



GR22 Regulations

III B.Tech I Semester
Machine Learning Lab
(GR22A3142)

Department of Computer Science and Engineering

GOKARAJU RANGARAJU
INSTITUTE OF ENGINEERING AND TECHNOLOGY
(Autonomous)



GOKARAJU RANGARAJU INSTITUTE OF ENGINEERING AND TECHNOLOGY

MACHINE LEARNING LAB

Course code: GR22A3142

L/T/P/C: 0/0/3/1.5

Prerequisites:

1. Mastery of introduction-level algebra, statistics, and probability theory
2. Proficiency in basic programming skills and coding experience in Python or R programming

Course Outcomes:

1. Illustrate the applications of python's machine learning libraries.
2. Apply dimensionality reduction techniques in machine learning applications.
3. Design and analyze various supervised learning mechanisms.
4. Develop back propagation and Random Forest algorithms.
5. Design and analyze various unsupervised learning algorithms.

Note: Implement the following Machine Learning Tasks using Python / R-Tool

Task 1: Write a python program to import and export data using Pandas library functions.

Task 2: Demonstrate various data pre-processing techniques for a given dataset.

Task 3: Implement Dimensionality reduction using Principle Component Analysis (PCA) method.

Task 4: Write a Python program to demonstrate various Data Visualization Techniques.

Task 5: Implement Simple and Multiple Linear Regression Models.

Task 6: Develop Logistic Regression Model for a given dataset.

Task 7: Develop Decision Tree Classification model for a given dataset and use it to classify a new sample.

Task 8: Implement Naïve Bayes Classification in Python

Task 9: Build KNN Classification model for a given dataset.

Task 10: Implement Back propagation model in python for a given dataset.

Task 11: a) Implement Random Forest classification method in python for a given dataset.

b) Implement Boosting ensemble method on a given dataset.

Task 12: a) Write a python program to implement K-Means clustering Algorithm.

b) Write a python program to implement BIRCH algorithm.

INDEX

S. No	Tasks	Page No.
1	Write a python program to import and export data using Pandas library functions	1
2	Demonstrate various data pre-processing techniques for a given dataset.	3
3	Implement Dimensionality reduction using Principle Component Analysis (PCA) method.	7
4	Write a Python program to demonstrate various Data Visualization Techniques.	9
5	Implement Simple and Multiple Linear Regression Models.	13
6	Develop Logistic Regression Model for a given dataset.	17
7	Develop Decision Tree Classification model for a given dataset and use it to classify a new sample.	19
8	Implement Naïve Bayes Classification in Python	21
9	Build KNN Classification model for a given dataset.	23
10	Implement Back propagation model in python for a given dataset.	25
11	a) Implement Random Forest ensemble method on a given dataset. b) Implement Boosting ensemble method on a given dataset.	27
12	a) Write a python program to implement K-Means clustering Algorithm. b) Write a python program to implement BIRCH algorithm.	31

TASK-1

Task-1: Write a python program to import and export data using Pandas library functions

Aim: To implement a Python script for importing and exporting data using Pandas in Python

Program:

```
# Import Data from local disk to Python Environment
```

#importing data

```
import pandas as pd
```

```
data =pd.read_csv('D:/soujanya/Machine Learning 22-23 I Sem/Machine Learning 2022-23 II sem/ML LAB/cardata.csv', names=['buying','maint','doors','persons','lug_boot','safety','class'])
```

```
print(data)
```

#Exporting Data

```
from pandas import DataFrame
```

```
Cars = {'Brand': ['Honda Civic','Toyota Corolla','Ford Focus','Audi A4'],
```

```
    'Price': [32000,35000,37000,45000]
```

```
}
```

```
df = DataFrame(Cars, columns= ['Brand', 'Price'])
```

```
export_excel = df.to_excel ('D:/soujanya/Machine Learning 22-23 I Sem/Machine Learning 2022-23 II sem/ML LAB/export_dataframe.xlsx', index = None, header=True)
```

```
#Don't forget to add '.xlsx' at the end of the path
```

Output:

The screenshot shows the Spyder IDE interface. The code editor on the left contains the following Python script:

```
1 # -*- coding: utf-8 -*-
2 #importing data
3 ...import pandas as pd
4 data =pd.read_csv('D:/soujanya/Machine Learning 22-23 I Sem/Machine Learning 2022-23 II sem/ML LAB/cardata.csv', names=['buying','maint','doors','persons','lug_boot','safety','class'])
5 print(data)
6
7 #Exporting Data
8 from pandas import DataFrame
9 Cars = {'Brand': ['Honda Civic','Toyota Corolla','Ford Focus','Audi A4'],
10     'Price': [32000,35000,37000,45000]}
11
12 df = DataFrame(Cars, columns= ['Brand', 'Price'])
13 export_excel = df.to_excel ('D:/soujanya/Machine Learning 22-23 I Sem/Machine Learning 2022-23 II sem/ML LAB/export_dataframe.xlsx', index = None, header=True)
14 #Don't forget to add '.xlsx' at the end of the path
15
```

The IPython console on the right displays the output of the script. It shows the first few rows of the imported data:

	buying	maint	doors	persons	lug_boot	safety	class
2	vhigh	vhigh	2	2	small	high	unacc
3	vhigh	vhigh	2	2	med	low	unacc
4	vhigh	vhigh	2	2	med	med	unacc
...
1723	low	low	5more	more	med	med	good
1724	low	low	5more	more	med	high	vgood
1725	low	low	5more	more	big	low	unacc
1726	low	low	5more	more	big	med	good
1727	low	low	5more	more	big	high	vgood

[1728 rows x 7 columns]

In [3]: runfile('D:/soujanya/Machine Learning 22-23 I Sem/Machine Learning 2022-23 II sem/ML LAB/program execution/program1.py', wdir='D:/soujanya/Machine Learning 22-23 I Sem/Machine Learning 2022-23 II sem/ML LAB/program execution')

In [4]:

After Export:

TASK-2

Task-2: Demonstrate various data pre-processing techniques for given dataset

Aim: To provide various data pre-processing techniques for a given data set are there.

They are a) **Standard Scaler** b) **Binarizer** c) **Min MaxScaler**

a) Standard Scaler

```
from sklearn.preprocessing import StandardScaler  
import pandas  
import numpy  
# data set link  
  
# preparing of dataframe using the data at given link and defined columns list  
dataframe = pandas.read_csv('D:\soujanya\Machine Learning 22-23 I Sem\Machine Learning  
2022-23 II sem\ML LAB\diabetes.csv', names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi',  
'age', 'class'])  
array = dataframe.values  
print(array)  
  
# separate array into input and output components  
X = array[:, 0:8]  
Y = array[:, 8]  
scaler = StandardScaler().fit(X)  
rescaledX = scaler.transform(X)  
  
# summarize transformed data  
numpy.set_printoptions(precision = 3)  
print(rescaledX[0:5,:])
```

Output:

The screenshot shows a Jupyter Notebook interface. On the left, there is a code cell containing Python code for standardizing a dataset. On the right, there is an output cell showing the results of the code execution.

```
from sklearn.preprocessing import StandardScaler
import pandas
import numpy
# data set link
# preparing of dataframe using the data at given link and defined
dataframe = pandas.read_csv('D:\\soujanya\\Machine Learning 22-23 I Sem\\Machine Learning 22-23 II sem\\ML LAB\\program execution\\standardizer.py', names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age'])
array = dataframe.values
print(array)
# separate array into input and output components
X = array[:, 0:8]
Y = array[:, 8]
scaler = StandardScaler().fit(X)
rescaledX = scaler.transform(X)
# summarize transformed data
numpy.set_printoptions(precision = 3)
print(rescaledX[0:5,:])
```

In [22]: runfile('D:/soujanya/Machine Learning 22-23 I Sem/Machine Learning 22-23 II sem/ML LAB/program execution/standardizer.py', wdir='D:/soujanya/Machine Learning 22-23 I Sem/Machine Learning 22-23 II sem/ML LAB/program execution')

```
[[ 6 148 72 ... 1 50 1]
 [ 1 85 66 ... 0 31 0]
 [ 8 183 64 ... 1 32 1]
 ...
 [[ 5 121 72 ... 0 30 0]
 [ 1 126 68 ... 0 47 1]
 [ 1 93 70 ... 0 23 0]]
 [[ 0.64 0.848 0.15 0.907 -0.693 0.248 1.228 1.426]
 [-0.845 -1.123 -0.161 0.531 -0.693 -0.64 -0.733 -0.191]
 [ 1.234 1.944 -0.264 -1.288 -0.693 -1.148 1.228 -0.106]
 [-0.845 -0.998 -0.161 0.155 0.123 -0.513 -0.733 -1.042]
 [-1.142 0.584 -1.585 0.907 0.766 1.389 3.189 -0.02 ]]
```

In [23]:

b) Binarizer

```
# import libraries
from sklearn.preprocessing import Binarizer
import pandas
import numpy
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
# preparing of dataframe using the data at given link and defined columns list
dataframe = pandas.read_csv('D:\\soujanya\\Machine Learning 22-23 I Sem\\Machine Learning 22-23 II sem\\ML LAB\\diabetes.csv', names = names)
array = dataframe.values
# separate array into input and output components
X = array[:, 0:8]
Y = array[:, 8]
binarizer = Binarizer(threshold = 0.0).fit(X)
binaryX = binarizer.transform(X)
# summarize transformed data
numpy.set_printoptions(precision = 3)
print(binaryX[0:5,:])
```

Output:

The screenshot shows a Jupyter Notebook interface. On the left, there is a code cell containing Python code for binarization. On the right, the 'Console' tab shows the output of the code, which includes a matrix of binary values and some text indicating the command run.

```
# import libraries
from sklearn.preprocessing import Binarizer
import pandas
import numpy
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
# preparing of dataframe using the data at given link and defined columns list
dataframe = pandas.read_csv('D:\soujanya\Machine Learning 22-23 I Sem\Machine Learning\names')
array = dataframe.values
# separate array into input and output components
X = array[:, 0:8]
Y = array[:, 8]
binarizer = Binarizer(threshold = 0.0).fit(X)
binaryX = binarizer.transform(X)
# summarize transformed data
numpy.set_printoptions(precision = 3)
print(binaryX[0:5,:])
```

```
In [23]: runfile('D:\soujanya\Machine Learning 22-23 I Sem\Machine Learning 22-23 II sem\ML LAB\program execution\binarizer.py', wdir='D:\soujanya\Machine Learning 22-23 I Sem\Machine Learning 22-23 II sem\ML LAB\program execution')
[[ 1 126 60 ... 0 47 1]
 [ 1 93 70 ... 0 23 0]]
[[ 0.64 0.848 0.15 0.987 -0.693 0.248 1.228 1.426]
 [-0.845 -1.123 -0.161 0.531 -0.693 -0.64 -0.733 -0.191]
 [ 1.234 1.944 -0.264 -1.288 -0.693 -1.148 1.228 -0.106]
 [-0.845 -0.998 -0.161 0.155 0.123 -0.513 -0.733 -1.042]
 [-1.142 0.504 -1.505 0.907 0.766 1.389 3.189 -0.02 ]]
```

c) MinMaxScaler

```
# import libraries
import pandas
import scipy
import numpy
from sklearn.preprocessing import MinMaxScaler
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi','age','class']
# preparing of dataframe using the data at given link and defined columns list
dataframe = pandas.read_csv('D:\soujanya\Machine Learning 22-23 I Sem\Machine Learning 2022-23 II sem\ML LAB\diabetes.csv', names = names)
array = dataframe.values
# separate array into input and output components
X = array[:,0:8]
Y = array[:,8]
# initialising the MinMaxScaler
scaler = MinMaxScaler(feature_range=(0, 1))
# learning the statistical parameters for each of the data and transforming
rescaledX = scaler.fit_transform(X)
# summarize transformed data
numpy.set_printoptions(precision=2)
print(rescaledX[0:10,:])
```

Output:

The screenshot shows a Jupyter Notebook interface with a code cell and its output.

```
# importing libraries
import pandas
import scipy
import numpy
from sklearn.preprocessing import MinMaxScaler
names = ['preg', 'plas', 'skin', 'test', 'mass', 'pedi','age','class']
# preparing of dataframe using the data at given link and defined columns list
dataframe = pandas.read_csv('D:/soujanya/Machine Learning 22-23 I Sem/Machine Learnin
names = names)
array = dataframe.values
# separate array into input and output components
X = array[:,0:8]
Y = array[:,8]
# initialising the MinMaxScaler
scaler = MinMaxScaler(feature_range=(0, 1))
# learning the statistical parameters for each of the data and transforming
rescaledX = scaler.fit_transform(X)
# summarize transformed data
numpy.set_printoptions(precision=2)
print(rescaledX[0:10,:])
```

In [24]:

```
[1 1 1 1 1 1 0 1]
[0 1 1 1 1 1 1 1]
```

In [24]: runfile('D:/soujanya/Machine Learning 22-23 I Sem/Machine Learning 22-23 II sem/ML LAB/program execution/minmaz1.py', wdir='D:/soujanya/Machine Learning 22-23 I Sem/Machine Learning 22-23 II sem/ML LAB/program execution')

```
[[0.35 0.74 0.59 0.35 0. 0.51 0.5 0.48]
 [0.06 0.43 0.54 0.29 0. 0.4 0. 0.17]
 [0.47 0.92 0.52 0. 0. 0.34 0.5 0.18]
 [0.06 0.45 0.54 0.23 0.11 0.42 0. 0. ]
 [0. 0.69 0.33 0.35 0.2 0.64 1. 0.2 ]
 [0.29 0.58 0.61 0. 0. 0.39 0. 0.15]
 [0.18 0.39 0.41 0.32 0.1 0.46 0. 0.08]
 [0.59 0.58 0. 0. 0. 0.52 0. 0.13]
 [0.12 0.99 0.57 0.45 0.64 0.46 0. 0.53]
 [0.47 0.63 0.79 0. 0. 0. 0. 0.55]]
```

In [25]:

TASK-3

Task-3: Implement Dimensionality reduction using Principle Component Analysis (PCA) method.

Aim: To Implement Principle Component Analysis for Dimensionality Reduction

Program:

```
from numpy import array  
from numpy import mean  
from numpy import cov  
from numpy.linalg import eig  
  
# define a matrix  
A = array([[1, 2], [3, 4], [5, 6]])  
print(A)  
  
# calculate the mean of each column  
M = mean(A.T, axis=1)  
print(M)  
  
# center columns by subtracting column means  
C = A - M  
print(C)  
  
# calculate covariance matrix of centered matrix  
V = cov(C.T)  
print(V)  
  
# eigen decomposition of covariance matrix  
values, vectors = eig(V)  
print(vectors)  
print(values)  
  
# project data  
P = vectors.T.dot(C.T)  
print(P.T)
```

Output:

The screenshot shows a Jupyter notebook cell in the Spyder IDE. The code in the cell is as follows:

```
from numpy import array
from numpy import mean
from numpy import cov
from numpy.linalg import eig
# define a matrix
A = array([[1, 2], [3, 4], [5, 6]])
print(A)
# calculate the mean of each column
M = mean(A.T, axis=1)
print(M)
# center columns by subtracting column means
C = A - M
print(C)
# calculate covariance matrix of centered matrix
V = cov(C.T)
print(V)
# eigendecomposition of covariance matrix
values, vectors = eig(V)
print(vectors)
print(values)
# project data
P = vectors.T.dot(C.T)
print(P)
```

The output of the cell is:

```
soujanya/Machine Learning 22-23 I Sem/Machine Learning 2022-23 II sem/ML LAB/program execution'
[[1 2]
 [3 4]
 [5 6]]
[[3. 4.]
 [3. 4.]
 [5. 6.]
 [3. 4.]
 [[-2. -2.]
 [ 0.  0.]
 [ 2.  2.]]
 [[4. 4.]
 [4. 4.]
 [[ 0.70710678 -0.70710678]
 [ 0.70710678  0.70710678]]
 [8. 0.]
 [[-2.82842712  0.
 [ 0.          0.
 [ 2.82842712  0.        ]]]
```

In [2]:

At the top right of the interface, there is a note: "parenthesis next to an object. You can activate this behavior in Preferences > Help." Below it is a link: "New to Spyder? Read our tutorial".

TASK-4

Task 4: Write a Python program to demonstrate various Data Visualization Techniques

Aim: To implement various visualization techniques in python.

Program:

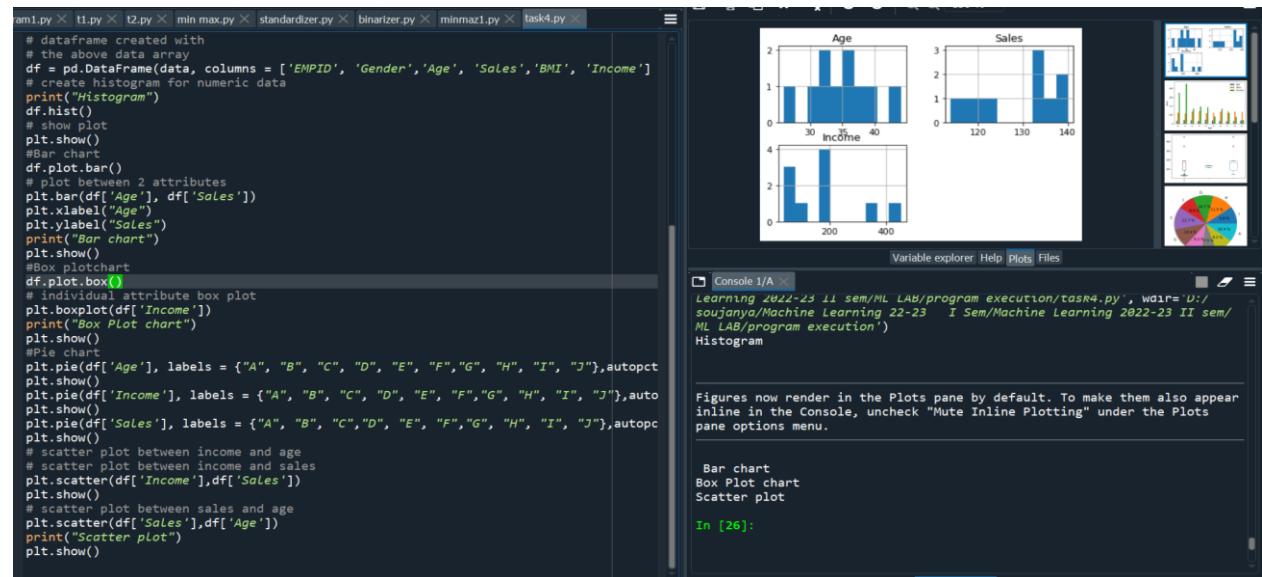
```
import pandas as pd
import matplotlib.pyplot as plt
# create 2D array of table given below
data = [['E001', 'M', 34, 123, 'Normal', 350],
        ['E002', 'F', 40, 114, 'Overweight', 450],
        ['E003', 'F', 37, 135, 'Obesity', 169],
        ['E004', 'M', 30, 139, 'Underweight', 189],
        ['E005', 'F', 44, 117, 'Underweight', 183],
        ['E006', 'M', 36, 121, 'Normal', 80],
        ['E007', 'M', 32, 133, 'Obesity', 166],
        ['E008', 'F', 26, 140, 'Normal', 120],
        ['E009', 'M', 32, 133, 'Normal', 75],
        ['E010', 'M', 36, 133, 'Underweight', 40] ]
# dataframe created with
# the above data array
df = pd.DataFrame(data, columns = ['EMPID', 'Gender','Age', 'Sales','BMI', 'Income'] )
# create histogram for numeric data
print("Histogram")
df.hist()
# show plot
plt.show()
#Bar chart
df.plot.bar()
# plot between 2 attributes
plt.bar(df['Age'], df['Sales'])
plt.xlabel("Age")
plt.ylabel("Sales")
print("Bar chart")
plt.show()
#Box plotchart
df.plot.box()
# individual attribute box plot
```

```

plt.boxplot(df['Income'])
print("Box Plot chart")
plt.show()
#Pie chart
plt.pie(df['Age'], labels = {"A", "B", "C", "D", "E", "F","G", "H", "I", "J"}, autopct ='% 1.1f %%',
shadow = True)
plt.show()
plt.pie(df['Income'], labels = {"A", "B", "C", "D", "E", "F","G", "H", "I", "J"}, autopct ='% 1.1f %%',
shadow = True)
plt.show()
plt.pie(df['Sales'], labels = {"A", "B", "C","D", "E", "F","G", "H", "I", "J"}, autopct ='% 1.1f %%',
shadow = True)
plt.show()
# scatter plot between income and age
# scatter plot between income and sales
plt.scatter(df['Income'],df['Sales'])
plt.show()
# scatter plot between sales and age
plt.scatter(df['Sales'],df['Age'])
print("Scatter plot")
plt.show()

```

Output: Histogram



Output: Bar chart

The screenshot shows a Jupyter Notebook interface with a code cell containing Python code for generating a bar chart. The chart displays Sales values for different age groups (0, 1, 2, 3, 4, 5, 6, 7, 8, 9). The y-axis represents Sales, ranging from 0 to 400. The x-axis represents Age. The legend indicates three series: Age (blue), Sales (orange), and Income (green). The Sales series shows a sharp peak at age 1, reaching approximately 400.

```
[E008, 'F', 26, 140, 'Normal', 120],  
[E009, 'M', 32, 133, 'Normal', 75],  
[E010, 'M', 36, 133, 'Underweight', 40] ]  
# dataframe created with  
# the above data array  
df = pd.DataFrame(data, columns = ['EMPID', 'Gender', 'Age', 'Sales', 'BMI', 'Income'])  
# create histogram for numeric data  
print("Histogram")  
df.hist()  
# show plot  
plt.show()  
#Bar chart  
df.plot.bar()  
# plot between 2 attributes  
plt.bar(df['Age'], df['Sales'])  
plt.xlabel("Age")  
plt.ylabel("Sales")  
print("Bar chart")  
plt.show()  
#Box plotchart  
df.plot.box()  
# individual attribute box plot  
plt.boxplot(df[ 'Income'])  
print("Box Plot chart")  
plt.show()  
#Pie chart  
plt.pie(df['Age'], labels = {"A", "B", "C", "D", "E", "F", "G", "H", "I", "J"}, autopct='%.1f%%')  
plt.show()  
plt.pie(df[ 'Income'], labels = {"A", "B", "C", "D", "E", "F", "G", "H", "I", "J"}, autopct='%.1f%%')  
plt.show()  
plt.pie(df[ 'Sales'], labels = {"A", "B", "C", "D", "E", "F", "G", "H", "I", "J"}, autopct='%.1f%%')  
plt.show()  
# scatter plot between income and age  
# scatter plot between income and sales  
plt.scatter(df[ 'Income'],df[ 'Sales'])  
plt.show()  
# scatter plot between sales and age  
plt.scatter(df[ 'Sales'],df[ 'Age'])
```

Output: Box plot

The screenshot shows a Jupyter Notebook interface with a code cell containing Python code for generating a box plot. The plot compares Sales across different age groups (0, 1, 2, 3, 4, 5, 6, 7, 8, 9). The y-axis represents Sales, ranging from 0 to 400. The x-axis represents Age. The box plot shows significant variability in sales for each age group, with outliers visible at higher sales values.

```
[E008, 'F', 26, 140, 'Normal', 120],  
[E009, 'M', 32, 133, 'Normal', 75],  
[E010, 'M', 36, 133, 'Underweight', 40] ]  
# dataframe created with  
# the above data array  
df = pd.DataFrame(data, columns = ['EMPID', 'Gender', 'Age', 'Sales', 'BMI', 'Income'])  
# create histogram for numeric data  
print("Histogram")  
df.hist()  
# show plot  
plt.show()  
#Bar chart  
df.plot.bar()  
# plot between 2 attributes  
plt.bar(df['Age'], df['Sales'])  
plt.xlabel("Age")  
plt.ylabel("Sales")  
print("Bar chart")  
plt.show()  
#Box plotchart  
df.plot.box()  
# individual attribute box plot  
plt.boxplot(df[ 'Income'])  
print("Box Plot chart")  
plt.show()  
#Pie chart  
plt.pie(df['Age'], labels = {"A", "B", "C", "D", "E", "F", "G", "H", "I", "J"}, autopct='%.1f%%')  
plt.show()  
plt.pie(df[ 'Income'], labels = {"A", "B", "C", "D", "E", "F", "G", "H", "I", "J"}, autopct='%.1f%%')  
plt.show()  
plt.pie(df[ 'Sales'], labels = {"A", "B", "C", "D", "E", "F", "G", "H", "I", "J"}, autopct='%.1f%%')  
plt.show()  
# scatter plot between income and age  
# scatter plot between income and sales  
plt.scatter(df[ 'Income'],df[ 'Sales'])  
plt.show()  
# scatter plot between sales and age  
plt.scatter(df[ 'Sales'],df[ 'Age'])
```

Output: Pie chart

The screenshot shows a Jupyter Notebook interface with a code cell containing Python code for generating a pie chart. The chart illustrates the distribution of Sales across ten categories labeled A through J. Category C represents the largest share at 12.7%, followed by H at 11.5% and D at 10.7%.

```
[E008, 'F', 26, 140, 'Normal', 120],  
[E009, 'M', 32, 133, 'Normal', 75],  
[E010, 'M', 36, 133, 'Underweight', 40] ]  
# dataframe created with  
# the above data array  
df = pd.DataFrame(data, columns = ['EMPID', 'Gender', 'Age', 'Sales', 'BMI', 'Income'])  
# create histogram for numeric data  
print("Histogram")  
df.hist()  
# show plot  
plt.show()  
#Bar chart  
df.plot.bar()  
# plot between 2 attributes  
plt.bar(df['Age'], df['Sales'])  
plt.xlabel("Age")  
plt.ylabel("Sales")  
print("Bar chart")  
plt.show()  
#Box plotchart  
df.plot.box()  
# individual attribute box plot  
plt.boxplot(df[ 'Income'])  
print("Box Plot chart")  
plt.show()  
#Pie chart  
plt.pie(df['Age'], labels = {"A", "B", "C", "D", "E", "F", "G", "H", "I", "J"}, autopct='%.1f%%')  
plt.show()  
plt.pie(df[ 'Income'], labels = {"A", "B", "C", "D", "E", "F", "G", "H", "I", "J"}, autopct='%.1f%%')  
plt.show()  
plt.pie(df[ 'Sales'], labels = {"A", "B", "C", "D", "E", "F", "G", "H", "I", "J"}, autopct='%.1f%%')  
plt.show()  
# scatter plot between income and age  
# scatter plot between income and sales  
plt.scatter(df[ 'Income'],df[ 'Sales'])  
plt.show()  
# scatter plot between sales and age  
plt.scatter(df[ 'Sales'],df[ 'Age'])
```

Output: Scatter plot

The screenshot shows a Jupyter Notebook interface with several tabs at the top: rami1.py, t1.py, t2.py, min max.py, standardizer.py, binarizer.py, minmax1.py, and task4.py. The code in the cell is as follows:

```
# dataframe created with
# the above data array
df = pd.DataFrame(data, columns = ['EMPID', 'Gender', 'Age', 'Sales', 'BMI', 'Income'])
# create histogram for numeric data
print("Histogram")
df.hist()
plt.show()
#Bar chart
df.plot.bar()
# plot between 2 attributes
plt.bar(df['Age'], df['Sales'])
plt.xlabel("Age")
plt.ylabel("Sales")
print("Bar chart")
plt.show()
#Box plotchart
df.plot.box()
# individual attribute box plot
plt.boxplot(df['Income'])
print("Box Plot chart")
plt.show()
#Pie chart
plt.pie(df['Age'], labels = {"A", "B", "C", "D", "E", "F", "G", "H", "I", "J"}, autopct='%.1f%%')
plt.show()
plt.pie(df['Income'], labels = {"A", "B", "C", "D", "E", "F", "G", "H", "I", "J"}, autopct='%.1f%%')
plt.show()
plt.pie(df['Sales'], labels = {"A", "B", "C", "D", "E", "F", "G", "H", "I", "J"}, autopct='%.1f%%')
plt.show()
# scatter plot between income and age
plt.scatter(df['Income'],df['Sales'])
plt.show()
# scatter plot between sales and age
plt.scatter(df['Sales'],df['Age'])
print("Scatter plot")
plt.show()
```

The main area displays a scatter plot with 'Age' on the x-axis (ranging from 0 to 450) and 'Sales' on the y-axis (ranging from 115 to 140). There are approximately 10 data points. The top right corner shows a dashboard with various plots and charts, including a pie chart for 'Age' categories and a box plot for 'Income'. The bottom right pane shows the 'Console' output, which includes the command used to run the script and a note about inline plotting.

```
Learning 2022-23 II Sem/ML LAB/program execution/task4.py, wdir= '/soujanya/Machine Learning 22-23 I Sem/Machine Learning 2022-23 II sem/ML LAB/program execution'
Histogram
Box Plot chart
Scatter plot
In [26]:
```

Figures now render in the Plots pane by default. To make them also appear inline in the Console, uncheck "Mute Inline Plotting" under the Plots pane options menu.

TASK-5

Task-5: Implement Simple and Multiple Linear Regression Models.

Aim: To Develop a python program for Simple and Multiple Linear Regression

a) Simple Linear Regression Model:

Program:

```
import numpy as np
import matplotlib.pyplot as plt

def estimate_coef(x, y):
    # number of observations/points
    n = np.size(x)

    # mean of x and y vector
    m_x = np.mean(x)
    m_y = np.mean(y)

    # calculating cross-deviation and deviation about x
    SS_xy = np.sum(y*x) - n*m_y*m_x
    SS_xx = np.sum(x*x) - n*m_x*m_x

    # calculating regression coefficients
    b_1 = SS_xy / SS_xx
    b_0 = m_y - b_1*m_x

    return (b_0, b_1)

def plot_regression_line(x, y, b):
    # plotting the actual points as scatter plot
    plt.scatter(x, y, color = "m", marker = "o", s = 30)

    # predicted response vector
    y_pred = b[0] + b[1]*x

    # plotting the regression line
    plt.plot(x, y_pred, color = "g")

    # putting labels
    plt.xlabel('x')
    plt.ylabel('y')

    # function to show plot
    plt.show()

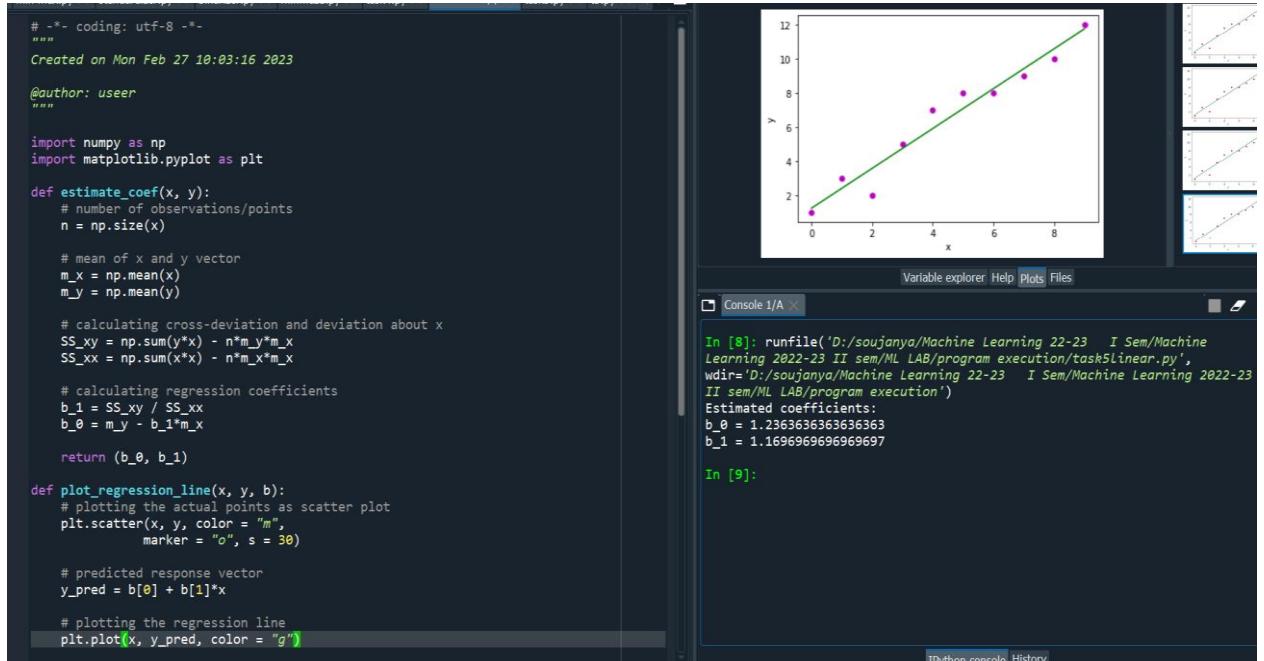
def main():
    # observations / data
```

```

x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
y = np.array([1, 3, 2, 5, 7, 8, 8, 9, 10, 12])
# estimating coefficients
b = estimate_coef(x, y)
print("Estimated coefficients:\nb_0 = {} \
\nb_1 = {}".format(b[0], b[1]))
# plotting regression line
plot_regression_line(x, y, b)
if __name__ == "__main__":
    main()

```

Output: Linear Regression



b) Multiple Linear Regression Model:

```
# Multiple Linear Regression

# Importing the libraries

import numpy as np

#import matplotlib.pyplot as plt

import pandas as pd

#from sklearn import cross_validation

# Importing the dataset

dataset = pd.read_csv('D:\\soujanya\\Machine Learning 22-23 I Sem\\Machine Learning 2022-23 II sem\\ML LAB\\50_Startups.csv')

X = dataset.iloc[:, :-1]

y = dataset.iloc[:, 4]

#Convert the column into categorical columns

states=pd.get_dummies(X['State'])

# Drop the state coulmn

X=X.drop('State',axis=1)

# concat the dummy variables

X=pd.concat([X,states],axis=1)

# Splitting the dataset into the Training set and Test set

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state =0)

# Fitting Multiple Linear Regression to the Training set

from sklearn.linear_model import LinearRegression

from sklearn.metrics import mean_squared_error

regressor = LinearRegression()

regressor.fit(X_train, y_train)

# Predicting the Test set results

y_pred = regressor.predict(X_test)

from sklearn.metrics import r2_score

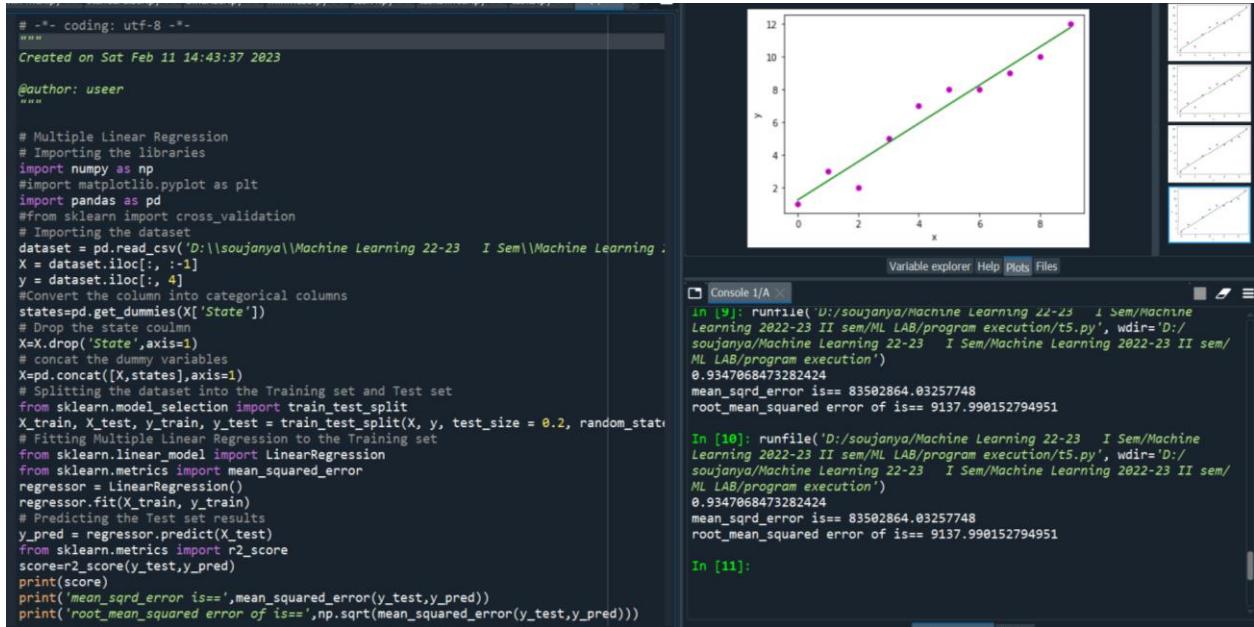
score=r2_score(y_test,y_pred)

print(score)

print('mean_sqrd_error is==',mean_squared_error(y_test,y_pred))

print('root_mean_squared error of is==',np.sqrt(mean_squared_error(y_test,y_pred)))
```

Output: Multiple Linear Regression



TASK-6

Task-6: Develop Logistic Regression Model for a given dataset.

Aim: Develop a python program for Logistic Regression

Program:

```
import pandas as pd  
  
col_names = ['pregnant', 'glucose', 'bp', 'skin', 'insulin', 'bmi', 'pedigree', 'age', 'label']  
  
# load dataset  
  
pima = pd.read_csv("D:\soujanya\Machine Learning 22-23 I Sem\Machine Learning 2022-23 II  
sem\ML LAB/diabetes.csv", header=None, names=col_names)  
  
print(pima)  
  
feature_cols = ['pregnant', 'insulin', 'bmi', 'age','glucose','bp']  
  
X = pima[feature_cols] # Features  
  
y = pima.label # Target variable  
  
print(X)  
  
print(y)  
  
# split X and y into training and testing sets  
  
from sklearn.model_selection import train_test_split  
  
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.25,random_state=0)  
  
# import the class  
  
from sklearn.linear_model import LogisticRegression  
  
# instantiate the model (using the default parameters)  
  
logreg = LogisticRegression(random_state=0, solver='lbfgs')  
  
# fit the model with data  
  
logreg.fit(X_train,y_train)  
  
y_pred=logreg.predict(X_test)  
  
# import the metrics class  
  
from sklearn import metrics  
  
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)  
  
print(cnf_matrix)  
  
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))  
print("Precision:",metrics.precision_score(y_test, y_pred))
```

Output:

The screenshot shows the Spyder Python IDE interface. On the left is a code editor with several tabs at the top: task1.py, task4.py, task5linear.py, task3.py, t5.py, logisticregression.py (which is the active tab), decision trees.py, and task8.py. The code in the editor is for a logistic regression model using the Pima Indian dataset. On the right is a console window titled 'Console 1/A' showing the execution of the script. A 'Usage' help box is open, providing information on how to get help for objects. The console output includes the creation timestamp, author information, dataset loading, feature selection, model instantiation, training, prediction, and evaluation metrics (Accuracy: 0.8020833333333334 and Precision: 0.75). The status bar at the bottom indicates 'In [17]:'.

```
# -*- coding: utf-8 -*-
"""
Created on Sat Apr  1 13:00:28 2023

@author: useer
"""

import pandas as pd
col_names = ['pregnant', 'glucose', 'bp', 'skin', 'insulin', 'bmi', 'pedigree', 'age',
'Label']
# load dataset
pima = pd.read_csv("D:\soujanya\Machine Learning 22-23\I Sem\Machine Learning 2022-23\pima.csv")
print(pima)
feature_cols = ['pregnant', 'insulin', 'bmi', 'age', 'glucose', 'bp']
X = pima[feature_cols] # Features
y = pima.label # Target variable
print(X)
print(y)
# split X and y into training and testing sets
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.25,random_state=0)
# import the class
from sklearn.linear_model import LogisticRegression
# instantiate the model (using the default parameters)
logreg = LogisticRegression(random_state=0, solver='lbfgs')
# fit the model with data
logreg.fit(X_train,y_train)
y_pred=logreg.predict(X_test)
# import the metrics class
from sklearn import metrics
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
print(cnf_matrix)
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print("Precision:",metrics.precision_score(y_test, y_pred))
```

Usage

Here you can get help of any object by pressing **Ctrl+H** in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Help*.

New to Spyder? Read our [tutorial](#)

Variable explorer Help Plots Files

In [17]:

	label
1	0
2	1
3	0
4	1
..	
763	0
764	0
765	0
766	1
767	0

Name: label, Length: 768, dtype: int64
[[118 12]
 [26 36]]
Accuracy: 0.8020833333333334
Precision: 0.75

TASK-7

Task-7: Develop Decision Tree Classification model for a given dataset and use it to classify a new sample.

Aim: To Construct Decision Tree for Classification of any data set.

Program:

```
import os  
  
import numpy as np  
  
import pandas as pd  
  
import numpy as np, pandas as pd  
  
import matplotlib.pyplot as plt  
  
from sklearn import tree, metrics  
  
from sklearn.model_selection import train_test_split  
  
import pylab  
  
data=pd.read_csv('D:\soujanya\Machine Learning 22-23 I Sem\Machine Learning 2022-23 II sem\ML LAB\cardata.csv',names=['buying','maint','doors','persons','lug_boot','safety','class'])  
  
print(data)  
  
data.head()  
  
data.info()  
  
data['class'],class_names = pd.factorize(data['class'])  
  
print(class_names)  
  
print(data['class'].unique())  
  
data['buying'],_ = pd.factorize(data['buying'])  
  
data['maint'],_ = pd.factorize(data['maint'])  
  
data['doors'],_ = pd.factorize(data['doors'])  
  
data['persons'],_ = pd.factorize(data['persons'])  
  
data['lug_boot'],_ = pd.factorize(data['lug_boot'])  
  
data['safety'],_ = pd.factorize(data['safety'])  
  
print(data)
```

```

data.head()

data.info()

X = data.iloc[:, :-1]

y = data.iloc[:, -1]

# implementing train-test-split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)

# train the decision tree

dtree = tree.DecisionTreeClassifier(criterion='entropy', max_depth=3, random_state=0)

dtree.fit(X_train, y_train)

#use the model to make predictions with the test data

y_pred = dtree.predict(X_test)

# how did our model perform?

cnf_matrix = metrics.confusion_matrix(y_test, y_pred)

print(cnf_matrix)

count_misclassified = (y_test != y_pred).sum()

print('Misclassified samples: {}'.format(count_misclassified))

accuracy = metrics.accuracy_score(y_test, y_pred)

print('Accuracy: {:.2f}'.format(accuracy))

```

Output:

The screenshot shows the Spyder IDE interface with the code editor, variable explorer, plots, and files tabs. The code editor contains the provided Python script. The variable explorer shows the data frame structure with columns: buying, maint, doors, persons, lug_boot, safety, and class. The plots tab shows a confusion matrix plot. The files tab lists several other Python files. The right side of the interface displays the usage documentation for the current object.

```

import pandas as pd
import numpy as np, pandas as pd
import matplotlib.pyplot as plt
from sklearn import tree, metrics
from sklearn.model_selection import train_test_split
import pylab
data=pd.read_csv('D:\soujanya\Machine Learning 22-23\I Sem\Machine Learning 2022-23\car.data')
print(data)
data.head()
data.info()
data['class'].class_names = pd.factorize(data['class'])
print(class_names)
print(data['class'].unique())
data['buying']_- = pd.factorize(data['buying'])
data['maint']_- = pd.factorize(data['maint'])
data['doors']_- = pd.factorize(data['doors'])
data['persons']_- = pd.factorize(data['persons'])
data['lug_boot']_- = pd.factorize(data['lug_boot'])
data['safety']_- = pd.factorize(data['safety'])
print(data)
data.head()
data.info()
X = data.iloc[:, :-1]
y = data.iloc[:, -1]
# Implementing train-test-split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
# Train the decision tree
dtree = tree.DecisionTreeClassifier(criterion='entropy', max_depth=3, random_state=0)
dtree.fit(X_train, y_train)
# Use the model to make predictions with the test data
y_pred = dtree.predict(X_test)
# How did our model perform?
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
print(cnf_matrix)
count_misclassified = (y_test != y_pred).sum()
print('Misclassified samples: {}'.format(count_misclassified))
accuracy = metrics.accuracy_score(y_test, y_pred)
print('Accuracy: {:.2f}'.format(accuracy))

```

TASK-8

Task-8: Implement Naïve Bayes Classification in Python

Aim: To develop a python program for Naïve Bayes Classification Model for a given dataset.

Program:

```
import numpy as np
import pandas as pd
from sklearn.naive_bayes import GaussianNB
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score
#from sklearn import cross_validation
play_tennis = pd.read_csv("D:\soujanya\Machine Learning 22-23 I Sem\Machine Learning
2022-23 II sem\ML LAB\PlayTennis.csv")
print(play_tennis.head())
number = LabelEncoder()
play_tennis['Outlook'] = number.fit_transform(play_tennis['Outlook'])
play_tennis["Temperature"] = number.fit_transform(play_tennis['Temperature'])
play_tennis['Humidity'] = number.fit_transform(play_tennis['Humidity'])
play_tennis['Wind'] = number.fit_transform(play_tennis['Wind'])
play_tennis['Play Tennis'] = number.fit_transform(play_tennis['Play Tennis'])
features = ["Outlook", "Temperature", "Humidity", "Wind"]
target = "Play Tennis"
print(play_tennis.head())
from sklearn.model_selection import train_test_split
features_train, features_test, target_train, target_test =
train_test_split(play_tennis[features], play_tennis[target], test_size = 0.33, random_state = 54)
model = GaussianNB()
model.fit(features_train, target_train)
pred = model.predict(features_test)
accuracy = accuracy_score(target_test, pred)
print(accuracy)
print(model.predict([[2,1,0,1]]))
```

Output:

The screenshot shows the Spyder IDE interface with the following components:

- Editor:** Displays the Python script `task8.py` containing code for a Gaussian Naive Bayes classifier. The code reads a CSV file, performs feature scaling, splits the data into training and testing sets, trains a GaussianNB model, and prints the accuracy.
- Usage Help:** A floating window titled "Usage" provides information on how to get help for objects in the Editor or Console.
- Console:** Shows the command-line output of the script's execution. It includes the command run, the current working directory, and the resulting confusion matrix and accuracy score.
- Variable Explorer:** A small window at the bottom right showing available variables: Outlook, Temperature, Humidity, Wind, Play, Tennis, and pred.

```
# -*- coding: utf-8 -*-
"""
Created on Mon Apr  3 14:23:33 2023
@author: useer
"""

import numpy as np
import pandas as pd
from sklearn.naive_bayes import GaussianNB
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score
# from sklearn import cross_validation
play_tennis = pd.read_csv("D:/soujanya/Machine Learning 22-23 I Sem/Machine Learning 22-23 II sem/ML LAB/program execution/task8.py", wdir='D:/soujanya/Machine Learning 22-23 I Sem/Machine Learning 22-23 II sem/ML LAB/program execution')
print(play_tennis.head())
number = LabelEncoder()
play_tennis['Outlook'] = number.fit_transform(play_tennis['Outlook'])
play_tennis['Temperature'] = number.fit_transform(play_tennis['Temperature'])
play_tennis['Humidity'] = number.fit_transform(play_tennis['Humidity'])
play_tennis['Wind'] = number.fit_transform(play_tennis['Wind'])
play_tennis['Play Tennis'] = number.fit_transform(play_tennis['Play Tennis'])
features = ['Outlook', 'Temperature', 'Humidity', 'Wind']
target = "Play Tennis"
print(play_tennis.head())
from sklearn.model_selection import train_test_split
features_train, features_test, target_train, target_test = train_test_split(play_tennis, model = GaussianNB()
model.fit(features_train, target_train)
pred = model.predict(features_test)
accuracy = accuracy_score(target_test, pred)
print(accuracy)
print(model.predict([[2,1,0,1]]))

0.8
[0]
```

```
Learning 22-23 II sem/ML LAB/program execution/task8.py', wdir='D:/soujanya/Machine Learning 22-23 I Sem/Machine Learning 22-23 II sem/ML LAB/program execution')
Outlook Temperature Humidity Wind Play Tennis
0 Sunny Hot High Weak No
1 Sunny Hot High Strong Yes
2 Overcast Hot High Weak Yes
3 Rain Mild High Weak Yes
4 Rain Cool Normal Weak Yes
Outlook Temperature Humidity Wind Play Tennis
0 2 1 0 1 0
1 2 1 0 0 0
2 0 1 0 1 1
3 1 2 0 1 1
4 1 0 1 1 1
In [19]:
```

TASK-9

Task-9: Build KNN Classification model for a given dataset.

Aim: Develop a python program for KNN algorithm to classify Iris data set.

Program:

```
from sklearn.datasets import load_iris  
  
from sklearn.neighbors import KNeighborsClassifier  
  
import numpy as np  
  
from sklearn.model_selection import train_test_split  
  
iris_dataset=load_iris()  
  
print("\n IRIS FEATURES \ TARGET NAMES: \n ", iris_dataset.target_names)  
  
for i in range(len(iris_dataset.target_names)):  
  
    print("\n[{}]:[{}]\n".format(i,iris_dataset.target_names[i]))  
  
#print("\n IRIS DATA :\n",iris_dataset["data"])  
  
X_train, X_test, y_train, y_test = train_test_split(iris_dataset["data"], iris_dataset["target"],  
random_state=0)  
  
#print("\n Target :\n",iris_dataset["target"])  
  
#print("\n X TRAIN \n", X_train)  
  
#print("\n X TEST \n", X_test) print("\n Y TRAIN \n", y_train)  
  
#print("\n Y TEST \n", y_test)  
  
kn = KNeighborsClassifier(n_neighbors=1)  
  
kn.fit(X_train, y_train)  
  
x_new = np.array([[5, 2.9, 1, 0.2]])  
  
print("\n XNEW \n",x_new)  
  
prediction = kn.predict(x_new)  
  
print("\n Predicted target value: {} \n".format(prediction))  
  
print("\n Predicted feature name: {} \n".format(iris_dataset["target_names"][prediction]))  
  
i=1  
  
x= X_test[i]  
  
x_new = np.array([x])  
  
print("\n XNEW \n",x_new)
```

```

for i in range(len(X_test)):

    x = X_test[i]

    x_new = np.array([x])

    prediction = kn.predict(x_new)

    print("\n Actual : {0} Predicted
:{2}{3}".format(y_test[i],iris_dataset["target_names"][y_test[i]],prediction,iris_dataset["target_name
s"][prediction]))

    print("\n TEST SCORE[ACCURACY]: {:.2f}\n".format(kn.score(X_test, y_test)))

```

Output:-

The screenshot shows the Spyder IDE interface with the following details:

- Code Editor:** Displays the Python script for K-Nearest Neighbors classification.
- Console:**
 - Shows the output of the script execution.
 - Prints the target names: 'setosa', 'versicolor', 'virginica'.
 - Prints the X TRAIN and Y TRAIN data.
 - Prints the X TEST and Y TEST data.
 - Prints the XNEW sample: [5. 2.9 1. 0.2].
 - Prints the Predicted target value: 'setosa'.
 - Prints the Predicted feature name: 'setosa'.
 - Prints the XNEW sample again: [6. 2.2 4. 1.]
 - Prints the Actual class: 'versicolor' and Predicted class: 'virginica'.
 - Prints the TEST SCORE[ACCURACY]: 0.97.
- Help:** A pop-up window titled "Usage" provides information on how to get help for objects in the Editor or Console.

TASK-10

Task-10: Implement Back propagation model in python for a given dataset.

Aim: Develop a Python program to build the Back propagation algorithm and test the same using appropriate data sets.

Program:

```
import numpy as np

X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)
y = np.array(([92], [86], [89]), dtype=float)

X = X/np.amax(X, axis=0) # maximum of X array longitudinally
y = y/100 #Sigmoid Function

def sigmoid(x): return(1/(1 + np.exp(-x))) #Derivative of Sigmoid Function
def derivatives_sigmoid(x): return(x * (1 - x)) #Variable initialization

epoch=7000 #Setting training iterations
lr=0.1 #Setting learning rate

inputlayer_neurons = 2 #number of features in data set
hiddenlayer_neurons = 3 #number of hidden layers neurons
output_neurons = 1 #number of neurons at output layer
#weight and bias initialization

wh=np.random.uniform(size=(inputlayer_neurons,hiddenlayer_neurons))
bh=np.random.uniform(size=(1,hiddenlayer_neurons))
wout=np.random.uniform(size=(hiddenlayer_neurons,output_neurons))
bout=np.random.uniform(size=(1,output_neurons))

#draws a random range of numbers uniformly of dim x*y
for i in range(epoch):
    #Forward Propogation
    hinp1=np.dot(X,wh)
    hinp=hinp1 + bh
    hlayer_act = sigmoid(hinp)
    outinp1=np.dot(hlayer_act,wout)
    outinp= outinp1+ bout
    output = sigmoid(outinp) #Backpropagation
    EO = y-output
    outgrad = derivatives_sigmoid(output)
```

```

d_output = EO* outgrad

EH = d_output.dot(wout.T)

hiddengrad = derivatives_sigmoid(hlayer_act)#how much hidden layer wts

d_hiddenlayer = EH * hiddengrad

wout += hlayer_act.T.dot(d_output) *lr # dotproduct of nextlayererror and currentlayerop # bout +=

np.sum(d_output, axis=0,keepdims=True)Python program to implement Boosting ensemble

method*lr

wh += X.T.dot(d_hiddenlayer) *lr

#bh += np.sum(d_hiddenlayer, axis=0,keepdims=True)*lr

print("Input: \n" + str(X))

print("Actual Output: \n" + str(y))

print("Predicted Output: \n" ,output)

```

Output:

The screenshot shows the Spyder IDE interface with the following details:

- Code Editor:** Displays the Python script `Task10.py` containing the provided code for a neural network.
- Usage Help:** A floating window titled "Usage" provides information on how to get help for objects in the editor or console.
- Console:** Shows the execution of the script in "Console 1/A".
- Output:**
 - Input:** The input data is shown as a 3x2 matrix:

[0.92]	[0.33333333 0.55555556]
[1.	0.66666667]]

 - Actual Output:** The actual output values are [0.92], [0.86], and [0.89].
 - Predicted Output:** The predicted output values are [0.89539394], [0.87867047], and [0.89550187].

TASK-11

Task-11: a) Implement Random Forest classification method in python for a given dataset.

b) Implement Boosting ensemble method on a given dataset.

Aim: a) Implement Random Forest classification method in python for a given dataset.

Program:

```
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data=pd.read_csv('D:\soujanya\Machine Learning 22-23 I Sem\Machine Learning 2022-23 II sem\ML LAB\Datasets\diabetes.csv',names=names)
print(data)
X =data.drop('class',axis=1)
y = data['class']
from sklearn.model_selection import train_test_split # implementing train-testsplit
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=66)
from sklearn import model_selection
#random forestmodelcreation
rfc =RandomForestClassifier()
rfc.fit(X_train,y_train)
# predictions
rfc_predict = rfc.predict(X_test)
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report,confusion_matrix
rfc_cv_score = cross_val_score(rfc, X, y, cv=10, scoring='roc_auc')
print("==== Confusion Matrix ====")
print(confusion_matrix(y_test,rfc_predict))
print("\n")
print("==== Classification Report ====")
print(classification_report(y_test, rfc_predict))
print("\n")
print("==== All AUC Scores====")
print(rfc_cv_score)
print("\n")
print("==== Mean AUC Score ====")
print("Mean AUC Score - Random Forest: ", rfc_cv_score.mean())
```

Output:

The screenshot shows the Spyder IDE interface with a Jupyter notebook cell. The code implements a Random Forest classifier on a dataset, calculates cross-validation scores, and prints classification reports and AUC scores.

```
# -*- coding: utf-8 -*-
"""
Created on Thu Apr 20 14:19:37 2023
@author: useer
"""

import pandas as pd
from sklearn.ensemble import RandomForestClassifier
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data=pd.read_csv('D:\\soujanya\\Machine Learning 22-23   I Sem\\Machine Learning 2022-23\\Task11a.csv')
print(data)
X = data.drop(['class'],axis=1)
y = data['class']
from sklearn.model_selection import train_test_split # implementing train-testsplit
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
#random forestmodelcreation
rfc =RandomForestClassifier() | rfc.fit(X_train,y_train)
# predictions
rfc_predict = rfc.predict(X_test)
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report,confusion_matrix
rfc_cv_score = cross_val_score(rfc, X, y, cv=10, scoring='roc_auc')
print("== Confusion Matrix ==")
print(confusion_matrix(y_test,rfc_predict))
print('\n')
print("== Classification Report ==")
print(classification_report(y_test, rfc_predict))
print('\n')
print("== ALL AUC Scores==")
print(rfc_cv_score)
print('\n')
print("== Mean AUC Score ==")
print("Mean AUC Score - Random Forest: ", rfc_cv_score.mean())
```

The output section shows the usage help, a table of cross-validation scores, and the final mean AUC score.

Usage

Here you can get help of any object by pressing **Ctrl+H** in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Help*.

New to Spyder? Read our [tutorial](#)

Variable explorer Help Plots Files

Console 1/A

	0	0.82	0.86	0.84	176
1	0.65	0.56	0.60	78	

	accuracy	0.77	macro avg	0.73	0.71	0.72	254
	weighted avg	0.76	0.77	0.77	0.77	0.77	254

== All AUC Scores==

[0.78592593 0.80592593 0.82 0.74748741 0.80962963 0.85407407 0.87703704 0.91481481 0.79538462 0.84730769]

== Mean AUC Score ==

Mean AUC Score - Random Forest: 0.8257507122507123

In [4]:

b) Implement Boosting ensemble method on a given dataset

Aim: Develop a Python program to implement Boosting ensemble method

Program:

```
#importing utility modules
import pandas as pd
import numpy as np
#from sklearn import cross_validation
from sklearn.metrics import mean_squared_error
# importing machine learning models for prediction
from sklearn.ensemble import GradientBoostingRegressor
col_names = ['pregnant', 'glucose', 'bp', 'skin', 'insulin', 'bmi', 'pedigree', 'age', 'label'] # load dataset
pima = pd.read_csv("D:\soujanya\Machine Learning 22-23 I Sem\Machine Learning 2022-23 II sem\ML LAB\Datasets/diabetes.csv", header=None, names=col_names)
print(pima)
feature_cols = ['pregnant', 'insulin', 'bmi', 'age','glucose','bp']
X = pima[feature_cols] # Features
y = pima.label # Target variable
# Splitting between train data into training and validation dataset
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20) # initializing the boosting module with default parameters
model = GradientBoostingRegressor()
# training the model on the train dataset
model.fit(X_train, y_train)
# predicting the output on the test dataset
y_pred = model.predict(X_test)
# printing the root mean squared error between real value and predicted value
from sklearn.metrics import r2_score
print('score==',r2_score(y_test,y_pred))
print('mean_sqrd_error is==',mean_squared_error(y_test,y_pred))
print('root_mean_squared error of is==',np.sqrt(mean_squared_error(y_test,y_pred)))
```

Output:

The screenshot shows a Jupyter notebook cell in the Spyder IDE. The code imports necessary libraries (utility modules, pandas, numpy, cross-validation, mean_squared_error, GradientBoostingRegressor), reads a CSV file containing a dataset with columns like pregnant, glucose, bp, skin, insulin, bmi, pedigree, age, and label, splits the data into training and testing sets, trains a GradientBoostingRegressor, and calculates various performance metrics (r2_score, mean_squared_error, root_mean_squared_error).

```
decision trees.py task8.py l5.py task9.py Task10.py Task11a.py Task12.py Task11b.py
Thu Apr 20 14:51:36 2023
:eer

utility modules
ias as pd
y as np
rn import cross_validation
n.metrics import mean_squared_error
; machine learning models for prediction
n.ensemble import GradientBoostingRegressor
:[['pregnant', 'glucose', 'bp', 'skin', 'insulin', 'bmi', 'pedigree', 'age', 'label']]
ead_csv("D:\soujanya\Machine Learning 22-23 I Sem\Machine Learning 2022-23 II sem\
)
is = ['pregnant', 'insulin', 'bmi', 'age','glucose','bp']
ature_cols] # Features
bel # Target variable
; between train data into training and validation dataset
n.model_selection import train_test_split
,test, y_train, y_test = train_test_split(X, y, test_size=0.20) # initializing the bo
idientBoostingRegressor()
the model on the train dataset
_,train, y_train)

g the output on the test dataset
odel.predict(X_test)
the root mean squared error between real value and predicted value
n.metrics import r2_score
r==',r2_score(y_test,y_pred)
_sqrd_error is== ,mean_squared_error(y_test,y_pred))
_mean_squared_error of is==',np.sqrt(mean_squared_error(y_test,y_pred)))
```

Usage

Here you can get help of any object by pressing **Ctrl+I** in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Help*.

New to Spyder? Read our [tutorial](#)

Variable explorer Help Plots Files

Console 1/A

	pregnant	glucose	bp	skin	insulin	bmi	pedigree	age	label
0	6	148	72	35	0	34	1	50	1
1	1	85	66	29	0	27	0	31	0
2	8	183	64	0	0	23	1	32	1
3	1	89	66	23	94	28	0	21	0
4	0	137	40	35	168	43	2	33	1
..
763	10	101	76	48	180	33	0	63	0
764	2	122	70	27	0	37	0	27	0
765	5	121	72	23	112	26	0	38	0
766	1	126	60	0	0	30	0	47	1
767	1	93	70	31	0	30	0	23	0

[768 rows x 9 columns]
score== 0.347234610297636
mean_sqrd_error is== 0.14986960477860392
root_mean_squared_error of is== 0.3871299585134221

In [8]:

TASK-12

Task-12: a) Write a python program to implement K-Means Clustering Algorithm

b) Write a python program to implement BIRCH algorithm.

a) Aim: Develop a Python program to implement K-means Clustering Algorithm

Program:

```
from sklearn.cluster import KMeans
from sklearn import metrics
import numpy as np
import matplotlib.pyplot as plt

x1 = np.array([3, 1, 1, 2, 1, 6, 6, 6, 5, 6, 7, 8, 9, 8, 9, 9, 8])
x2 = np.array([5, 4, 6, 6, 5, 8, 6, 7, 6, 7, 1, 2, 1, 2, 3, 2, 3])

plt.plot()
plt.xlim([0, 10])
plt.ylim([0, 10])
plt.title('Dataset')
plt.scatter(x1, x2)
plt.show()

# create new plot and data
plt.plot()

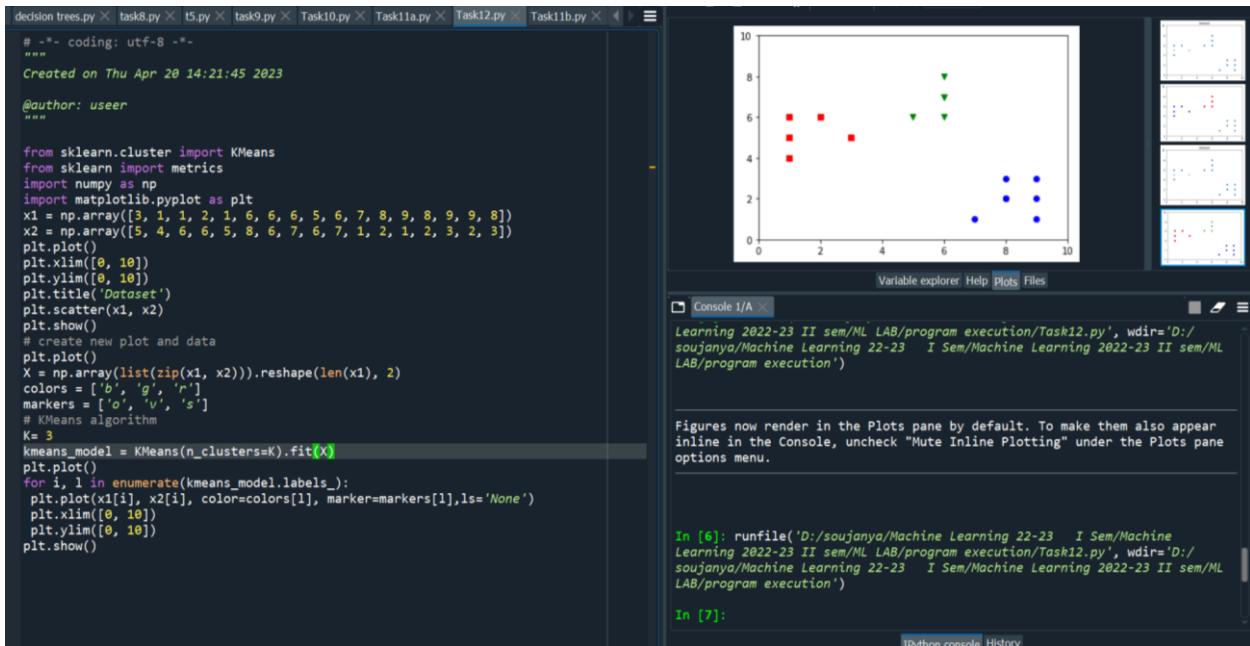
X = np.array(list(zip(x1, x2))).reshape(len(x1), 2)
colors = ['b', 'g', 'r']
markers = ['o', 'v', 's']

# KMeans algorithm
K= 3

kmeans_model = KMeans(n_clusters=K).fit(X)

plt.plot()
for i, l in enumerate(kmeans_model.labels_):
    plt.plot(x1[i], x2[i], color=colors[l], marker=markers[l], ls='None')
plt.xlim([0, 10])
plt.ylim([0, 10])
plt.show()
```

Output:



The screenshot shows a Jupyter Notebook interface. On the left, a code cell contains Python code for K-Means clustering. The code imports necessary libraries, defines two datasets (x1 and x2), and performs K-Means clustering with 3 clusters. On the right, there are three panes: a 'Plots' pane showing a scatter plot of the clustered data points, a 'Variable explorer' pane, and a 'Console' pane displaying the execution command.

```
# -*- coding: utf-8 -*-
"""
Created on Thu Apr 20 14:21:45 2023

@author: useer
"""

from sklearn.cluster import KMeans
from sklearn import metrics
import numpy as np
import matplotlib.pyplot as plt
x1 = np.array([3, 1, 2, 1, 6, 6, 6, 5, 6, 7, 8, 9, 8, 9, 9, 8])
x2 = np.array([5, 4, 6, 6, 5, 8, 6, 7, 6, 7, 1, 2, 1, 2, 3, 2, 3])
plt.plot()
plt.xlim([0, 10])
plt.ylim([0, 10])
plt.title('Dataset')
plt.scatter(x1, x2)
plt.show()
# create new plot and data
plt.plot()
X = np.array(list(zip(x1, x2))).reshape(len(x1), 2)
colors = ['b', 'g', 'r']
markers = ['o', 'v', 's']
# KMeans algorithm
K= 3
kmeans_model = KMeans(n_clusters=K).fit(X)
plt.plot()
for i, l in enumerate(kmeans_model.labels_):
    plt.plot(x1[i], x2[i], color=colors[l], marker=markers[l], ls='None')
plt.xlim([0, 10])
plt.ylim([0, 10])
plt.show()
```

Learning 2022-23 II sem/ML LAB/program execution/Task12.py', wdir='D:/soujanya/Machine Learning 22-23 I Sem/Machine Learning 2022-23 II sem/ML LAB/program execution'

Figures now render in the Plots pane by default. To make them also appear inline in the Console, uncheck "Mute Inline Plotting" under the Plots pane options menu.

In [6]: runfile('D:/soujanya/Machine Learning 22-23 I Sem/Machine Learning 2022-23 II sem/ML LAB/program execution/Task12.py', wdir='D:/soujanya/Machine Learning 22-23 I Sem/Machine Learning 2022-23 II sem/ML LAB/program execution')

In [7]:

b) Aim: Develop a Python program to implement BIRCH algorithm.

Program:

```
# Import required libraries and modules
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.cluster import Birch

# Generating 600 samples using make_blobs
dataset, clusters = make_blobs(n_samples = 600, centers = 8, cluster_std = 0.75, random_state = 0)

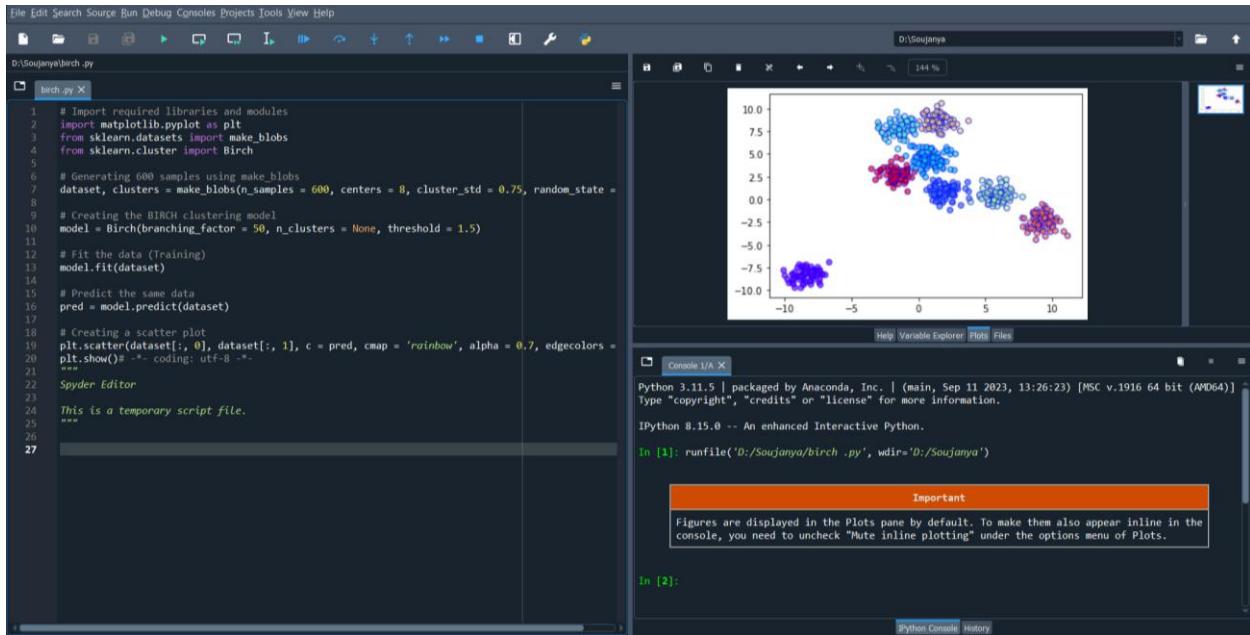
# Creating the BIRCH clustering model
model = Birch(branching_factor = 50, n_clusters = None, threshold = 1.5)

# Fit the data (Training)
model.fit(dataset)

# Predict the same data
pred = model.predict(dataset)

# Creating a scatter plot
plt.scatter(dataset[:, 0], dataset[:, 1], c = pred, cmap = 'rainbow', alpha = 0.7, edgecolors = 'b')
plt.show()
```

Output:



The screenshot shows the Spyder IDE interface with a Python script named `birch.py` open in the editor. The code imports necessary libraries, generates a dataset of 600 samples, creates a BIRCH clustering model, fits it to the data, and then plots the results. The resulting scatter plot displays 600 points colored according to their cluster assignment, forming approximately 8 distinct clusters. The plot is shown in the Plots pane of the IDE.

```
1 # Import required libraries and modules
2 import matplotlib.pyplot as plt
3 from sklearn.datasets import make_blobs
4 from sklearn.cluster import Birch
5
6 # Generating 600 samples using make_blobs
7 dataset, clusters = make_blobs(n_samples = 600, centers = 8, cluster_std = 0.75, random_state =
8
9 # Creating the BIRCH clustering model
10 model = Birch(branching_factor = 50, n_clusters = None, threshold = 1.5)
11
12 # Fit the data (Training)
13 model.fit(dataset)
14
15 # Predict the same data
16 pred = model.predict(dataset)
17
18 # Creating a scatter plot
19 plt.scatter(dataset[:, 0], dataset[:, 1], c = pred, cmap = 'rainbow', alpha = 0.7, edgecolors =
20 plt.show()# -*- coding: utf-8 -*-
21 """
22 Spyder Editor
23
24 This is a temporary script file.
25 """
26
27
```

In [1]: `runfile('D:/Soujanya/birch .py', wdir='D:/Soujanya')`

Important
Figures are displayed in the Plots pane by default. To make them also appear inline in the console, you need to uncheck "Mute inline plotting" under the options menu of Plots.

In [2]: