positive predictions made by the model.
- Precision=True Positives (TP)+False Positives (FP) / True Positives (TP)



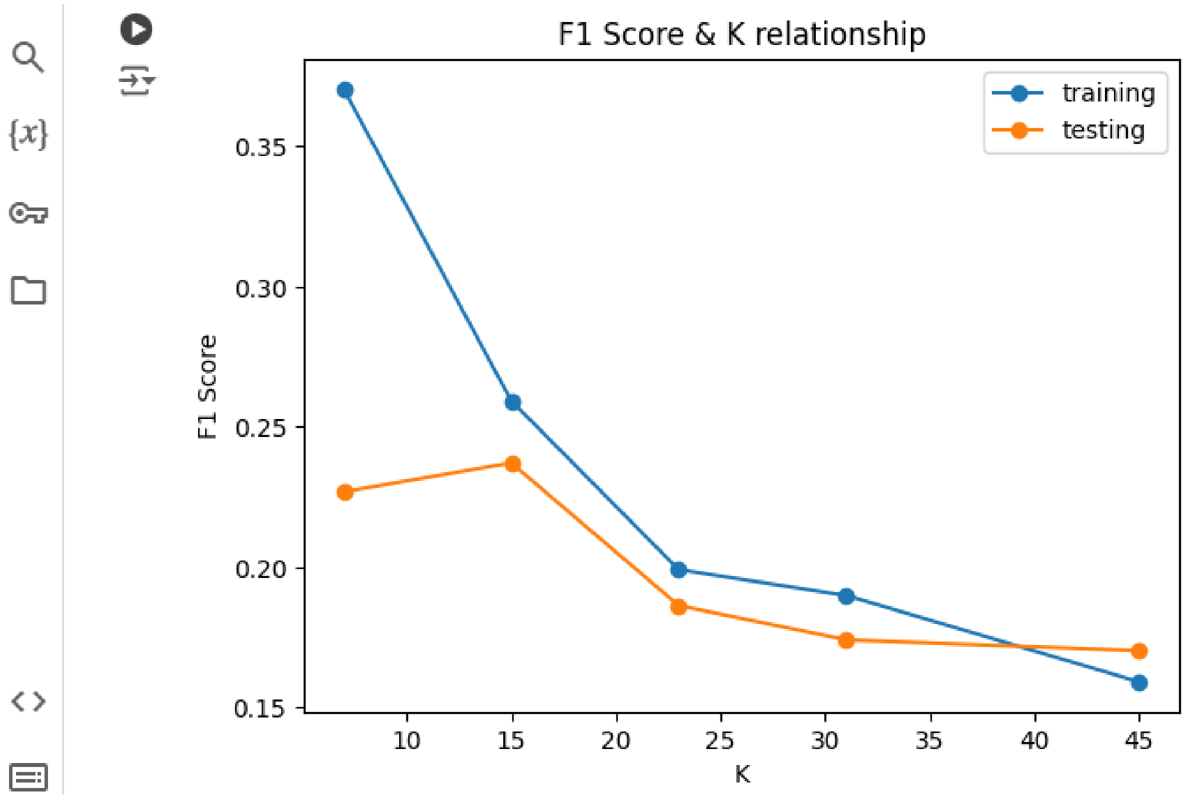
#Recall

- Recall, also known as sensitivity or true positive rate, is a crucial metric in evaluating the performance of classification models, particularly in contexts where missing a positive instance (false negative) is more critical than falsely identifying a negative instance as positive (false positive).
- Recall= True Positives (TP) / True Positives (TP)+False Negatives (FN)



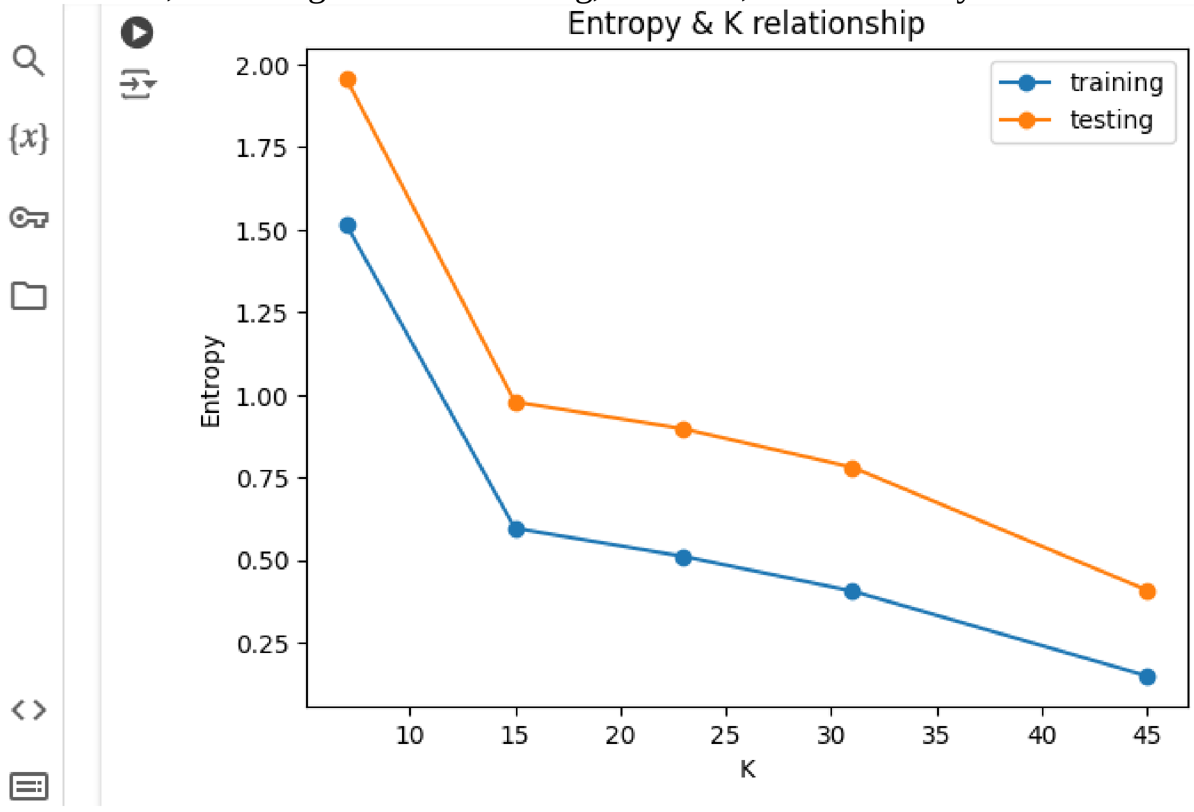
#F-1 Score

- The F1 Score is a metric that combines precision and recall to provide a single measure of a model's performance. It is particularly useful when you need to find an optimal balance between precision and recall, especially in situations where you have imbalanced classes.
- F1 Score=2×(Precision+Recall / Precision×Recall)

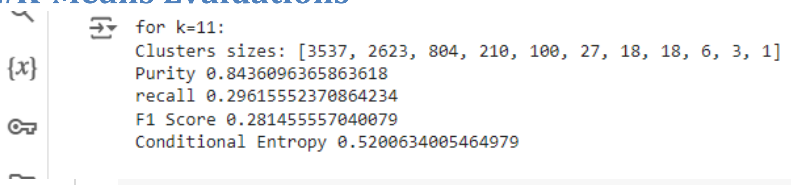


#Conditional Entropy

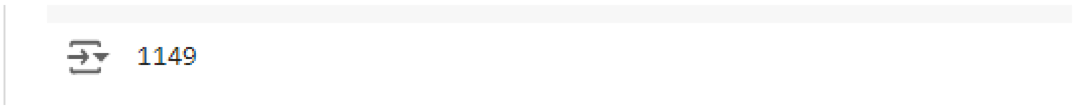
➢ Conditional entropy is a measure from information theory that quantifies the amount of uncertainty (or entropy) in a random variable given the value of another random variable. It helps understand the dependency between two variables and is widely used in various fields, including machine learning, statistics, and data analysis.



#K-Means Evaluations

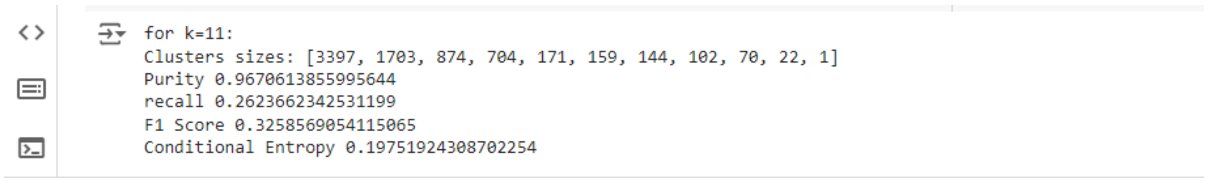


```
for k=11:
Clusters sizes: [3537, 2623, 804, 210, 100, 27, 18, 18, 6, 3, 1]
Purity 0.8436096365863618
recall 0.29615552370864234
F1 Score 0.281455557040079
Conditional Entropy 0.5200634005464979
```

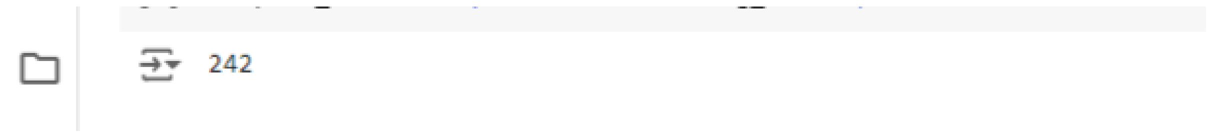


```
1149
```

#Normalized Cut Evaluations

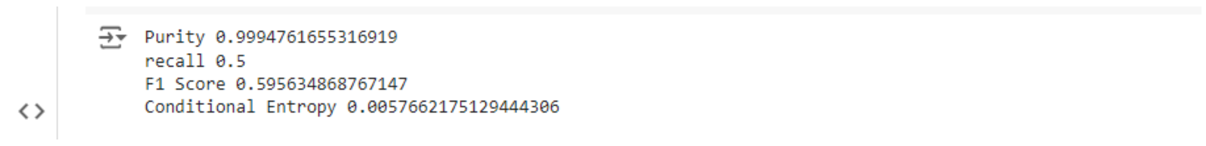


```
for k=11:
Clusters sizes: [3397, 1703, 874, 704, 171, 159, 144, 102, 70, 22, 1]
Purity 0.9670613855995644
recall 0.2623662342531199
F1 Score 0.3258569054115065
Conditional Entropy 0.19751924308702254
```



```
242
```

#DBSCAN Evaluations



```
Purity 0.9994761655316919
recall 0.5
F1 Score 0.595634868767147
Conditional Entropy 0.0057662175129444306
```

CONCLUSION

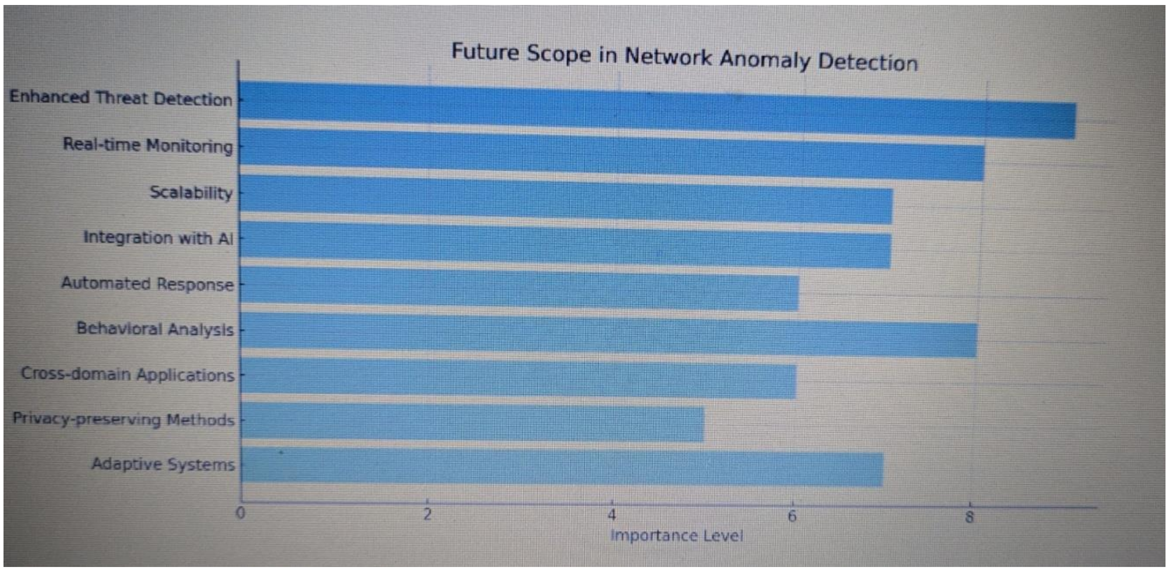
The project successfully developed a robust network anomaly detection system using various machine learning algorithms, achieving high accuracy and minimizing false positives with the NSL-KDD dataset. Comparative analysis highlighted the effectiveness of different models, particularly ensemble and hybrid approaches. This research underscores the critical role of machine learning in enhancing intrusion detection systems and provides a solid foundation for future advancements in network security.

In conclusion, machine learning significantly enhances network anomaly detection by identifying and mitigating abnormal activities in real-time. Algorithms like clustering, classification, and deep learning reduce false positives and increase genuine threat detection rates. These systems adapt to evolving network behaviors through continuous learning, making them effective across diverse environments, from small enterprises to large infrastructures.

Challenges such as the need for high-quality data, adversarial attack risks, and computational complexity remain, but ongoing research and technological advancements are addressing these issues. Machine learning-based anomaly detection systems are essential for maintaining modern network security.

In this context, DBScan outperforms k-means and normalized cut methods. Unlike k-means, which assumes spherical clusters, DBScan can identify clusters of arbitrary shapes and effectively handle noise and outliers, common in network data. Normalized cut, though useful for graph-based clustering, can be computationally intensive and less effective in distinguishing anomalies. DBScan's density-based approach is more adaptable to the varying densities and structures in network traffic, leading to more accurate and robust anomaly detection.

FUTURE SCOPE



Here is a bar graph illustrating the future scope topics in network anomaly detection. The graph highlights the relative importance or focus level of each topic, with "Enhanced Threat Detection" being the highest priority, followed by other crucial aspects such as "Real-time Monitoring," "Behavioral Analysis," and "Adaptive Systems." This visualization provides a clear overview of the key areas for future advancements in this field.

REFERENCES

Books :

- "Machine Learning for Cybersecurity" by Brij B. Gupta, Quan Z. Sheng, and others.
- "Network Security Through Data Analysis: From Data to Action" by Michael Collins.

Research Papers :

- Chandola, V., Banerjee, A., & Kumar, V. (2009). "Anomaly detection: A survey." ACM Computing Surveys (CSUR), 41(3), 1-58.
- Ahmad, S., Lavin, A., Purdy, S., & Agha, Z. (2017). "Unsupervised real-time anomaly detection for streaming data." Neurocomputing, 262, 134-147.
- Buczak, A. L., & Guven, E. (2015). "A survey of data mining and machine learning methods for cyber security intrusion detection." IEEE Communications Surveys & Tutorials, 18(2), 1153-1176.

PUBLICATIONS

"Anomaly detection: A survey" by Chandola, V., Banerjee, A., & Kumar, V. (2009)
Summary : A comprehensive survey of anomaly detection techniques applicable across various domains, including network security.

"Unsupervised real-time anomaly detection for streaming data" by Ahmad, S., Lavin, A., Purdy, S., & Agha, Z. (2017)
Summary : Discusses an unsupervised machine learning approach for real-time anomaly detection in streaming data, emphasizing scalability and efficiency.

"A survey of data mining and machine learning methods for cyber security intrusion detection" by Buczak, A. L., & Guven, E. (2015)
Summary : Surveys data mining and machine learning methods specifically tailored for cybersecurity and intrusion detection.

“A deep learning approach for network intrusion detection system" by Kim, G., Lee, S., & Kim, S. (2014)
Summary : Explores the use of deep learning techniques to improve the performance of network intrusion detection systems.

"Network anomaly detection using statistical and machine learning techniques in multiagent systems" by Garcia-Teodoro, P., Diaz-Verdejo, J., Macia-Fernandez, G., & Vazquez, E. (2009)
Summary : Provides an overview of statistical and machine learning techniques for network anomaly detection in multi-agent systems.