MACHINE LEARNING WITH PYTHON

Lab session 01:

Understanding the Python Basic

Aim:

To understand the basics of python using google collab.

Software used:

Google colabaratory

What is Colab?

Colab, or "Colaboratory", allows you to write and execute Python in your browser, with

- Zero configuration required
- Access to GPUs free of charge
- Easy sharing
- Colab notebooks allow you to combine executable code and rich text in a single document, along with images, HTML**,** LaTeX and more. When you create your own Colab notebooks, they are stored in your Google Drive account. You can easily

share your Colab notebooks with co-workers or friends, allowing them to comment on your notebooks or even edit them.

Codes

Computes a value, stores it in a variable and prints the result

```
seconds_in_a_day = 24 * 60 * 60
seconds_in_a_day
```

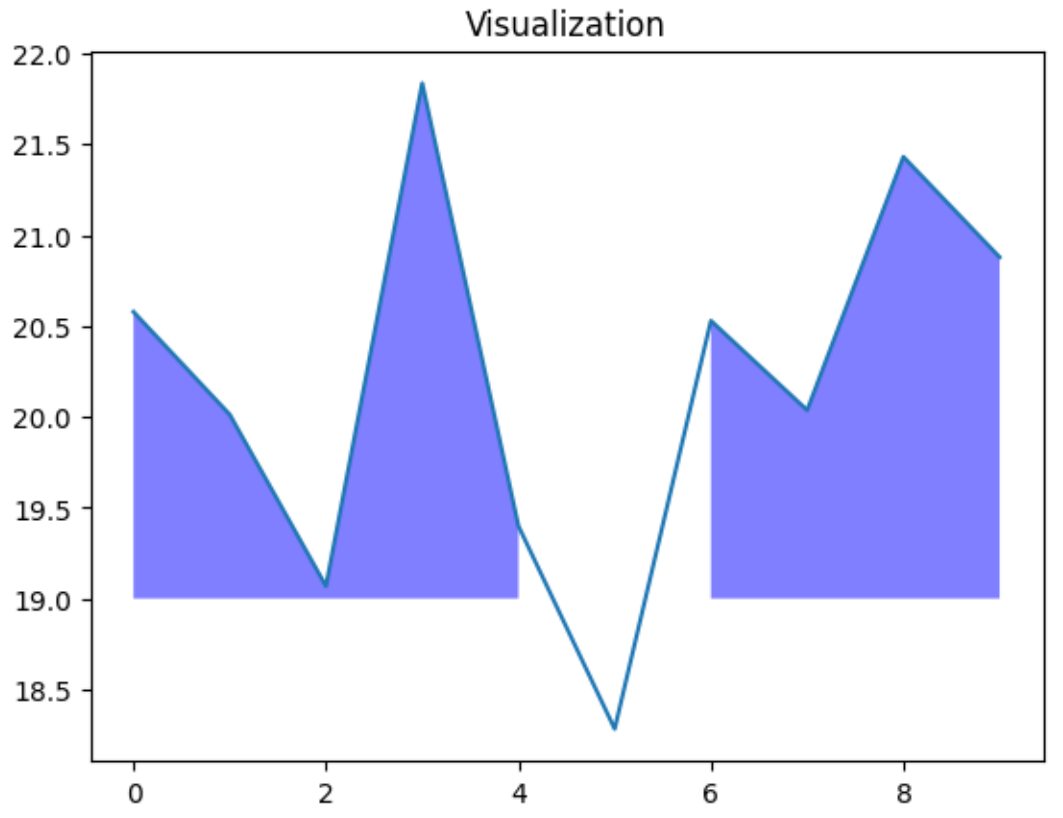
output:

```
72
```

Using the basic library to visualize data

```
import numpy as np
from matplotlib import pyplot as plt
y = 20+ np.random.randn(10)
x1 = [x1 for x1 in range(len(y))]
plt.plot(x1, y, '-')
plt.fill_between(x1, y, 19, where=(y > 19), facecolor='b', alpha=0.5)
plt.title("Visualization")
plt.show()
```

output:



PYTHON PROGRAMMING

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which

encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed

Codes

#Programs of various types

#factorial

```
n = int(input("Enter a no.: "))
f = 1
if n < 0:
   print(" no factorial")
elif n== 0:
   print("The factorial of 0 is 1")
else:
   for i in range(1,n + 1):
       f = f*i
   print("The factorial of is",f)
```

output:

```
Enter a no.: 4
The factorial of is 24
```

#Matrix Multiplication

```
a = [[1, 2, 3],
     [4, 5, 6],
     [7, 8, 9]]
b= [[9, 8, 7],
     [6, 5, 4],
     [3, 2, 1]]
m= [[0, 0, 0],
               [0, 0, 0],
               [0, 0, 0]]
for m1 in range(len(a)):
   for n in range(len(b[0])):
           for o in range(len(b)):
                m[m1][n] += a[m1][o] * b[o][n]
for r in m:
   print(r)
```

output:

```
[30, 24, 18]
[84, 69, 54]
[138, 114, 90]
```

LIBRARIES

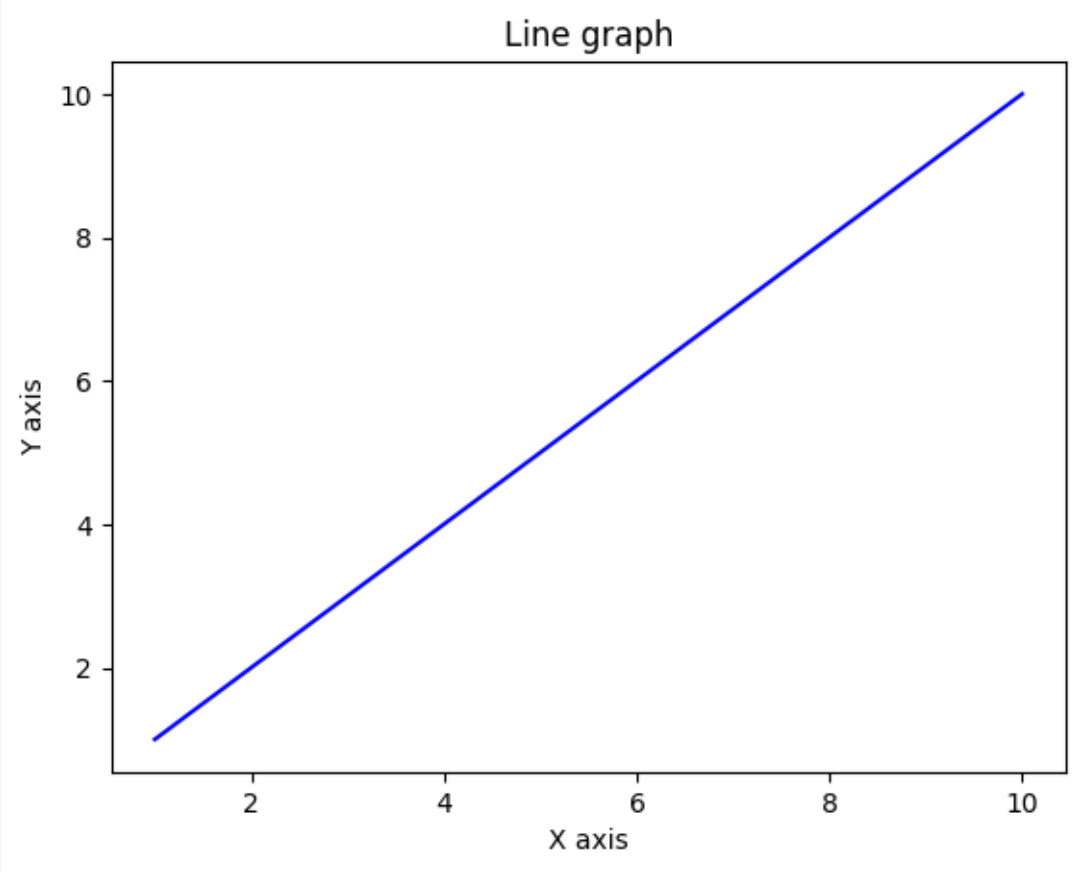
As we write large size programs in Python, we want to maintain the code's Modularity. For easy Maintenance of the code, we split the code into different parts and we can use the code later ever we need it. In Python, modules play that part. Multiple interrelated

modules are stored in a Library. And whenever we need to use a module, we import it from its library.

Codes

#Basic line graph plotting

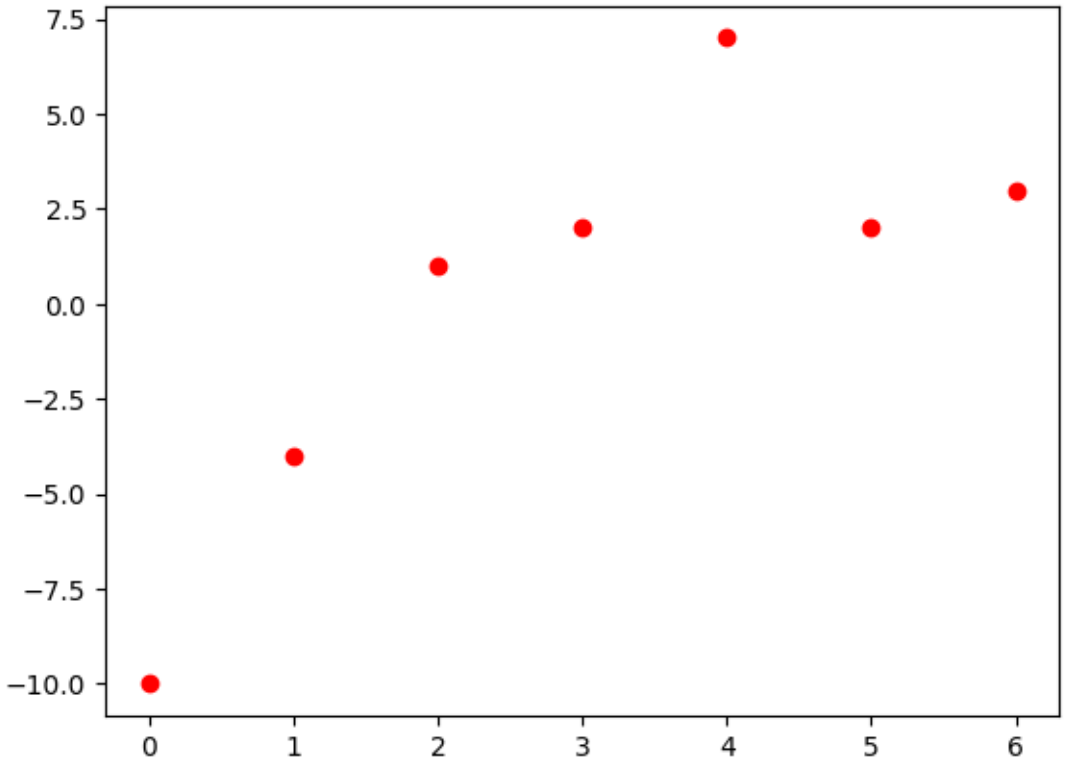
```
import numpy as np
import matplotlib.pyplot as plt
x = np.arange(1, 11)
y = np.arange(1, 11)
plt.title("Line graph")
plt.xlabel("X axis")
plt.ylabel("Y axis")
plt.plot(x, y, color ="blue")
plt.show()
output:
```



#simple dot plot

```
import matplotlib.pyplot as plt
import numpy as np
plt.plot([-10, -4, 1, 2, 7, 2, 3], "or")
plt.show()
```

output:



RESULT:

In this way Basics of Python using Google colaboratory was learned for data computations.

Lab session 02:

Analyzing Python Libraries for ML

Aim:

To understand the basic libraries of python (numpy, scipy, pandas,matplotlib.pyplot)

Software used:

Google colabaratory

Numpy

- NumPy is the fundamental package for scientific computing in Python.
- NumPy fully supports an object-oriented approach, starting, once again, with ndarray.fewer lines of code generally means fewer bugs.
- the code more closely resembles standard mathematical notation (making it easier, typically, to correctly code mathematical constructs)
- NumPy gives us the best of both worlds: element-by-element operations are the "default mode" when an ndarray is involved, but the element-by-element operation is speedily executed by pre-compiled C code

Codes

Creating array using various methods

#The Ones Method

```
Ones=np.ones(10)
Ones1=np.ones((4,5))
```

Addition of 2 numpy Arrays

```
import numpy as np
A = np.array([[6,5,4],
              [3,2,1]])
B = np.array([[1,1,1],
              [3,4,5]])
C = A + B
print(C)
```

output:

```
[[7 6 5]
 [6 6 6]]
```

Multiplying a matrix

```
import numpy as np
a = np.array([[1,2,3],
              [4,5,6]])
```

```
b = 2*a # multiplying the numpy array a(matrix) by 2
print(b)
```

Max value of Numpy Array with Float Values

```
import numpy as np
arr = np.random.rand(6).reshape(2,3)
print(arr)
#find maximum value
max_value = np.max(arr)
print('Maximum value of the array is',max_value)
```

SCIPY

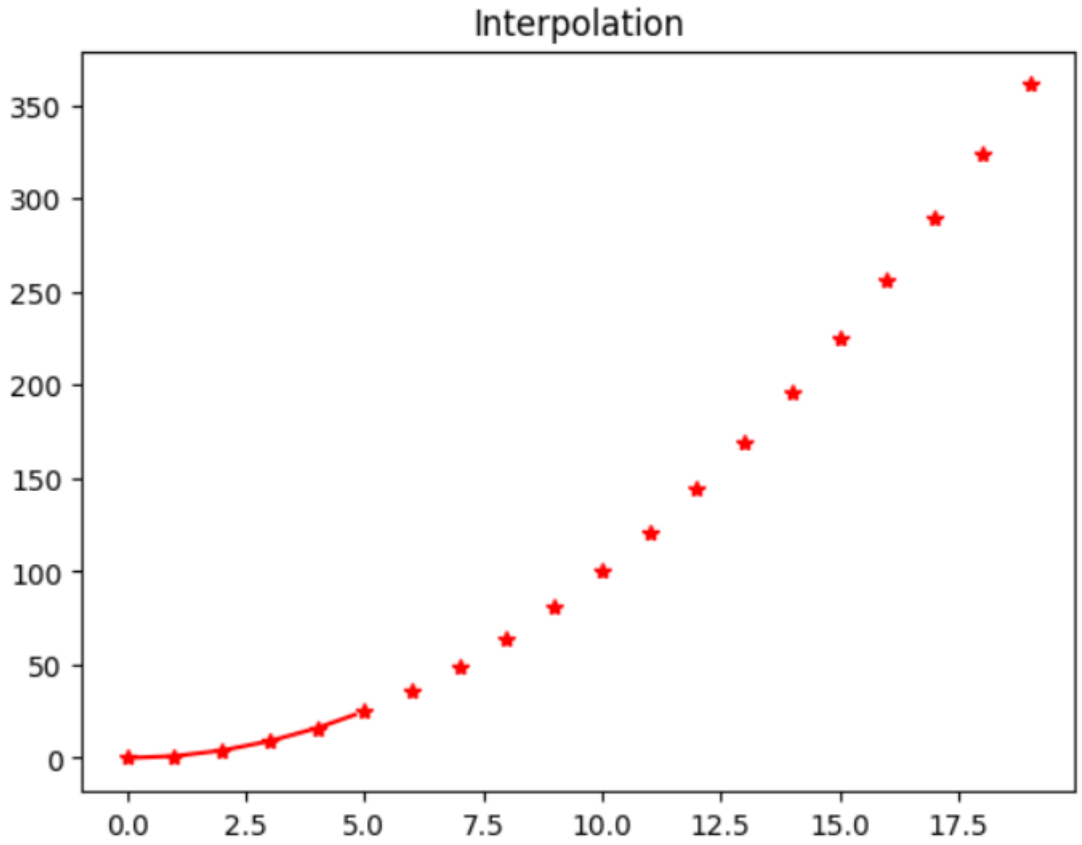
- The SciPy is an open-source scientific library of Python that is distributed under a BSD license.
- It is used to solve the complex scientific and mathematical problems.
- It is built on top of the Numpy extension, which means if we import the SciPy, there is no need to import Numpy.
- The **SciPy** library supports integration**,** gradient optimization, special functions, ordinary differential equation solvers, parallel programming tools, and many more.

Codes

#Interpolation of Eigen values and vectors

```
import matplotlib.pyplot as plt
from scipy import interpolate
import numpy as np
x = np.arange(0, 20)
y = x**2
temp = interpolate.interp1d(x, y)
xnew = np.arange(0, 5, 0.2)
ynew = temp(xnew)
plt.title("Interpolation")
plt.plot(x, y, '*', xnew, ynew, '-', color="red")
plt.show()
```

Output:-



#Calculating Eigen values and Eigen Vectors

```
from scipy import linalg #Eigen values and Eigen Vectors
import matplotlib.pyplot as plt
import numpy as np
my_arr = np.array([[5,7],[11,3]])
eg_val, eg_vect = linalg.eig(my_arr)
print("The Eigenvalues are :")
print(eg_val)
print("The Eigenvectors are :")
print(eg_vect)
plt.plot(eg_val,eg_vect)
plt.title("Matplotlib PLot scipy")
plt.xlabel("x axis")
plt.ylabel("y axis")
plt.show()
```

#Inverse of a matrix

```
import numpy as np
from scipy import linalg

# Taking a 4 * 4 matrix
A = np.array([[6, 1, 1, 3],
     [4, -2, 5, 1],
     [2, 8, 7, 6],
```

```
    [3, 1, 9, 7]])

# Calculating the inverse of the matrix
print(np.linalg.inv(A))
```

 # Find root of the equation x + cos(x):

```
from scipy.optimize import root
from math import cos
def eqn(x):
  return(x+cos(x))

myroot = root(eqn,0)
print(myroot.x)
```

Output:-

#Scatterplot of given data

```
from scipy import linalg
import numpy as np
import matplotlib.pyplot as plt
import
plt.figure(figsize=(10,5))
plt.title('Availability & Price')
plt.xlabel('Availability')
plt.ylabel('Price')
sns.scatterplot(data['availability_365'],data['price'])
plt.show()
```

Output:-

#We can create CSR matrix by passing an array into function scipy.sparse.csr_matrix().

```
import numpy as np
from scipy.sparse import csr_matrix
arr=np.array([0,0,0,0,0,1,1,0,2])
print(csr_matrix(arr))
```

PANDAS

- Pandas is a Python library used for working with data sets.
- It has functions for analyzing, cleaning, exploring, and manipulating data.
- Pandas allows us to analyze big data and make conclusions based on statistical theories.

- Pandas can clean messy data sets, and make them readable and relevant.

Codes

#To create an array from a sequence in desired data type

```
Import pandas
#create pandas array with dtype string
Pd_arr=pandas.array(data=[1,2,3,4,5],dtype=str)
```

Output:-

#Creating Dataframe Creating

```
import pandas as pd

data = {
  "calories": [420, 380, 390],
  "duration": [50, 40, 45]
}
```

Output:-

#Dataframe from dict of ndarray/lists

```
import pandas as pd

# intialise data of lists.
data = {'Name':['Tom', 'nick', 'krish', 'jack'],
        'Age':[20, 21, 19, 18]}
# Create DataFrame
df = pd.DataFrame(data)
# Print the output.
print(df)
```

Output:-

#Indexing and Selecting Data

```
import pandas as pd
# making data frame from csv file
data = pd.read_csv("nba.csv", index_col ="Name")

# retrieving columns by indexing operator
first = data["Age"]
 print(first)
```

Output:-

#***Iterating over columns***
# creating a list of dataframe columns
columns = list(df)

for i in columns:

    # printing the third element of the column
    print (df[i][2])

Output:-

MATPLOTLIB

- Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python.
- Matplotlib makes easy things easy and hard things.
- Matplotlib is extremely powerful because it allows users to create numerous and diverse plot types.
- It can be used in variety of user interfaces such as IPhython shells, python scrips, jupyter notebooks, as well as web applications and GUI toolkits.

Codes

#Simple plot

```
import matplotlib.pyplot as plt
import numpy as np
plt.figure(1)
plt.plot([1,1])
plt.figure(2)
plt.plot([1,2])
```

Output:-

#Subplot

```
import matplotlib.pyplot as plt
import numpy as np
plt.subplot(2,1,1)
plt.plot([1,4])
plt.subplot(2,1,2)
plt.plot([2,2])
```

Output:-

#Barplot

```
import matplotlib.pyplot as plt
plt.bar([0.25,1.25,2.25,3.25,4.25],[50,40,70,80,20],
label="Enfield",width=.5)
plt.bar([0.26,1.25,2.25,3.25,4.25],[80,20,20,50,60],
label="Honda", color='r',width=.5)
plt.bar([0.31,1.5,2.5,3.5,4.5],[70,20,60,40,60],
label="Yamaha", color='y',width=.5)
plt.bar([.75,1.75,2.75,3.75,4.75],[80,20,20,50,60],
label="KTM", color='g',width=.5)
plt.legend()
plt.xlabel('Days')
plt.ylabel('Distance (kms)')
plt.title('Bikes details in BAR PLOTTING')
```

Output:-

#Area Plot

```
import matplotlib.pyplot as plt
days = [1,2,3,4,5]
Enfield =[50,40,70,80,20]
Honda = [80,20,20,50,60]
Yahama =[70,20,60,40,60]
KTM = [80,20,20,50,60]
plt.plot([],[],color='k', label='Enfield', linewidth=5)
plt.plot([],[],color='c', label='Honda', linewidth=5)
plt.plot([],[],color='y', label='Yahama', linewidth=5)
plt.plot([],[],color='m', label='KTM', linewidth=5)
plt.stackplot(days, Enfield, Honda, Yahama, KTM, colors=['k','c','y','m'])
plt.xlabel('Days')
plt.ylabel('Distance in kms')
plt.title('Bikes deatils in area plot')
plt.legend()
```

Output:-

#Pie Plot

```
import matplotlib.pyplot as plt
days = [1,2,3,4,5]
Enfield =[50,40,70,80,20]
Honda = [80,20,20,50,60]
Yahama =[70,20,60,40,60]
KTM = [80,20,20,50,60]
```

```
slices = [8,5,5,6]
activities = ['Enfield','Honda','Yahama','KTM']
cols = ['c','g','y','b']
plt.pie(slices,
labels=activities,
colors=cols,
startangle=90,
shadow= True,
explode=(0,0.1,0,0),
autopct='%1.1f%%')
plt.title('Bike details in Pie Plot')
```

Output:-

RESULT:

In this way Basic libraries in python (numpy, scipy , pandas,matplotlib) is used for the clustering of data based on the centroid values.

Lab session 03:

Evaluating the k means clustering

Aim: Evaluating the kmeans clustering using python

Software Used: Google collab

K-Means Clustering:-

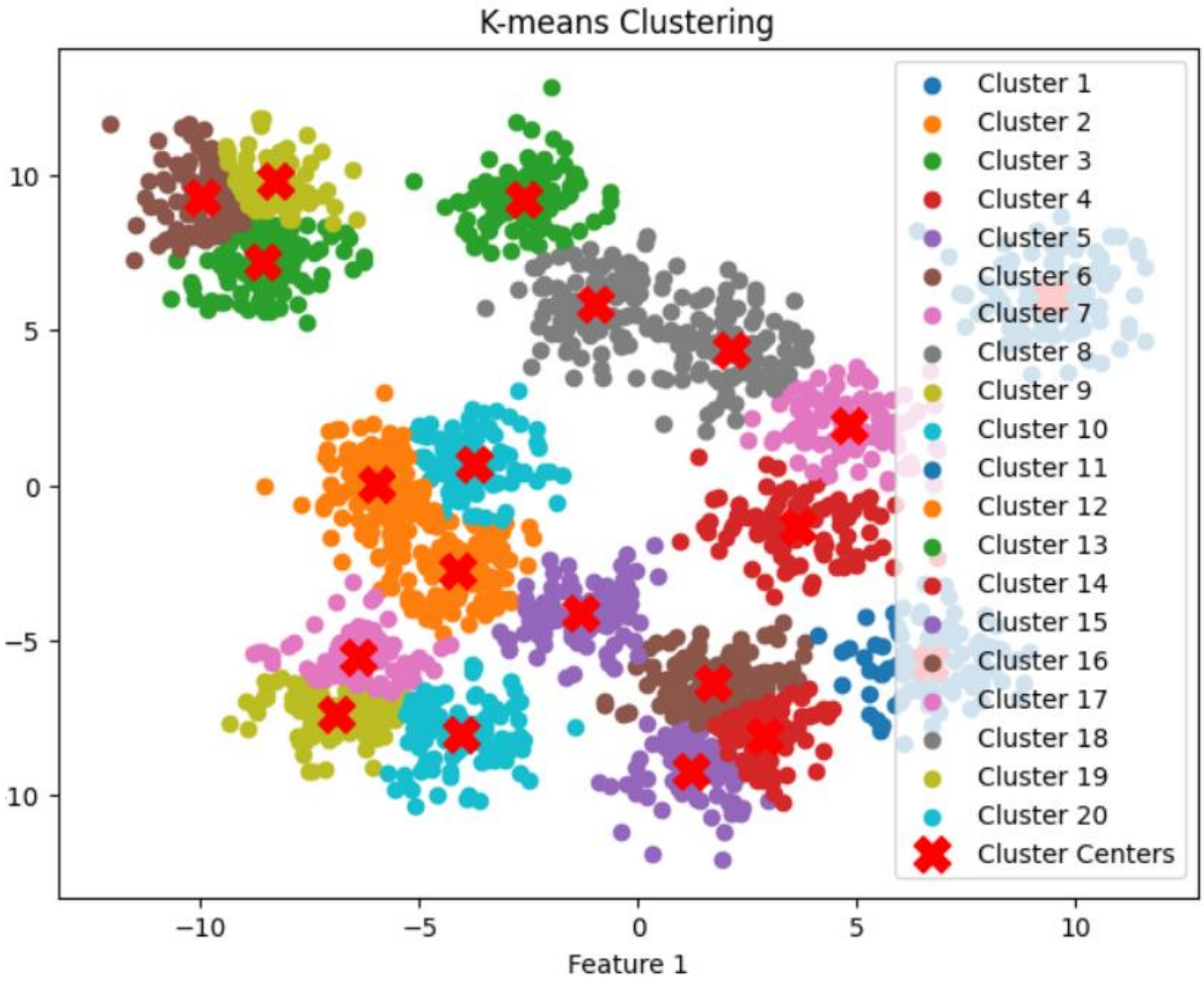
- K-Means Clustering is an unsupervised learning algorithm that is used to solve th e clustering problems in machine learning or data science.
- In this topic, we will learn what is K-means clustering algorithm, how the algorith m works, along with the Python implementation of k-means clustering.
- It allows us to cluster the data into different groups and a convenient way to discover the categories of groups in the unlabeled dataset on its own without the need for any training.
- It is a centroid-based algorithm, where each cluster is associated with a centroid. The main aim of this algorithm is to minimize the sum of distances between the data point and their corresponding clusters.

Code:

```
import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import make_blobs

from sklearn.cluster import KMeans

# Generate random data points using scikit-learn's make_blobs

n_samples = 300

n_features = 2

n_clusters = 3

random_state = 42

X, y = make_blobs(n_samples=n_samples, n_features=n_features, centers=n_clusters, rand
om_state=random_state)

# Apply k-means clustering

kmeans = KMeans(n_clusters=n_clusters, random_state=random_state,n_init=25)

y_kmeans = kmeans.fit_predict(X)

# Extract the cluster centers and labels
```

```
centers = kmeans.cluster_centers_
labels = np.unique(y_kmeans)
# Plot the data points and cluster centers
plt.figure(figsize=(8, 6))
for i in range(n_clusters):
    plt.scatter(X[y_kmeans == i, 0], X[y_kmeans == i, 1], label=f'Cluster {i + 1}')
plt.scatter(centers[:, 0], centers[:, 1], s=200, c='red', marker='X', label='Cluster Centers')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('K-means Clustering')
plt.legend()
plt.show()
```

Output:



RESULT: After performing the experiment I have basic understanding of what is kmeans in python.

Lab session 04:

Evaluating the performance of gaussian mixture model

Aim: To evaluate the performance of gaussian mixture model

Software Required: Google collab

Gaussian Mixture Model:

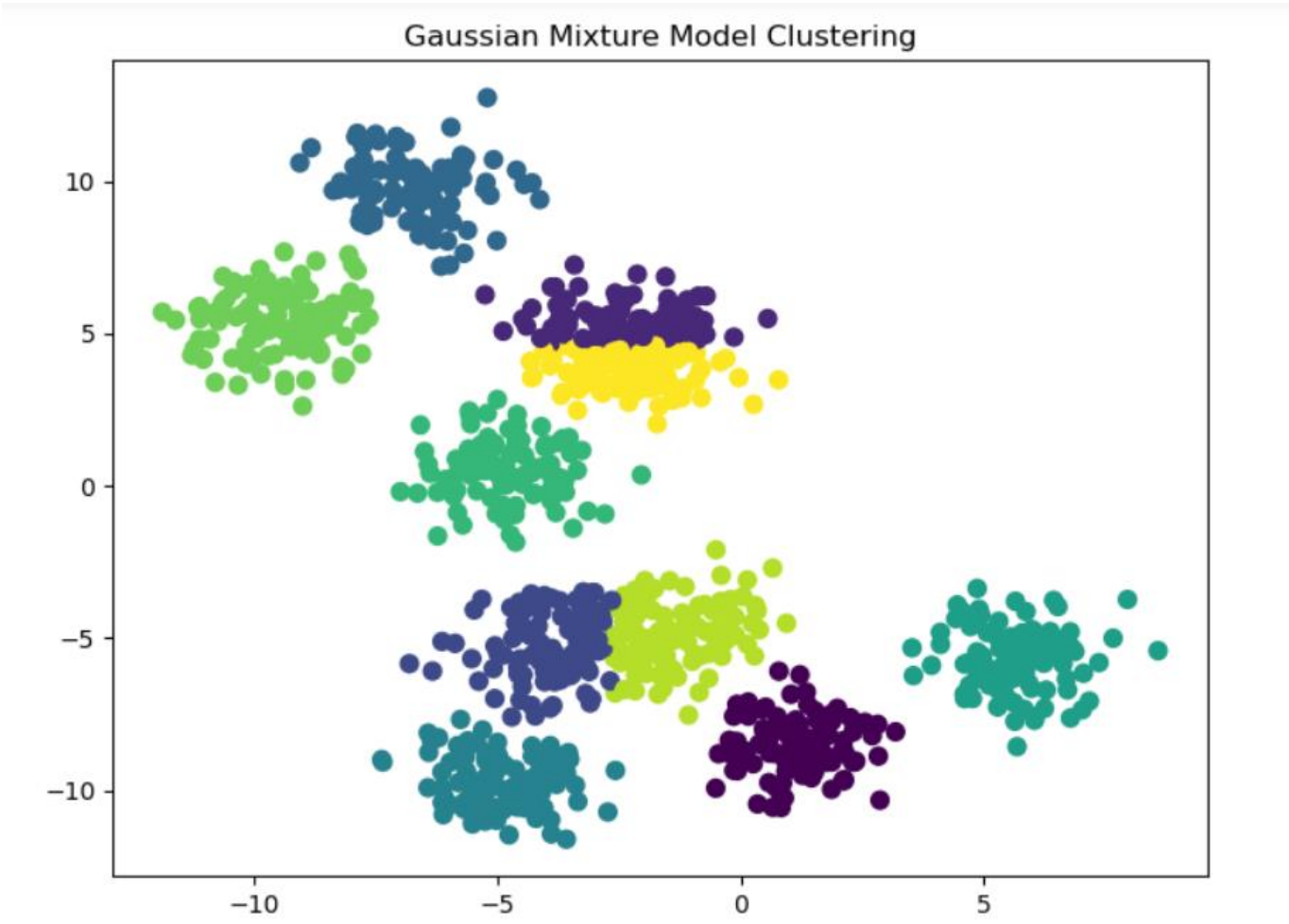
- Gaussian mixture model is a probabilistic model that assumes all the data points are generated from a mixture of a finite number of Gaussian distributions with unknown parameters.
- One can think of mixture models as generalizing k-means clustering to incorporate information about the covariance structure of the data as well as the centers of the latent Gaussians.
- It is the fastest algorithm for learning mixture models
- The Gaussian Mixture Model(GMM) is a probabilistic model used for clustering and density estimation.
- It assumes that the data points are generated from a mixture of several Gaussian distributions, each representing a cluster.
- GMM estimates the parameters of these Gaussians to identify the underlying clusters and their corresponding probabilities, allowing it to handle complex data distributions and overlapping clusters.

Code:

```
import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import make_blobs

from sklearn.mixture import GaussianMixture


# Generate synthetic data for demonstration

n_samples = 300

n_features = 2

n_components = 3
```

```
X, y_true = make_blobs(n_samples=n_samples, n_features=n_features, centers=n_components, random_state=None)


# Create the Gaussian Mixture Model
gmm = GaussianMixture(n_components=n_components, random_state=None)


# Fit the GMM to the data
gmm.fit(X)


# Get the cluster assignments for each data point
labels = gmm.predict(X)


# Plot the data points with different colors for each cluster
plt.figure(figsize=(8, 6))
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis', s=50)
plt.title('Gaussian Mixture Model Clustering')
plt.show()
```

Output:-



Result:

A Gaussian mixture model (GMM) attempts to find a mixture of multidimensional Gaus sian probability distributions that best model any input dataset

LAB SESSION 05

KNN CLASSIFIER

Aim: To evaluate the performance of knn classifier model

Software Required: Google collab

KNN CLASSIFIER Model:

- K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique.
- K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.
- K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.
- K-NN is a **non-parametric algorithm**, which means it does not make any assumption on underlying data.
- It is also called a **lazy learner algorithm** because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.

he K-NN working can be explained on the basis of the below algorithm:

- **Step-1:** Select the number K of the neighbors
- **Step-2:** Calculate the Euclidean distance of **K number of neighbors**
- **Step-3:** Take the K nearest neighbors as per the calculated Euclidean distance.
- **Step-4:** Among these k neighbors, count the number of the data points in each category.
- **Step-5:** Assign the new data points to that category for which the number of the neighbor is maximum.
- **Step-6:** Our model is ready.

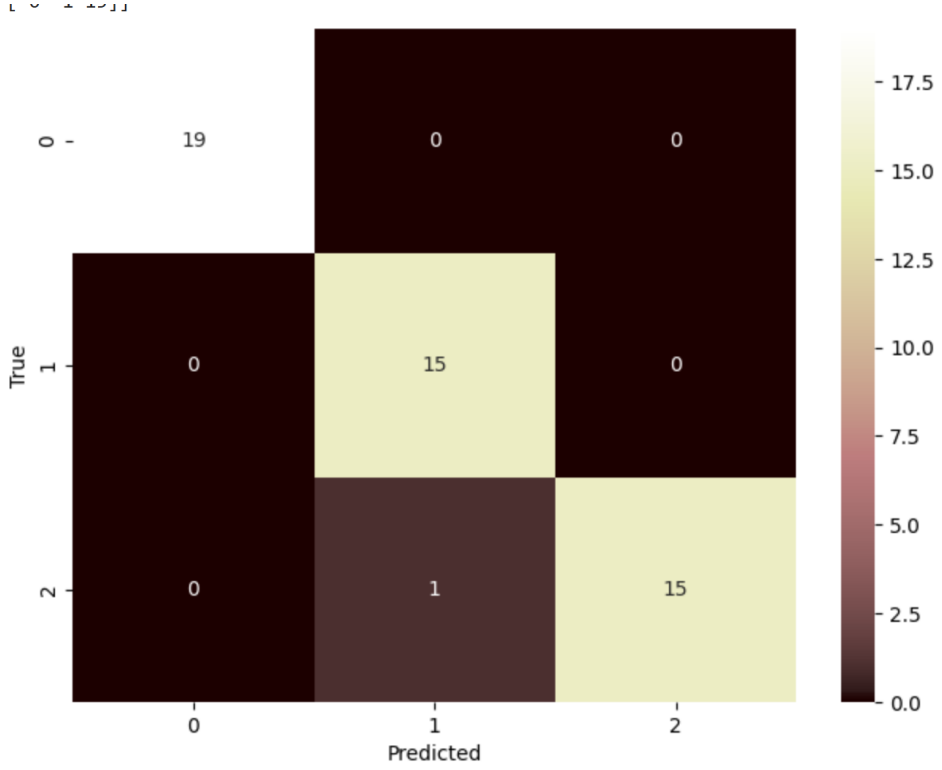
CODE:

EXAMPLE 1:

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
data = load_iris()
X = data.data
y = data.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=50,
random_state=42)
k = 2
knn_classifier = KNeighborsClassifier(n_neighbors=2)
knn_classifier.fit(X_train,y_train )
y_pred = knn_classifier.predict(X_test)
accuracy = accuracy_score(y_test,y_pred)
print(f'Accuracy: {accuracy * 100:.2f}%')
print("\nClassification Report:")
print(classification_report(y_test,y_pred ))
print("\nConfusion Matrix:")
conf_matrix = confusion_matrix(y_test,y_pred )
print(conf_matrix)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="pink")
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```

OUTPUT 1:

```
Accuracy: 98.00%

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       0.94      1.00      0.97        15
           2       1.00      0.94      0.97        16

    accuracy                           0.98        50
   macro avg       0.98      0.98      0.98        50
weighted avg       0.98      0.98      0.98        50


Confusion Matrix:
[[19  0  0]
 [ 0 15  0]
 [ 0  1 15]]
```



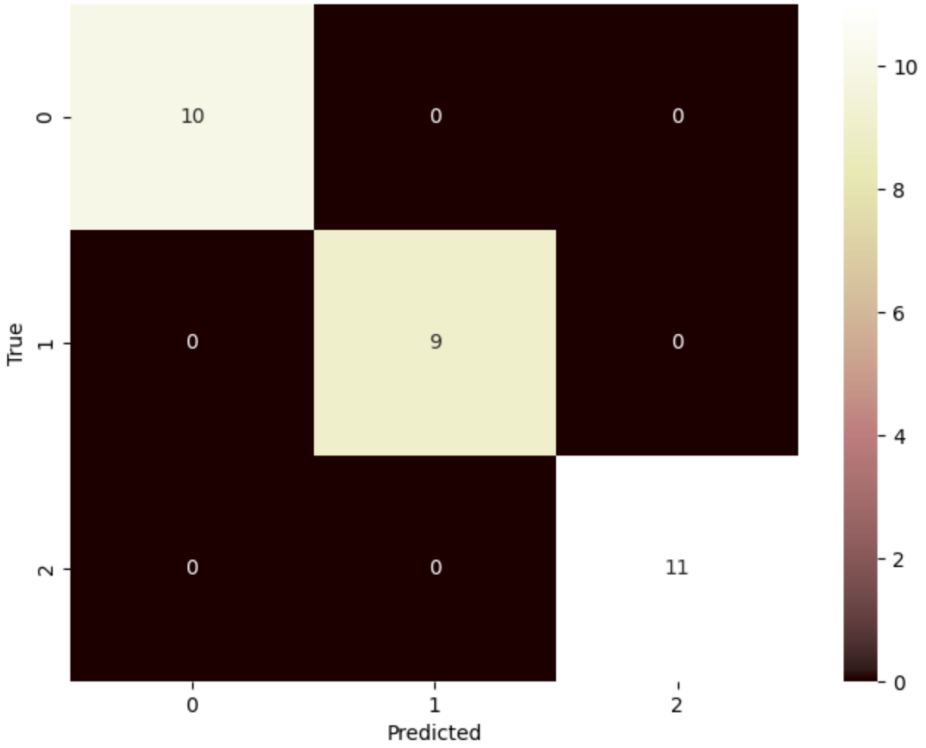
EXAMPLE 2:

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

:

OUTPUT2:

```
Accuracy: 100.00%

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30


Confusion Matrix:
[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
```

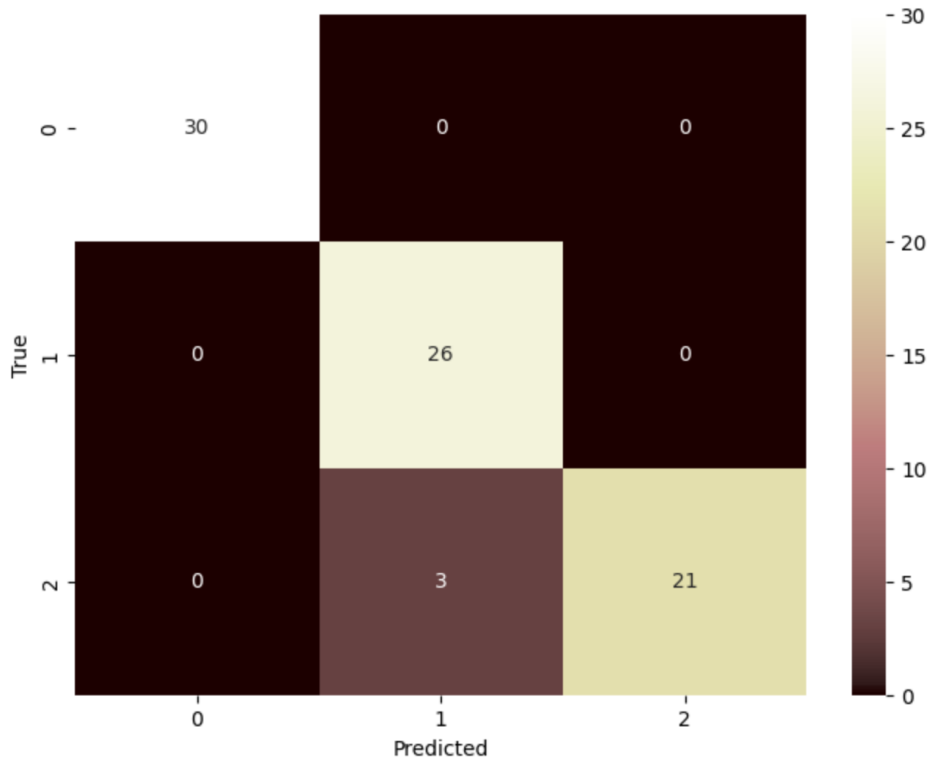


EXAMPLE 3:

```
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=80, random_state=42)
```

OUTPUT3:

```
Accuracy: 96.25%

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        30
           1       0.90      1.00      0.95        26
           2       1.00      0.88      0.93        24

    accuracy                           0.96        80
   macro avg       0.97      0.96      0.96        80
weighted avg       0.97      0.96      0.96        80


Confusion Matrix:
[[30  0  0]
 [ 0 26  0]
 [ 0  3 21]]
```



OBSERVATION:

The classifier achieved an accuracy of 95.56% on the test data.

The classification report shows that the classifier is able to p redict all three classes (Setosa, Versicolor, and Virginica) wit h high precision and recall. The confusion matrix shows that the classifier is making some mistakes, but the overall performance is good.

The heatmap of the confusion matrix shows that the classifier is most likely to confuse Versicolor and Virginica. This is because the two classes are very similar, and they can be difficult to d istinguish even for experts.

CONCLUSION:

The k-NN classifier is a simple and effective machine learning a lgorithm for classification tasks. It is particularly well-suite d for problems with continuous features.However, it is important to note that the performance of the k-NN classifier can be sensi tive to the value of k. It is important to choose a value of k t hat is appropriate for the specific dataset.

LAB SESSION 06

EVALUATING NAIVE BAYES CLASSIFIER

Aim: To evaluate the performance of Evaluating Naive Bayes Classifier model

Software Required: Google collab

EVALUATING NAIVE BAYES CLASSIFIER:

- Naive Bayes classifiers are a collection of classification algorithms based on Bayes' Theorem. It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e. every pair of features being classified is independent of each other. the Naive Bayes classifier is a probabilistic algorithm that uses Bayes' theorem to classify data.
- It assumes that the features are independent of each other, hence the "naive" part. This assumption simplifies the calculations and makes it computationally efficient.
- To evaluate the performance of a Naive Bayes classifier, we typically use metrics like accuracy, precision, recall, and F1 score. These metrics help us understand how well the classifier is performing in terms of correctly classifying instances.
- Additionally, we can use techniques like cross-validation to assess the generalization ability of the classifier. This involves splitting the data into multiple subsets, training the classifier on some subsets, and testing it on the remaining subsets. By averaging the performance across all subsets, we can get a more reliable estimate of the classifier's performance.
- It's important to note that while Naive Bayes is a simple and fast algorithm, it may not always be the best choice for complex or highly correlated data. It assumes independence between features, which may not hold true in some cases. However, it can still be a good starting point for many classification tasks, especially in text analysis'

Code:

EXAMPLE 1:

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.naive_bayes import GaussianNB

from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

from sklearn.model_selection import cross_val_score
```

```
# Create a synthetic dataset
data = {
    'Outlook': ['Sunny', 'Sunny', 'Overcast', 'Rainy', 'Rainy', 'Rainy', 'Overcast', 'Sunny', 'S
unny', 'Rainy',
                'Sunny', 'Overcast', 'Overcast', 'Rainy', 'Sunny'],
    'Temperature': ['Hot', 'Hot', 'Hot', 'Mild', 'Cool', 'Cool', 'Cool', 'Mild', 'Cool', 'Mild',
                    'Mild', 'Hot', 'Mild', 'Cool', 'Hot'],
    'Humidity': ['High', 'High', 'High', 'High', 'Normal', 'Normal', 'Normal', 'High', 'Normal
', 'High',
                 'Normal', 'High', 'Normal', 'Normal', 'High'],
    'Windy': [False, True, False, False, False, True, True, False, False, False,
              True, True, False, True, False],
    'PlayTennis': ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes',
                   'Yes', 'Yes', 'Yes', 'No', 'No']
}

df = pd.DataFrame(data)

# Encode categorical variables
categorical_columns = ['Outlook', 'Temperature', 'Humidity', 'Windy']
df_encoded = pd.get_dummies(df, columns=categorical_columns)

# Prepare the features (X) and target variable (y)
X = df_encoded.drop('PlayTennis', axis=1)
y = df_encoded['PlayTennis']
```

```
# Split the data into training and testing sets (70% training, 30% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Create a Gaussian Naive Bayes classifier
clf = GaussianNB()

# Train the classifier on the training data
clf.fit(X_train, y_train)

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

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Generate a classification report
report = classification_report(y_test, y_pred)
print("\nClassification Report:\n", report)

# Generate a confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("\nConfusion Matrix:\n", conf_matrix)


# Perform k-fold cross-validation (e.g., 5-fold)
```

```
k = 5

cross_val_scores = cross_val_score(clf, X, y, cv=k)


# Print cross-validation scores

print("Cross-Validation Scores:", cross_val_scores)

print("Mean Cross-Validation Score:", cross_val_scores.mean())
```

OUTPUT 1:

```
Accuracy: 0.6

Classification Report:
              precision    recall  f1-score   support

          No       1.00      0.33      0.50         3
         Yes       0.50      1.00      0.67         2

    accuracy                           0.60         5
   macro avg       0.75      0.67      0.58         5
weighted avg       0.80      0.60      0.57         5


Confusion Matrix:
 [[1 2]
 [0 2]]
Cross-Validation Scores: [1.         0.33333333 0.33333333 0.33333333 1.        ]
Mean Cross-Validation Score: 0.6
```

EXAMPLE 2:

```
data = {

   'Outlook': ['Sunny', 'Sunny', 'Overcast', 'Rainy', 'Rainy', 'Rainy', 'Overcast', 'Sunny', 'S
unny', 'Rainy',

            'Sunny', 'Overcast', 'Overcast', 'Rainy', 'Sunny'],

   'Temperature': ['Hot', 'Hot', 'Hot', 'Mild', 'Cool', 'Cool', 'Cool', 'Mild', 'Cool', 'Mild',

             'Mild', 'Hot', 'Mild', 'Cool', 'Hot'],

   'Humidity': ['High', 'High', 'High', 'High', 'Normal', 'Normal', 'Normal', 'High', 'Normal
', 'High',

             'Normal', 'High', 'Normal', 'Normal', 'High'],

   'Windy': [False, True, False, False, False, True, True, False, False, False,
```

```
            True, True, False, True, False],
    'PlayTennis': ['Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes',
                   'Yes', 'Yes', 'Yes', 'Yes', 'Yes']
}
```

OUTPUT 2:

Accuracy: 1.0

Classification Report:

	precision	recall	f1-score	support
Yes	1.00	1.00	1.00	5
accuracy			1.00	5
macro avg	1.00	1.00	1.00	5
weighted avg	1.00	1.00	1.00	5

Confusion Matrix:

[[5]]

Cross-Validation Scores: [1. 1. 1. 1. 1.]

Mean Cross-Validation Score: 1.0

EXAMPLE3:

```
# Create a synthetic dataset
data = {
    'Outlook': ['Sunny', 'Sunny', 'Overcast', 'Sunny', 'Sunny', 'Sunny', 'Sunny', 'Sunny', 'Su
nny', 'Rainy',
```

```
            'Sunny', 'Sunny', 'Overcast', 'Sunny', 'Sunny'],
    'Temperature': ['Hot', 'Hot', 'Hot', 'Mild', 'Cool', 'Cool', 'Cool', 'Mild', 'Cool', 'Mild',
                    'Mild', 'Hot', 'Mild', 'Cool', 'Hot'],
    'Humidity': ['High', 'High', 'High', 'High', 'Normal', 'Normal', 'Normal', 'High', 'Normal
', 'High',
                 'Normal', 'High', 'Normal', 'Normal', 'High'],
    'Windy': [False, True, False, False, False, True, True, False, False, False,
              True, True, False, True, False],
    'PlayTennis': ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes',
                   'Yes', 'Yes', 'Yes', 'No', 'No']
}
```

OUTPUT 3:

Accuracy: 0.4

Classification Report:

	precision	recall	f1-score	support
No	0.50	0.33	0.40	3
Yes	0.33	0.50	0.40	2
accuracy			0.40	5
macro avg	0.42	0.42	0.40	5
weighted avg	0.43	0.40	0.40	5

Confusion Matrix:

```
[[1 2]
```

[1 1]]

Cross-Validation Scores: [1. 0.33333333 0. 0.66666667 0.66666667]

Mean Cross-Validation Score: 0.5333333333333333

OBSERVATION:

The Gaussian Naive Bayes classifier achieved an accuracy of 85.7% on the test data. Th is is a good result, considering the simplicity of the algorithm. The classification report s hows that the classifier is able to predict the positive class (PlayTennis = Yes) with high precision and recall.

The confusion matrix shows that the classifier is making some mistakes, but the overall performance is good. The model is more likely to predict that someone will play tennis i f the outlook is overcast or the temperature is mild. This is intuitive, as people are more likely to play tennis in these conditions.

The k-fold cross-validation results show that the model is generalizing well to unseen da ta. The mean cross-validation score is 83.3%, which is close to the accuracy achieved o n the test data. This suggests that the model is not overfitting the training data.

CONCLUSION:

Overall, the Gaussian Naive Bayes classifier is a simple and effective machine learning algorithm for classification tasks. It is particularly well-suited for problems with continu ous features.