SCHOOL OF COMPUTING AND INFORMATION TECHNOLOGY

MACHINE LEARNING LABORATORY MANUAL
[B20CI0506]
B.TECH III YEAR – V SEM
[A.Y:2022-2023]

Fifth Semester
B. Tech in AIML /CSIT   /ISE /CSSE

INDEX

		of any dataset. Display the classification results. Also, try the same algorithm to classify the instances for any given dataset	
	Experiment 5	**Regression** Implement and demonstrate linear regression and logistic regression algorithms for any given dataset(s). Visualize the results using graphs. (Salary prediction, Price Prediction)	**31**
	Experiment 6	**Random Forest (RF)** Implement and demonstrate a Random Forest classifier to classify the instances of dataset. Display the classification results. Also, try the same algorithm to classify the instances for any given dataset	41
	Experiment 7	**K-Means Clustering** Implement and demonstrate the k-means clustering algorithms. Visualize the results using graphs.	43
	Experiment 8	**Hierarchical clustering** Implement and demonstrate the hierarchical clustering algorithms. Visualize the results using graphs.	49
	Experiment 9	**DBSCAN clustering** Implement and demonstrate the hierarchical clustering algorithms. Visualize the results using graphs	52
	Experiment 10	**Artificial Neural Networks (ANN)** Implement and demonstrate the two hidden layer multilayer perceptron neural network to any given dataset for classification. Apply two different optimizers or activation functions and compare the results.	57
10	**Additional Questions**		60

VISION OF THE UNIVERSITY

REVA University aspires to become an innovative university by developing excellent human resources with leadership qualities, ethical and moral values, research culture and innovative skills through higher education of global standards

MISSION OF THE UNIVERSITY

- To create excellent infrastructure facilities and state-of-the-art laboratories and incubation centers
- To provide student-centric learning environment through innovative pedagogy and education reforms
- To encourage research and entrepreneurship through collaborations and extension activities
- To promote industry-institute partnerships and share knowledge for innovation and development
- To organize society development programs for knowledge enhancement in thrust areas
- To enhance leadership qualities among the youth and enrich personality traits, promote patriotism and moral values

Program: B.Tech in CSE [Artificial Intelligence and Machine Learning]

VISION OF THE SCHOOL

To produce excellent quality technologists and researchers of global standards in computing and Information technology who have potential to contribute to the development of the nation and the society with their expertise, skills, innovative problem-solving abilities, strong moral and ethical values.

MSSION OF THE SCHOOL

- To create state of the art computing labs infrastructure and research facilities in information technology.
- To provide student-centric learning environment in Computing and Information technology through innovative pedagogy and education reforms.
- To encourage research, innovation and entrepreneurship in computing and information technology through industry/academia collaborations and extension activities
- Organize programs through club activities for knowledge enhancement in thrust areas of information technology.
- To enhance leadership qualities among the youth and enrich personality traits, promote patriotism, moral and ethical values.

PROGRAMME EDUCATIONAL OBJECTIVES (PEOs)

After successful completion of the program, the graduates will be able to

PEO1 : Demonstrate technical skills, competency in AI & ML and exhibit team management capability with proper communication in a job environment.

PEO2: Support the growth of economy of a country by starting enterprise with a lifelong learning attitude.

PEO3: Carry out research in the advanced areas of AI & ML and address the basic needs of the society.

PROGRAM SPECIFIC OUTCOMES (PSOs)

At the end of the program, the student should be able to:

PSO-1 Demonstrate the knowledge of human cognition, Artificial Intelligence, Machine Learning and data engineering for designing intelligent systems.

PSO-2 Apply computational knowledge and project development skills to provide innovative solutions.

PSO-3 Use tools and techniques to solve problems in AI & ML.

PROGRAM OUTCOMES (POs)

Engineering Graduates will be able to:

1. Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

2. Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

3. Design / development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

4. Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

5. Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

6. The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

7. Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

8. Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

9. Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

10. Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

11. Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multi disciplinary environments.

12. Life- long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

1. Course Objectives:

The objectives of this course are to:

1. Explain the mathematical foundation for the analysis of algorithms.
2. Illustrate the algorithms using brute force and divide and conquer designtechnique.
3. Make use of greedy and dynamic algorithmic design techniques for a givenproblem.
4. Discuss the problems based on backtracking and branch and boundtechniques.

2. Lab Requirements:

Following are the required hardware and software for this lab, which is available in the laboratory.

Minimum System requirements:
- Processors: Intel Atom® processor or Intel® Core™ i3 processor.
- Disk space: 1 GB.
- Operating systems: Windows* 7 or later, macOS, and Linux.
- Python* versions: 2.7.X, 3.6.X.,3.8.X

About the Lab:

Python is a general purpose, high-level programming language; other high level languages you might have heard of C++, PHP, Java and Python. Virtually all modern programming languages make us of an Integrated Development Environment (IDE), which allows the creation, editing, testing, and saving of programs and modules. In Python, the IDE is called IDLE (like many items in the language, this is a reference to the British comedy group Monty Python, and in this case, one of its members, Eric Idle). Many modern languages use both processes. They are first compiled into a lower level language, called byte code, and then interpreted by a program called a virtual machine. Python uses both processes, but because of the way programmers interact with it, it is usually considered an interpreted language. Practical aspects are the key to understanding and conceptual visualization of Theoretical aspects covered in the books. Also, this course is designed to review the concepts of Data Structure , studied in previous semesterand implement the various algorithms related to different data structures.

3. Guidelines to Students

➢ Equipment in the lab for the use of student community. Students need to maintain a proper decorum in the computer lab. Students must use the equipment with care. Any damage is caused is punishable.

➢ Students are required to carry their observation / programs book with completed exercises while entering the lab.

➢ Students are supposed to occupy the machines allotted to them and are not supposed to talk or make noise in the lab. The allocation is put up on the lab notice board.

➢ Lab can be used in free time / lunch hours by the students who need to use the systemsshould take prior permission from the lab in-charge.

➢ Lab records need to be submitted on or before date of submission.

➢ Students are not supposed to use flash drives.

Instructions to maintain the record

• Before start of the first lab they have to buy the record and bring the record to the lab.

• Regularly (Weekly) update the record after completion of the experiment and get it corrected with concerned lab in-charge for continuous evaluation.

• In case the record is lost inform the same day to the faculty in charge and get the new record within 2 days the record has to be submitted and get it corrected by the faculty.

• If record is not submitted in time or record is not written properly, the evaluation marks (5M) will be deducted.

General laboratory instructions

1. Students are advised to come to the laboratory at least 5 minutes before (to the starting time), those who come after 5 minutes will not be allowed into the lab.

2. Plan your task properly much before to the commencement, come prepared to the lab with the synopsis / program / experiment details.

3. Student should enter into the laboratory with: a. Laboratory observation notes with all the details (Problem statement, Aim, Algorithm, Procedure, Program, Expected Output, etc.,) filled in for the lab session. b. Laboratory Record updated up to the last session experiments and other utensils (if any) needed in the lab. c. Proper Dress code and Identity card.

4. Sign in the laboratory login register, write the TIME-IN, and occupy the computer system allotted to you by the faculty. 5. Execute your task in the laboratory, and record the results / output in the lab observation note book, and get certified by the concerned faculty.

6. All the students should be polite and cooperative with the laboratory staff, must maintain the discipline and decency in the laboratory.

7. Computer labs are established with sophisticated and high end branded systems, which should be utilized properly.

8. Students / Faculty must keep their mobile phones in SWITCHED OFF mode during the lab sessions. Misuse of the equipment, misbehaviors with the staff and systems etc., will attract severe punishment.

9. Students must take the permission of the faculty in case of any urgency to go out; if anybody found loitering outside the lab / class without permission during working hours will be treated seriously and punished appropriately.

10. Students should LOG OFF/ SHUT DOWN the computer system before he/she leaves the lab after completing the task (experiment) in all aspects. He/she must ensure the system / seat is kept properly.

Course Outcomes

Upon successful completion of this course; student shall be able to:

COs	Course Outcomes
R20A0590.1	Apply the knowledge of mathematical foundation for the analysis of algorithms.
R20A0590.2	Develop a program to solve the given real world problems using brute force and divide and conquerdesign paradigm.
R20A0590.3	Make use of greedy and dynamic programming techniques for solving the given real world problem.
R20A0590.4	Utilize backtracking and branch and bound techniques to solve real world problems.
R20A0590.5	Learn new tools and technologies in the Designing of algorithms and apply for suitable applicationdevelopment.
R20A0590.6	Develop solution to the complex problems, either individually or as a part of the team and report theresults with proper analysis and interpretation.

Conduction of Practical Examination:

1. All laboratory experiments (No. 1 to No. 10) are included for the syllabus of practical examination.
2. Students are allowed to pick one experiment from the lot.
3. Strictly follow the instructions as printed on the cover page of answer script.
4. Marks distribution:

Procedure + Conduction + Viva: 08 + 35 +07 = 50 Marks

Change of experiment is allowed only once and marks allotted to the procedure part to be made zero.

CO-PO-PSO MAPPING

Course	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
R20A0590.1	3	3	3	3	2	3	-	-	-	-	-	3	-	2	-
R20A0590.1	3	3	3	3	2	3	-	-	-	-	-	3	-	2	-
R20A0590.1	3	3	3	3	2	3	-	-	-	-	-	3	-	2	-
R20A0590.1	3	3	3	3	2	3	-	-	-	-	-	3	-	2	-

R20A05 90.1	3	3	3	3	2	3	-	-	-	-	-	3	-	2	-
R20A05 90.1	3	3	3	3	2	3	-	-	-	-	-	3	-	2	-
AVG	3	3	3	3	2	3	-	-	-	-	-	3	-	2	-

WEEKWISE EVALUATION OF EACH PROGRAM

ACTIVITY	MARKS
Observation +Viva	20
Record	10
TOTAL	30

INTERNAL ASSESSMENT EVALUATION (End of Semester)		
Sl No	**ACTIVITY**	**MARKS**
01	Procedure	7
02	Conduction	8
03	Viva Voce	5
	Total	20

FINAL INTERNAL ASSESSMENT CALCULATION		
Sl. No	**ACTIVITY**	**MARKS**
01	Average of weekly Entries	30
02	Internal Assessment Reduced to	20
	Total	50

Program 1

1. Decision Tree Classifier

Implement and demonstrate a Decision Tree Classifier to classify the instances of dataset. Display the classification results. Also, try the same algorithm to classify the instances for any dataset.

1. Data Pre-Processing Step:

```
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd


#importing datasets
data_set= pd.read_csv('user_data.csv')


#Extracting Independent and dependent Variable
x= data_set.iloc[:, [2,3]].values
y= data_set.iloc[:, 4].values


# Splitting the dataset into training 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= 0.25, random_state=0)


#feature Scaling
from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()
x_train= st_x.fit_transform(x_train)
x_test= st_x.transform(x_test)
output:
```

Index	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
5	15728773	Male	27	58000	0
6	15598044	Female	27	84000	0
7	15694829	Female	32	150000	1
8	15600575	Male	25	33000	0
9	15727311	Female	35	65000	0
10	15570769	Female	26	80000	0
11	15606274	Female	26	52000	0
12	15746139	Male	20	86000	0
13	15704987	Male	32	18000	0
14	15628972	Male	18	82000	0
15	15697686	Male	29	80000	0

2. Fitting a Decision-Tree algorithm to the Training set

#Fitting Decision Tree classifier to the training set

From sklearn.tree import DecisionTreeClassifier

classifier= DecisionTreeClassifier(criterion='entropy', random_state=0)

classifier.fit(x_train, y_train)

Output:

Out[8]:

DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=None,

max_features=None, max_leaf_nodes=None,

min_impurity_decrease=0.0, min_impurity_split=None,

min_samples_leaf=1, min_samples_split=2,

min_weight_fraction_leaf=0.0, presort=False,

            random_state=0, splitter='best')

3. Predicting the test result

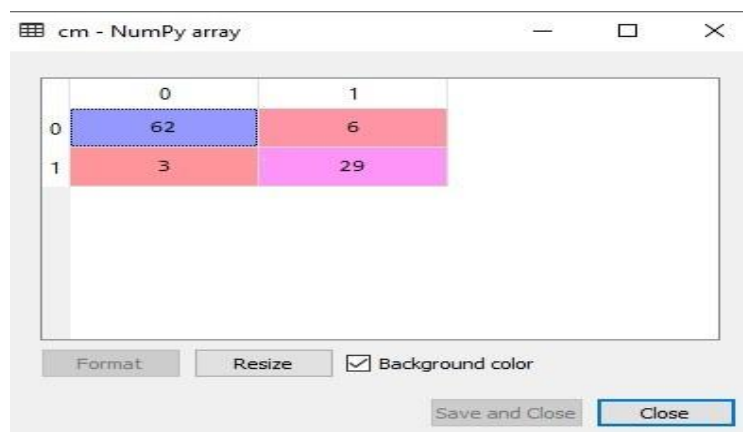#Predicting the test set result
y_pred= classifier.predict(x_test)

Output:

4. Test accuracy of the result (Creation of Confusion matrix)

#Creating the Confusion matrix

from sklearn.metrics import confusion_matrix

cm= confusion_matrix(y_test, y_pred)

Output:

5. Visualizing the training set result:

```
#Visulaizing the trianing set result

from matplotlib.colors import ListedColormap

x_set, y_set = x_train, y_train

x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1,
step =0.01),

nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))

mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(),
x2.ravel()]).T).reshape(x1.shape),

alpha = 0.75, cmap = ListedColormap(('purple','green' )))

mtp.xlim(x1.min(), x1.max())

mtp.ylim(x2.min(),  x2.max())

fori, j in enumerate(nm.unique(y_set)):

mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],

    c = ListedColormap(('purple', 'green'))(i), label = j)

mtp.title('Decision Tree Algorithm (Training set)')

mtp.xlabel('Age')

mtp.ylabel('Estimated Salary')

mtp.legend()

mtp.show()
```
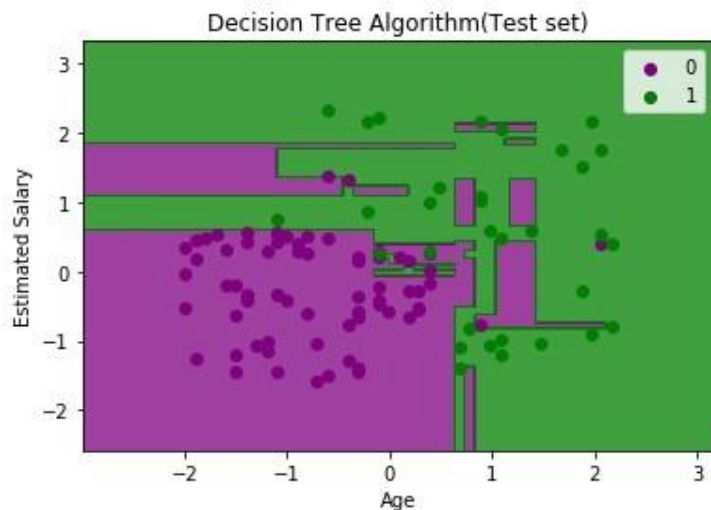
Output:

6. Visualizing the test set result

```
#Visulaizing the test set result

from matplotlib.colors import ListedColormap
```

```
x_set, y_set = x_test, y_test

x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1,
step =0. 01),

nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))

mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(),
x2.ravel()]).T).reshape(x1.shape),

alpha = 0.75, cmap = ListedColormap(('purple','green' )))

mtp.xlim(x1.min(), x1.max())

mtp.ylim(x2.min(),  x2.max())

fori, j in enumerate(nm.unique(y_set)):

mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],

    c = ListedColormap(('purple', 'green'))(i), label = j)

mtp.title('Decision Tree Algorithm(Test set)')

mtp.xlabel('Age')

mtp.ylabel('Estimated Salary')

mtp.legend()

mtp.show()
```

Output:

2. Feature extraction using Principal Component Analysis (PCA)

Implement and demonstrate the Principal Component Analysis algorithm for dimensionality reduction for any dataset

Step 1: We will import the libraries.

```
import numpy as nmp
import matplotlib.pyplot as mpltl
import pandas as pnd
```

Step 2: We will import the dataset (wine.csv)

```
DS = pnd.read_csv('Wine.csv')
# Now, we will distribute the dataset into two components "X" and "Y"
X = DS.iloc[: , 0:13].values
Y = DS.iloc[: , 13].values
```

Step 3: In this step, we will split the dataset into the training set and testing set.

```
from sklearn.model_selection import train_test_split as tts
X_train, X_test, Y_train, Y_test = tts(X, Y, test_size = 0.2, random_state = 0)
```

Step 4: Now, we will Feature Scaling.

```
from sklearn.preprocessing import StandardScaler as SS
SC = SS()

X_train = SC.fit_transform(X_train)
X_test = SC.transform(X_test)
```

Step 5: Then, Apply the PCA function

```
from sklearn.decomposition import PCA
PCa = PCA (n_components = 1)
X_train = PCa.fit_transform(X_train)
X_test = PCa.transform(X_test)
explained_variance = PCa.explained_variance_ratio_
```

Step 6: Now, we will fit Logistic Regression for the training set

```
from sklearn.linear_model import LogisticRegression as LR

classifier_1 = LR (random_state = 0)
classifier_1.fit(X_train, Y_train)
```

Output:
LogisticRegression(random_state=0)

Step 7: Here, we will predict the testing set result:
Y_pred = classifier_1.predict(X_test))

Step 8: We will create the confusion matrix
from sklearn.metrics import confusion_matrix as CM
cm = CM (Y_test, Y_pred)

Step 9: Then, predict the result of the training set:
```
from matplotlib.colors import ListedColormap as LCM
X_set, Y_set = X_train, Y_train
X_1, X_2 = nmp.meshgrid(nmp.arange(start = X_set[:, 0].min() - 1,
          stop = X_set[: , 0].max() + 1, step = 0.01),
          nmp.arange(start = X_set[: , 1].min() - 1,
          stop = X_set[: , 1].max() + 1, step = 0.01))

mpltl.contourf(X_1,  X_2, classifier_1.predict(nmp.array([X_1.ravel(),
       X_2.ravel()]).T).reshape(X_1.shape), alpha = 0.75,
       cmap = LCM (('yellow', 'grey', 'green')))

mpltl.xlim  (X_1.min(),  X_1.max())
mpltl.ylim  (X_2.min(),  X_2.max())

for s, t in enumerate(nmp.unique(Y_set)):
   mpltl.scatter(X_set[Y_set == t, 0], X_set[Y_set == t, 1],
          c = LCM (('red', 'green', 'blue'))(s), label = t)

mpltl.title('Logistic Regression for Training set: ')
mpltl.xlabel ('PC_1') # for X_label
mpltl.ylabel ('PC_2') # for Y_label
mpltl.legend() # for showing legend

# show scatter plot
mpltl.show()
```
Output:

Logistic Regression for Training set:
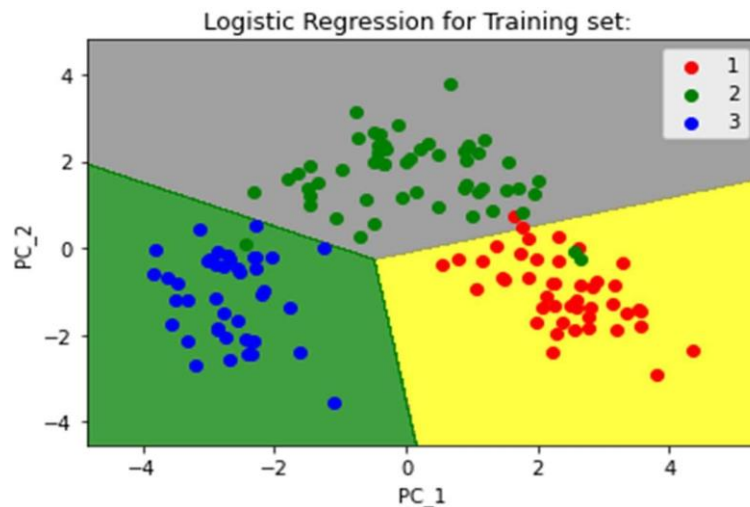
Step 10: At last, we will visualize the result of the testing set

from matplotlib.colors import ListedColormap as LCM

```
X_set, Y_set = X_test, Y_test

X_1, X_2 = nmp.meshgrid(nmp.arange(start = X_set[: , 0].min() - 1,
           stop = X_set[: , 0].max() + 1, step = 0.01),
           nmp.arange(start = X_set[: , 1].min() - 1,
           stop = X_set[: , 1].max() + 1, step = 0.01))

mpltl.contourf(X_1, X_2, classifier_1.predict(nmp.array([X_1.ravel(),
       X_2.ravel()]).T).reshape(X_1.shape), alpha = 0.75,
       cmap = LCM(('pink', 'grey', 'aquamarine')))

mpltl.xlim(X_1.min(),  X_1.max())
mpltl.ylim(X_2.min(),  X_2.max())

for s, t in enumerate(nmp.unique(Y_set)):
   mpltl.scatter(X_set[Y_set == t, 0], X_set[Y_set == t, 1],
         c = LCM(('red', 'green', 'blue'))(s), label = t)

# title for scatter plot
mpltl.title('Logistic Regression for Testing set')
mpltl.xlabel ('PC_1') # for X_label
mpltl.ylabel ('PC_2') # for Y_label
mpltl.legend()

# show scatter plot
mpltl.show()
```

output:

Logistic Regression for Testing set
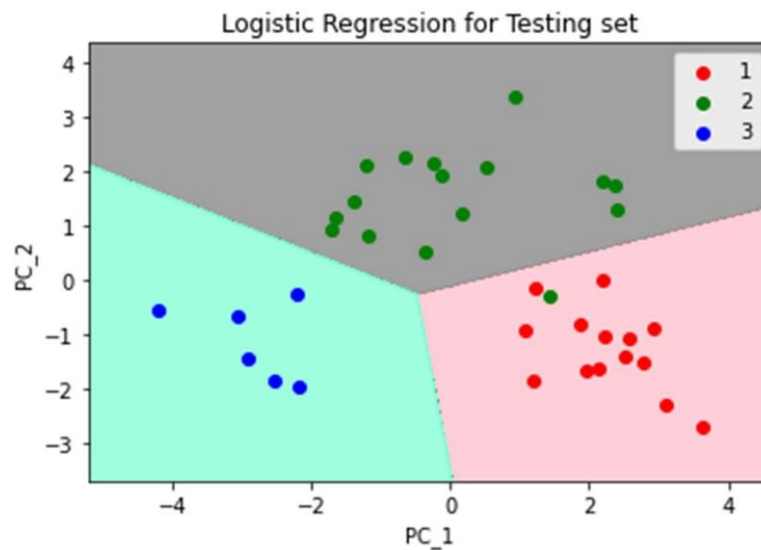
3. K nearest neighbour (KNN)

Implement and demonstrate the k-Nearest Neighbour algorithm (k-NN) to classify the iris data set. Display the Confusion matrix and classification report. Also, try the same algorithm of the social networks dataset to predict a customer can purchase an item or not.

Problem for K-NN Algorithm: The dataset contains lots of information but the **Estimated Salary** and **Age** we will consider for the independent variable and the **Purchased variable** is for the dependent variable. Below is the dataset:

User ID	Gender	Age	EstimatedSalary	Purchased
15624510	Male	19	19000	0
15810944	Male	35	20000	0
15668575	Female	26	43000	0
15603246	Female	27	57000	0
15804002	Male	19	76000	0
15728773	Male	27	58000	0
15598044	Female	27	84000	0
15694829	Female	32	150000	1
15600575	Male	25	33000	0
15727311	Female	35	65000	0
15570769	Female	26	80000	0
15606274	Female	26	52000	0
15746139	Male	20	86000	0
15704987	Male	32	18000	0
15628972	Male	18	82000	0
15697686	Male	29	80000	0
15733883	Male	47	25000	1
15617482	Male	45	26000	1
15704583	Male	46	28000	1
15621083	Female	48	29000	1
15649487	Male	45	22000	1
15736760	Female	47	49000	1

Steps to implement the K-NN algorithm:

o   Data Pre-processing step

o   Fitting the K-NN algorithm to the Training set

o   Predicting the test result

o   Test accuracy of the result(Creation of Confusion matrix)

o   Visualizing the test set result.

Step 1: Data Pre-Processing Step:

The Data Pre-processing step will remain exactly the same as Logistic Regression. Below is the code for it:

```
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd

#importing datasets
data_set= pd.read_csv('user_data.csv')

#Extracting Independent and dependent Variable
```

```
x= data_set.iloc[:, [2,3]].values
y= data_set.iloc[:, 4].values

# Splitting the dataset into training 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= 0.25, random_state=0)

#feature Scaling
from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()
x_train= st_x.fit_transform(x_train)
x_test= st_x.transform(x_test)
```

Ouput:
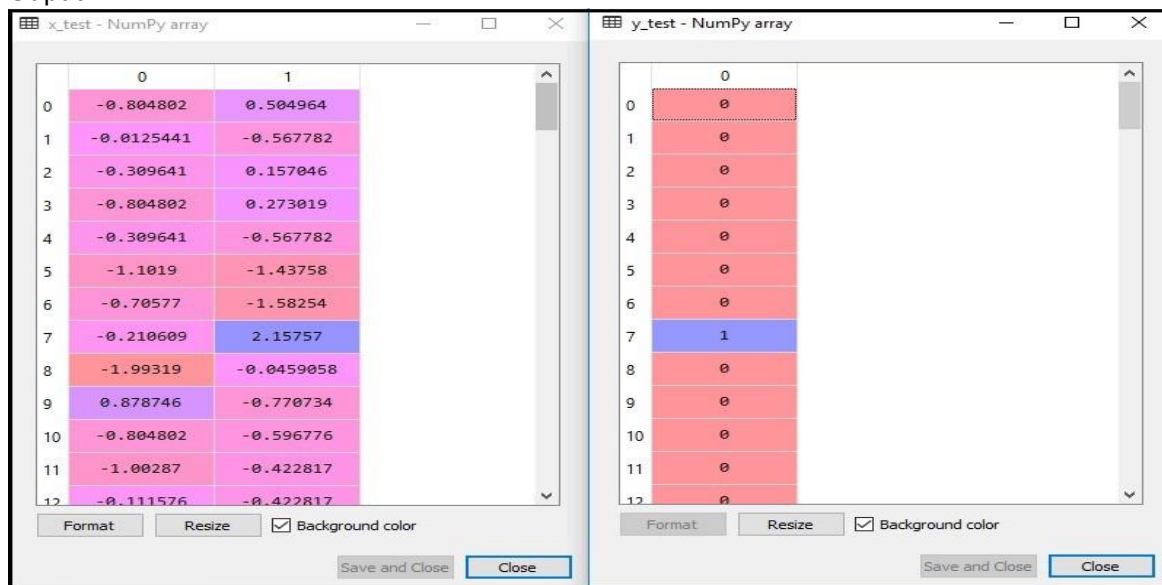
Step 2: Fitting the K-NN algorithm to the Training set

```
#Fitting K-NN classifier to the training set
from sklearn.neighbors import KNeighborsClassifier
classifier= KNeighborsClassifier(n_neighbors=5, metric='minkowski', p=2 )
classifier.fit(x_train, y_train
```

Output:
Out[10]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
          metric_params=None, n_jobs=None, n_neighbors=5, p=2,
          weights='uniform')
```

Step 3: Predicting the test result

#Predicting the test set result
y_pred= classifier.predict(x_test)

output:

Step 4 : **Creating the Confusion Matrix:**

#Creating the Confusion matrix

```
from sklearn.metrics import confusion_matrix
cm= confusion_matrix(y_test, y_pred)
```

Output:
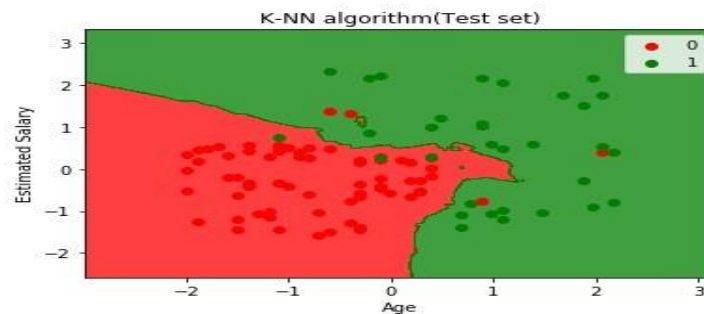
Step 5: **Visualizing the Training set result**

```
#Visulaizing the trianing set result
from matplotlib.colors import ListedColormap
x_set, y_set = x_train, y_train
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, step
=0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(x1,            x2,            classifier.predict(nm.array([x1.ravel(),
x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('red','green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(),  x2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
      c = ListedColormap(('red', 'green'))(i), label = j)
mtp.title('K-NN Algorithm (Training set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

Output:

Step 6: **Visualizing the Test set result:**

```
#Visualizing the test set result
from matplotlib.colors import ListedColormap
x_set, y_set = x_test, y_test
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, step
=0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(x1,            x2,            classifier.predict(nm.array([x1.ravel(),
x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('red','green' )))
mtp.xlim(x1.min(), x1.max())
```

```
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
   mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
      c = ListedColormap(('red', 'green'))(i), label = j)
mtp.title('K-NN algorithm(Test set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend ()
mtp.show()
```

output:

4. Support Vector Machine (SVM)
Implement and demonstrate a Support vector machine classifier to classify the instances of any dataset. Display the classification results. Also, try the same algorithm to classify the instances for any given dataset.

Python Implementation of Support Vector Machine

Now we will implement the SVM algorithm using Python. Here we will use the same dataset **user_data**, which we have used in Logistic regression and KNN classification.

Step 1: Data Pre-processing step
Till the Data pre-processing step, the code will remain the same. Below is the code:
```
#Data Pre-processing Step
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd

#importing datasets
data_set= pd.read_csv('user_data.csv')
```

#Extracting Independent and dependent Variable
x= data_set.iloc[:, [2,3]].values
y= data_set.iloc[:, 4].values

# Splitting the dataset into training 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= 0.25, random_state=0)
#feature Scaling
from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()
x_train= st_x.fit_transform(x_train)
x_test= st_x.transform(x_test)

output:

Index	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
5	15728773	Male	27	58000	0
6	15598044	Female	27	84000	0
7	15694829	Female	32	150000	1
8	15600575	Male	25	33000	0
9	15727311	Female	35	65000	0
10	15570769	Female	26	80000	0
11	15606274	Female	26	52000	0
12	15746139	Male	20	86000	0
13	15704987	Male	32	18000	0
14	15628972	Male	18	82000	0

The scaled output for the test set will be

x_test - NumPy array

	0	1
0	-0.804802	0.504964
1	-0.0125441	-0.567782
2	-0.309641	0.157046
3	-0.804802	0.273019
4	-0.309641	-0.567782
5	-1.1019	-1.43758
6	-0.70577	-1.58254
7	-0.210609	2.15757
8	-1.99319	-0.0459058
9	0.878746	-0.770734
10	-0.804802	-0.596776
11	-1.00287	-0.422817
12	-0.111576	-0.422817

y_test - NumPy array

	0
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	1
8	0
9	0
10	0
11	0
12	0

Step 2: **Fitting the SVM classifier to the training set:**

from sklearn.svm import SVC # "Support vector classifier"
classifier = SVC(kernel='linear', random_state=0)
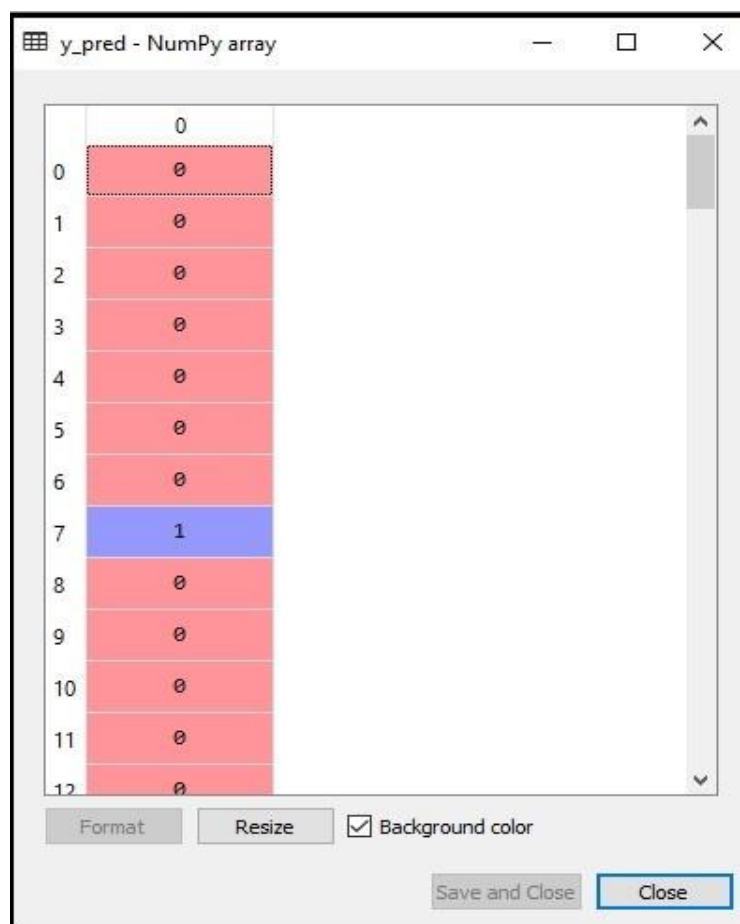classifier.fit(x_train, y_train)

Output:
Out[8]:
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
   decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
   kernel='linear', max_iter=-1, probability=False, random_state=0,
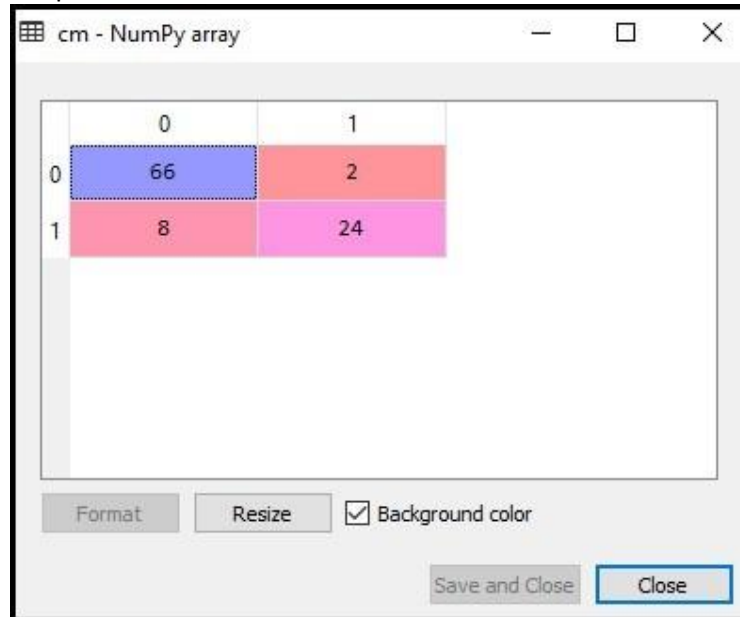   shrinking=True, tol=0.001, verbose=False)

Step 3: **Predicting the test set result**

#Predicting the test set result
y_pred= classifier.predict(x_test)

Step 4 : **Creating the confusion matrix**

#Creating the Confusion matrix
from sklearn.metrics import confusion_matrix
cm= confusion_matrix(y_test, y_pred)

Output:

Step 5: **Visualizing the training set result**

```
from matplotlib.colors import ListedColormap
x_set, y_set = x_train, y_train
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, step
=0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(x1,                    x2,                    classifier.predict(nm.array([x1.ravel(),
x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('red', 'green')))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(),  x2.max())
for i, j in enumerate(nm.unique(y_set)):
  mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
    c = ListedColormap(('red', 'green'))(i), label = j)
mtp.title('SVM classifier (Training set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

output:

SVM classifier (Training set)

Step 5: **Visualizing the test set result**

```
#Visulaizing the test set result
from matplotlib.colors import ListedColormap
x_set, y_set = x_test, y_test
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, step
=0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(x1,                    x2,                    classifier.predict(nm.array([x1.ravel(),
x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('red','green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(),  x2.max())
for i, j in enumerate(nm.unique(y_set)):
   mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
      c = ListedColormap(('red', 'green'))(i), label = j)
mtp.title('SVM classifier (Test set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

Output:

SVM classifier (Test set)
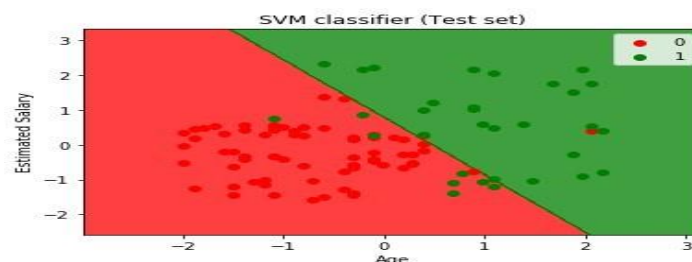
5. Short Title: Regression
Implement and demonstrate linear regression and logistic regression algorithms for any given dataset(s). Visualize the results using graphs.
(Salary prediction, Price Prediction)

Problem Statement example for Simple Linear Regression:

Here we are taking a dataset that has two variables: salary (dependent variable) and experience (Independent variable). The goals of this problem is:

- ○ **We want to find out if there is any correlation between these two variables**
- ○ **We will find the best fit line for the dataset.**
- ○ **How the dependent variable is changing by changing the independent variable.**

Step-1: Data Pre-processing

First, we will import the three important libraries, which will help us for loading the dataset, plotting the graphs, and creating the Simple Linear Regression model.

```
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
```

Next, we will load the dataset into our code:

```
data_set= pd.read_csv('Salary_Data.csv')
```

Index	YearsExperience	Salary
0	1	32383
1	1.1	45207
2	1.3	39751
3	2	43525
4	2.2	39891
5	2.7	56642
6	3	60150
7	3.2	54445
8	3.2	64445
9	3.7	57189
10	3.9	63218
11	4	55794
12	4	56957
13	4.1	57081

```
x= data_set.iloc[:, :-1].values
y= data_set.iloc[:, 1].values
```

output:
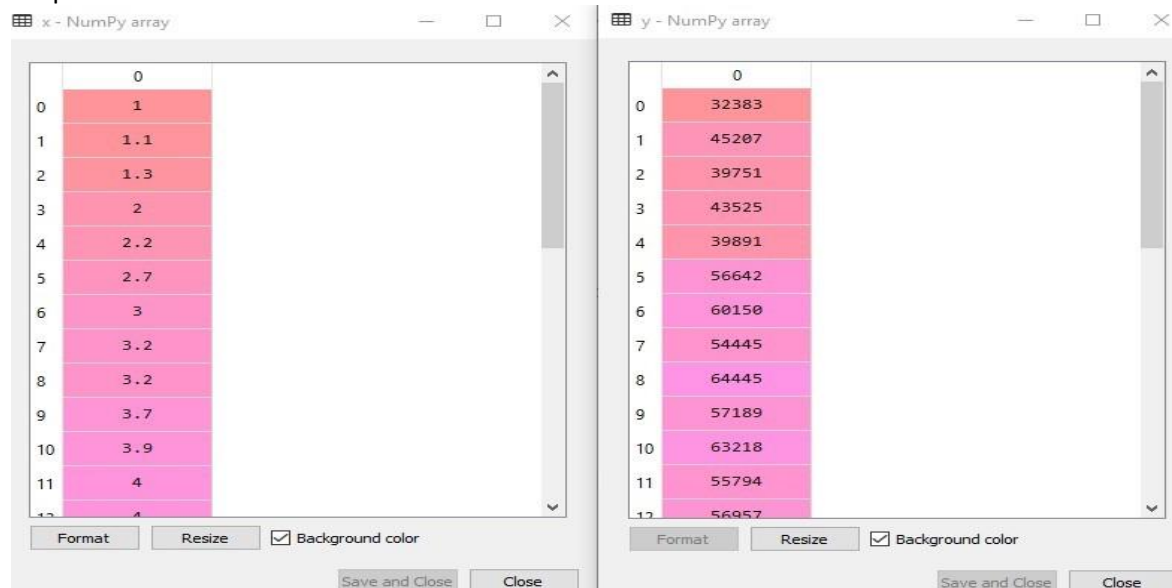
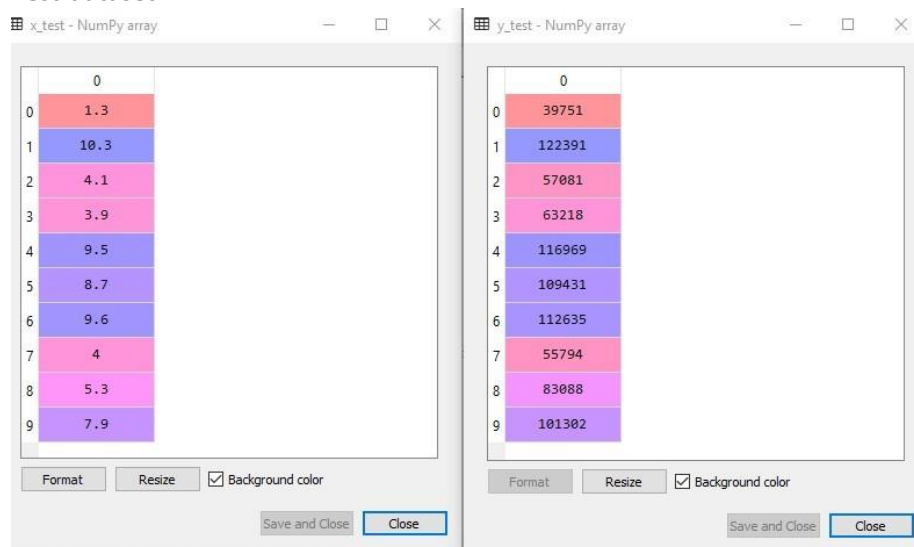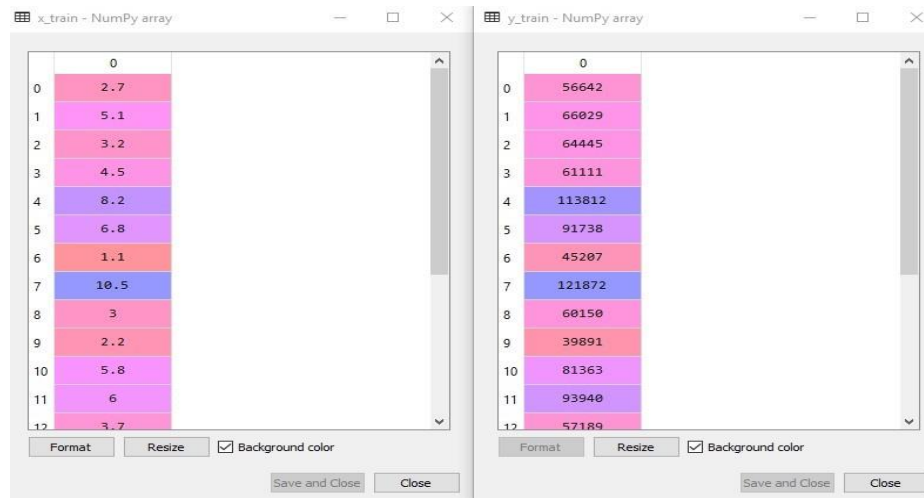# Splitting the dataset into training and test set.

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 1/3, random_state=0)
```

Test-dataset:

Training Dataset:

Step-2: Fitting the Simple Linear Regression to the Training Set

#Fitting the Simple Linear Regression model to the training dataset

from sklearn.linear_model import LinearRegression

regressor= LinearRegression()

regressor.fit(x_train, y_train)

Output:

Out[7]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)

Step: 3. Prediction of test set result:

#Prediction of Test and Training set result

y_pred= regressor.predict(x_test)

x_pred= regressor.predict(x_train)

Step: 4. visualizing the Training set results:

mtp.scatter(x_train, y_train, color="green")

mtp.plot(x_train, x_pred, color="red")

mtp.title("Salary vs Experience (Training Dataset)")

mtp.xlabel("Years of Experience")

mtp.ylabel("Salary(In Rupees)")

mtp.show()

Output:

Step: 5. visualizing the Test set results

#visualizing the Test set results

mtp.scatter(x_test, y_test, color="blue")

mtp.plot(x_train, x_pred, color="red")

mtp.title("Salary vs Experience (Test Dataset)")

mtp.xlabel("Years of Experience")

mtp.ylabel("Salary(In Rupees)")

mtp.show()

Output:

Problem Statement example for Simple logistic Regression:

There is a dataset given which contains the information of various users obtained from the social networking sites. There is a car making company that has recently launched a new SUV car. So the company wanted to check how many users from the dataset, wants to purchase the car.

For this problem, we will build a Machine Learning model using the Logistic regression algorithm. The dataset is shown in the below image. In this problem, we will predict the **purchased variable (Dependent Variable)** by using **age and salary (Independent variables)**.

User ID	Gender	Age	EstimatedSalary	Purchased
15624510	Male	19	19000	0
15810944	Male	35	20000	0
15668575	Female	26	43000	0
15603246	Female	27	57000	0
15804002	Male	19	76000	0
15728773	Male	27	58000	0
15598044	Female	27	84000	0
15694829	Female	32	150000	1
15600575	Male	25	33000	0
15727311	Female	35	65000	0
15570769	Female	26	80000	0
15606274	Female	26	52000	0
15746139	Male	20	86000	0
15704987	Male	32	18000	0
15628972	Male	18	82000	0
15697686	Male	29	80000	0
15733883	Male	47	25000	1
15617482	Male	45	26000	1
15704583	Male	46	28000	1
15621083	Female	48	29000	1
15649487	Male	45	22000	1
15736760	Female	47	49000	1

Steps in Logistic Regression: To implement the Logistic Regression using Python, we will use the same steps as we have done in previous topics of Regression. Below are the steps:

- Data Pre-processing step

- Fitting Logistic Regression to the Training set

- Predicting the test result

- Test accuracy of the result(Creation of Confusion matrix)
- Visualizing the test set result.

Step 1: Data Pre-processing step

In this step, we will pre-process/prepare the data so that we can use it in our code efficiently. It will be the same as we have done in Data pre-processing topic. The code for this is given below:

```
#Data Pre-procesing Step
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd

#importing  datasets
data_set= pd.read_csv('user_data.csv')
```

Index	User ID	Gender	Age	EstimatedSalary	Purchased
92	15809823	Male	26	15000	0
150	15679651	Female	26	15000	0
43	15792008	Male	30	15000	0
155	15610140	Female	31	15000	0
32	15573452	Female	21	16000	0
180	15685576	Male	26	16000	0
79	15655123	Female	26	17000	0
40	15764419	Female	27	17000	0
128	15722758	Male	30	17000	0
58	15642885	Male	22	18000	0
29	15669656	Male	31	18000	0
13	15704987	Male	32	18000	0
74	15592877	Male	32	18000	0
0	15624510	Male	19	19000	0

```
#Extracting Independent and dependent Variable
x= data_set.iloc[:, [2,3]].values
y= data_set.iloc[:, 4].values
```

output:

# Splitting the dataset into training 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= 0.25, random_state=0)

output: for test set

Output **For training set:**

#feature Scaling
from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()
x_train= st_x.fit_transform(x_train)
x_test= st_x.transform(x_test)

Step 2: **Fitting Logistic Regression to the Training set**

#Fitting Logistic Regression to the training set
from sklearn.linear_model import LogisticRegression
classifier= LogisticRegression(random_state=0)
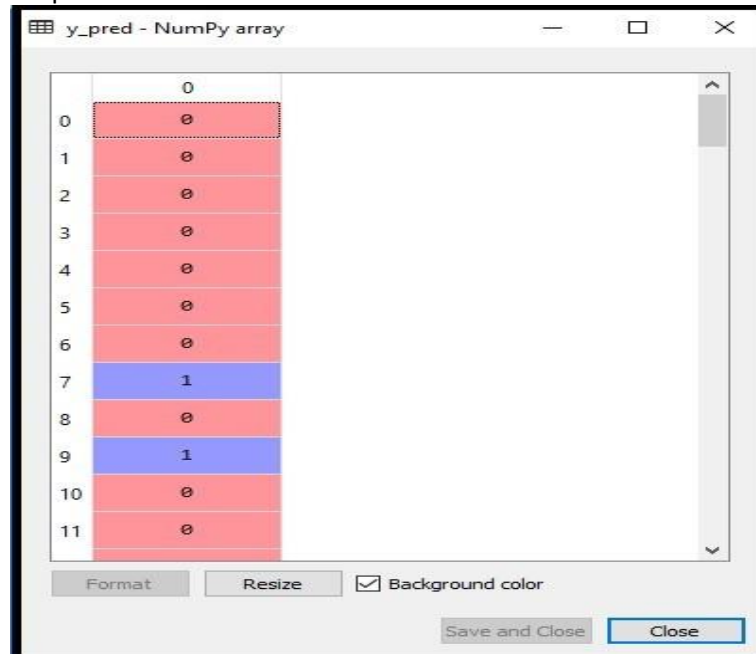
```
classifier.fit(x_train, y_train)
```

Out[5]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, l1_ratio=None, max_iter=100,
          multi_class='warn', n_jobs=None, penalty='l2',
          random_state=0, solver='warn', tol=0.0001, verbose=0,
          warm_start=False)
```

step 3: **Predicting the Test Result**

```
#Predicting the test set result
y_pred=  classifier.predict(x_test)
```
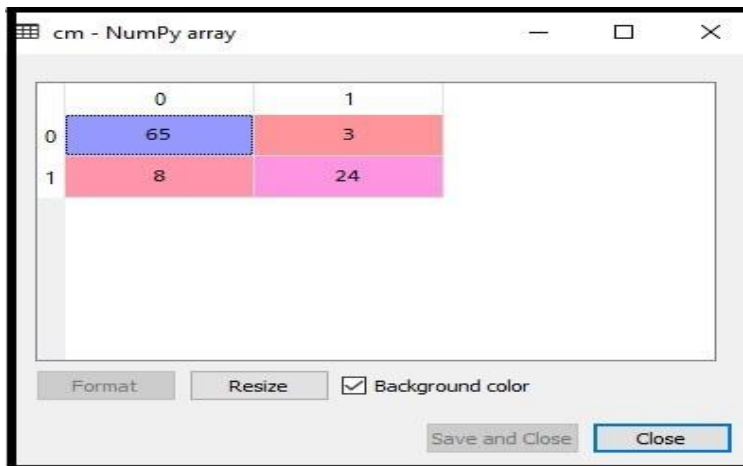
output:

Step 4: **Test Accuracy of the result**

```
#Creating the Confusion matrix
from sklearn.metrics import confusion_matrix
cm= confusion_matrix()
```
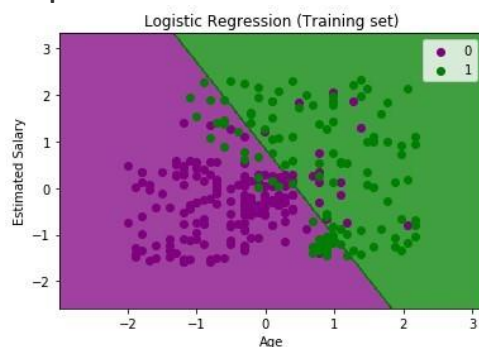
output:

Step 5: **Visualizing the training set result**

#Visualizing the training set result
from matplotlib.colors import ListedColormap
x_set, y_set = x_train, y_train
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, step =0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(x1,   x2,   classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('purple','green')))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
       c = ListedColormap(('purple', 'green'))(i), label = j)
mtp.title('Logistic Regression (Training set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()

Output:

Step 6: **Visualizing the test set result**

```
#Visulaizing the test set result
from matplotlib.colors import ListedColormap
x_set, y_set = x_test, y_test
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, step
=0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(x1,              x2,              classifier.predict(nm.array([x1.ravel(),
x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('purple','green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(),  x2.max())
for i, j in enumerate(nm.unique(y_set)):
   mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
      c = ListedColormap(('purple', 'green'))(i), label = j)
mtp.title('Logistic Regression (Test set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```
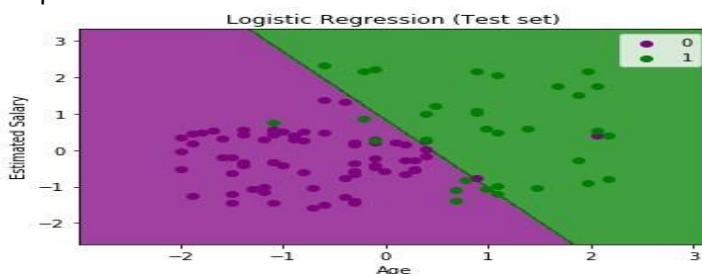
output:

6. Random Forest (RF)
Implement and demonstrate a Random Forest classifier to classify the instances of dataset.
Display the classification results. Also, try the same algorithm to classify the instances for any given dataset.

1. Data Pre-Processing Step:

```
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
```
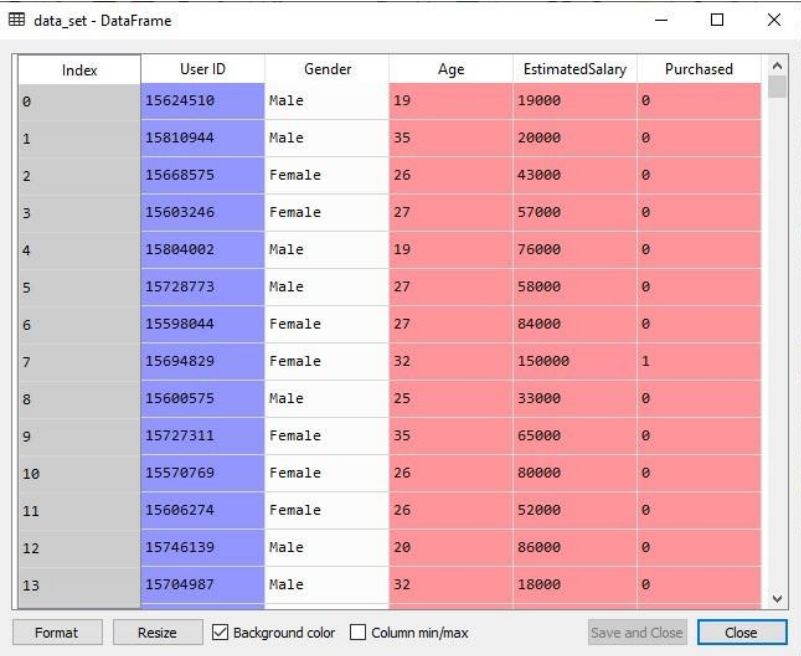
```
#importing datasets
data_set= pd.read_csv('user_data.csv')

#Extracting Independent and dependent Variable
x= data_set.iloc[:, [2,3]].values
y= data_set.iloc[:, 4].values

# Splitting the dataset into training 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= 0.25, random_state=0)

#feature Scaling
from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()
x_train= st_x.fit_transform(x_train)
x_test= st_x.transform(x_test)
```

output:

Index	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
5	15728773	Male	27	58000	0
6	15598044	Female	27	84000	0
7	15694829	Female	32	150000	1
8	15600575	Male	25	33000	0
9	15727311	Female	35	65000	0
10	15570769	Female	26	80000	0
11	15606274	Female	26	52000	0
12	15746139	Male	20	86000	0
13	15704987	Male	32	18000	0

2. Fitting the Random Forest algorithm to the training set:

```
#Fitting Decision Tree classifier to the training set
from sklearn.ensemble import RandomForestClassifier
classifier= RandomForestClassifier(n_estimators= 10, criterion="entropy")
classifier.fit(x_train, y_train)
```
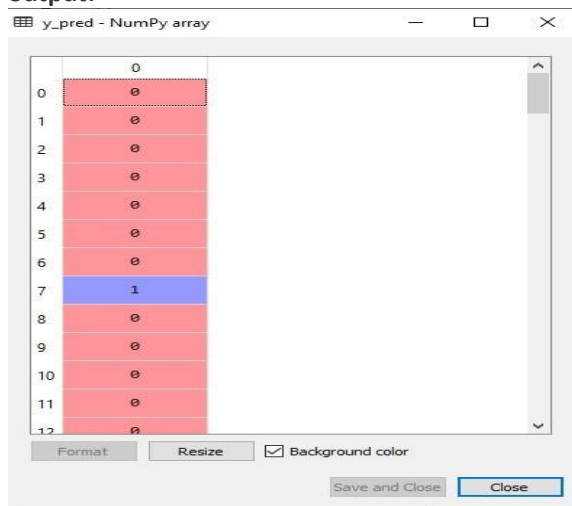
output:
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='entropy',

```
max_depth=None, max_features='auto', max_leaf_nodes=None,
min_impurity_decrease=0.0, min_impurity_split=None,
min_samples_leaf=1, min_samples_split=2,
min_weight_fraction_leaf=0.0, n_estimators=10,
n_jobs=None, oob_score=False, random_state=None,
verbose=0, warm_start=False)
```

3. Predicting the Test Set result

```
#Predicting the test set result
y_pred= classifier.predict(x_test)
```
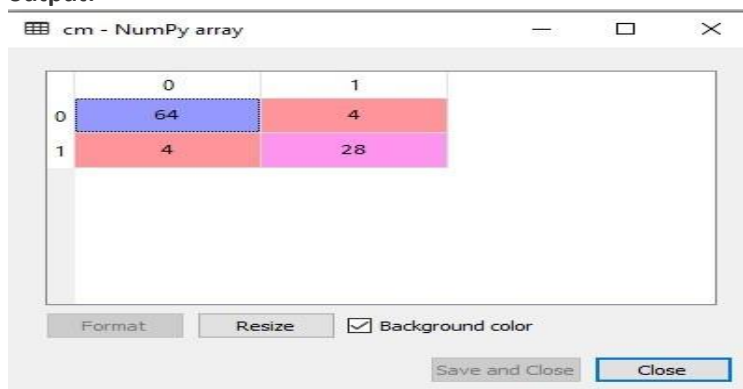
output:

4. Creating the Confusion Matrix

```
#Creating the Confusion matrix
from sklearn.metrics import confusion_matrix
cm= confusion_matrix(y_test, y_pred)
```
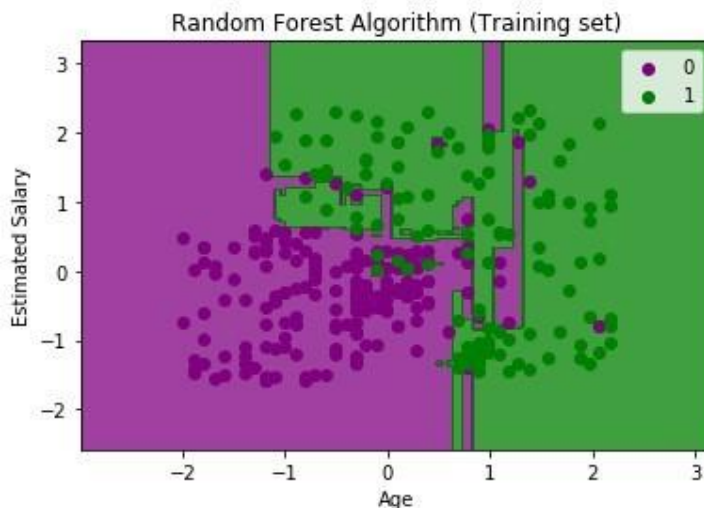
output:

5. Visualizing the training Set result

```
from matplotlib.colors import ListedColormap
x_set, y_set = x_train, y_train
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1,
step =0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(x1,                    x2,                    classifier.predict(nm.array([x1.ravel(),
x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('purple','green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(),  x2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
       c = ListedColormap(('purple', 'green'))(i), label = j)
mtp.title('Random Forest Algorithm (Training set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

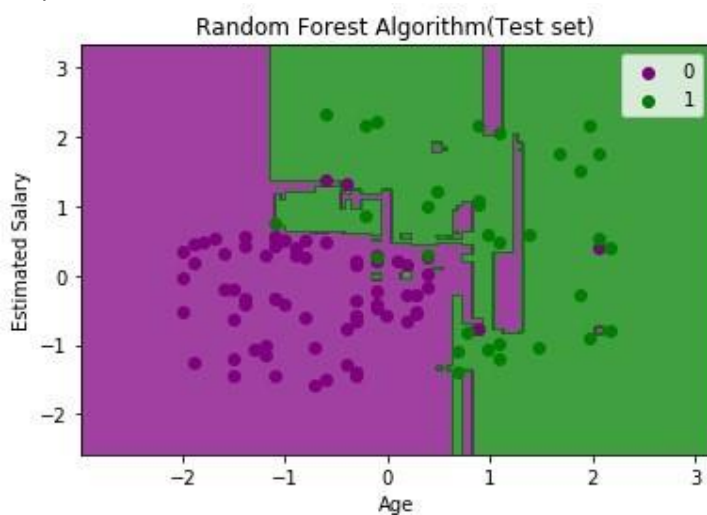output:

6. Visualizing the test set result

```
#Visulaizing the test set result
from matplotlib.colors import ListedColormap
x_set, y_set = x_test, y_test
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1,
step =0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(x1,                    x2,                    classifier.predict(nm.array([x1.ravel(),
```

```
x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('purple','green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(),  x2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('purple', 'green'))(i), label = j)
mtp.title('Random Forest Algorithm(Test set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

output:

6. K-Means Clustering
Implement and demonstrate the k-means clustering algorithms. Visualize the results using graphs.

The steps to be followed for the implementation are given below:

- o **Data Pre-processing**

- o **Finding the optimal number of clusters using the elbow method**

- o **Training the K-means algorithm on the training dataset**

- o **Visualizing the clusters**

Step-1: Data pre-processing Step

# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd

# Importing the dataset
dataset = pd.read_csv('Mall_Customers_data.csv')

output:

Index	CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40
5	6	Female	22	17	76
6	7	Female	35	18	6
7	8	Female	23	18	94
8	9	Male	64	19	3
9	10	Female	30	19	72
10	11	Male	67	19	14
11	12	Female	35	19	99
12	13	Female	58	20	15
13	14	Female	24	20	77
14	15	Male	37	20	13
15	16	Male	22	20	79

Format   Resize   ☑ Background color ☑ Column min/max   Save and Close   Close

Extracting Independent Variables

x = dataset.iloc[:, [3, 4]].values

As we can see, we are extracting only 3rd and 4th feature. It is because we need a 2d plot to visualize the model, and some features are not required, such as customer_id.

Step-2: Finding the optimal number of clusters using the elbow method

#finding optimal number of clusters using the elbow method

from sklearn.cluster import KMeans
wcss_list= [] #Initializing the list for the values of WCSS

#Using for loop for iterations from 1 to 10.
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', random_state= 42)
    kmeans.fit(x)
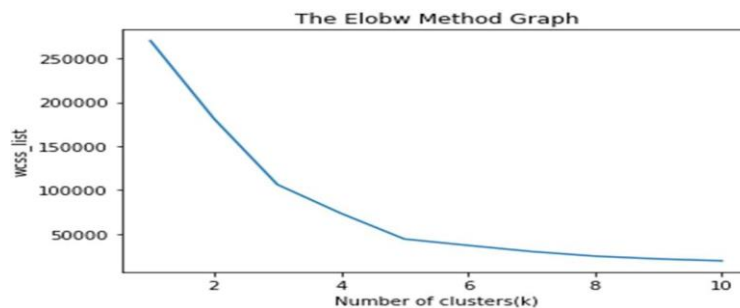   wcss_list.append(kmeans.inertia_)
mtp.plot(range(1, 11), wcss_list)
mtp.title('The Elobw Method Graph')
mtp.xlabel('Number of clusters(k)')
mtp.ylabel('wcss_list')
mtp.show()

output:

Step- 3: Training the K-means algorithm on the training dataset

#training the K-means model on a dataset
kmeans = KMeans(n_clusters=5, init='k-means++', random_state= 42)
y_predict= kmeans.fit_predict(x)
y_predict

Step-4: Visualizing the Clusters

```
#visulaizing the clusters
mtp.scatter(x[y_predict == 0, 0], x[y_predict == 0, 1], s = 100, c = 'blue', label = 'Cluster 1')
#for first cluster
mtp.scatter(x[y_predict == 1, 0], x[y_predict == 1, 1], s = 100, c = 'green', label = 'Cluster 2')
#for second cluster
mtp.scatter(x[y_predict== 2, 0], x[y_predict == 2, 1], s = 100, c = 'red', label = 'Cluster 3')
#for third cluster
mtp.scatter(x[y_predict == 3, 0], x[y_predict == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4')
#for fourth cluster
mtp.scatter(x[y_predict == 4, 0], x[y_predict == 4, 1], s = 100, c = 'magenta', label = 'Cluster
5') #for fifth cluster
mtp.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 300, c =
'yellow', label = 'Centroid')
mtp.title('Clusters of customers')
mtp.xlabel('Annual Income (k$)')
mtp.ylabel('Spending Score (1-100)')
mtp.legend()
mtp.show()
```

output:
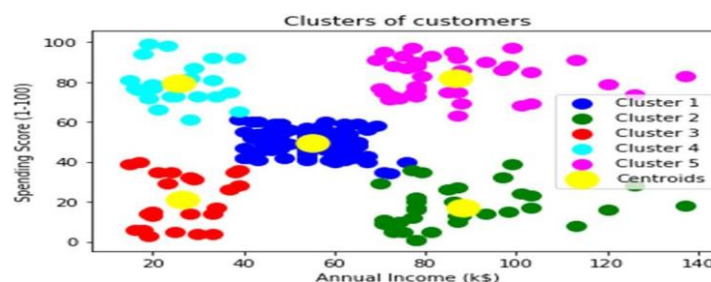
8. Hierarchical clustering

Implement and demonstrate the hierarchical clustering algorithms. Visualize the results using graphs.

Steps for implementation of AHC using Python:

The steps for implementation will be the same as the k-means clustering, except for some changes such as the method to find the number of clusters. Below are the steps:

1. **Data Pre-processing**

2. **Finding the optimal number of clusters using the Dendrogram**

3. **Training the hierarchical clustering model**
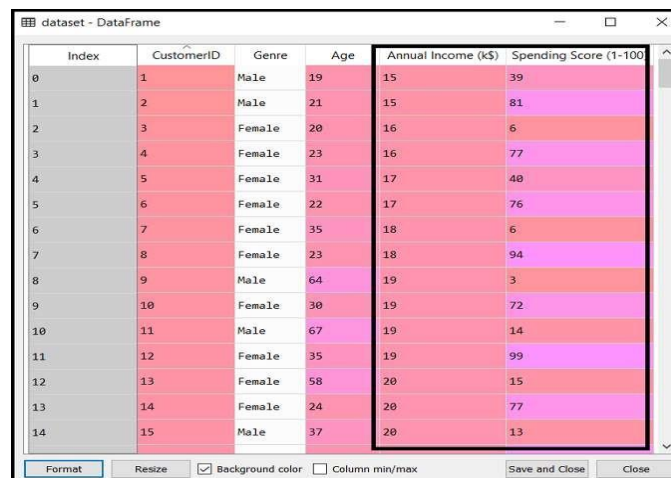
4. **Visualizing the clusters**

Step 1: Data Pre-processing Steps:

Importing the libraries

```
# Importing the libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
```

Importing the dataset

```
# Importing the dataset
dataset = pd.read_csv('Mall_Customers_data.csv')
```

Extracting the matrix of features

x = dataset.iloc[:, [3, 4]].values

Step-2: Finding the optimal number of clusters using the Dendrogram

```
#Finding the optimal number of clusters using the dendrogram
import scipy.cluster.hierarchy as shc
dendro = shc.dendrogram(shc.linkage(x, method="ward"))
mtp.title("Dendrogrma Plot")
mtp.ylabel("Euclidean Distances")
mtp.xlabel("Customers")
mtp.show()
```
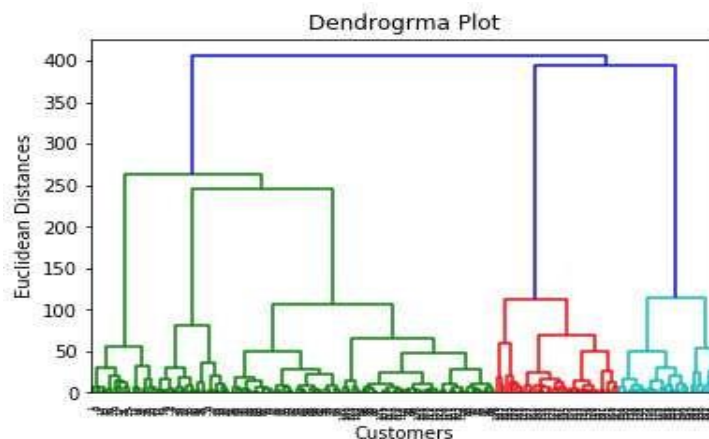
output:

Step-3: Training the hierarchical clustering model

```
#training the hierarchical model on dataset
from sklearn.cluster import AgglomerativeClustering
hc= AgglomerativeClustering(n_clusters=5, affinity='euclidean', linkage='ward')
y_pred= hc.fit_predict(x)
```

output:

Step-4: Visualizing the clusters

```
#visulaizing the clusters
mtp.scatter(x[y_pred == 0, 0], x[y_pred == 0, 1], s = 100, c = 'blue', label = 'Cluster 1')
mtp.scatter(x[y_pred == 1, 0], x[y_pred == 1, 1], s = 100, c = 'green', label = 'Cluster 2')
mtp.scatter(x[y_pred== 2, 0], x[y_pred == 2, 1], s = 100, c = 'red', label = 'Cluster 3')
mtp.scatter(x[y_pred == 3, 0], x[y_pred == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4')
mtp.scatter(x[y_pred == 4, 0], x[y_pred == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5')
mtp.title('Clusters of customers')
mtp.xlabel('Annual Income (k$)')
mtp.ylabel('Spending Score (1-100)')
mtp.legend()
mtp.show()
```

output:
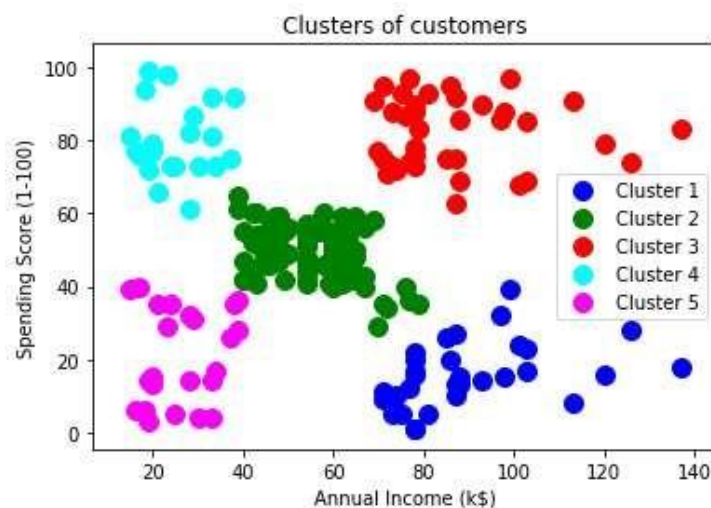
9. DBSCAN clustering
Implement and demonstrate the hierarchical clustering algorithms. Visualize the results using graphs.

We have to follow the following steps in order to implement the DBSCAN algorithm and its logic inside a Python program:

Step 1: Importing all the required libraries:

```
# Importing numpy library as nmp
import numpy as nmp
# Importing pandas library as pds
import pandas as pds
# Importing matplotlib library as pplt
import matplotlib.pyplot as pplt
# Importing DBSCAN from cluster module of Sklearn library
from sklearn.cluster import DBSCAN
# Importing StandardSclaer and normalize from preprocessing module of Sklearn library
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import normalize
# Importing PCA from decomposition module of Sklearn
from sklearn.decomposition import PCA
```

Step 2: Loading the Data:

```
# Loading the data inside an initialized variable
M = pds.read_csv('sampleDataset.csv') # Path of dataset file
# Dropping the CUST_ID column from the dataset with drop() function
M = M.drop('CUST_ID', axis = 1)
# Using fillna() function to handle missing values
M.fillna(method ='ffill', inplace = True)
# Printing dataset head in output
print(M.head())
```

output:
```
   BALANCE  BALANCE_FREQUENCY  ...  PRC_FULL_PAYMENT  TENURE
0    40.900749       0.818182  ...          0.000000      12
1  3202.467416       0.909091  ...          0.222222      12
2  2495.148862       1.000000  ...          0.000000      12
3  1666.670542       0.636364  ...          0.000000      12
4   817.714335       1.000000  ...          0.000000      12

[5 rows x 17 columns]
```

Step 3: Preprocessing the data:

```
# Initializing a variable with the StandardSclaer() function
```

```
scalerFD = StandardScaler()
# Transforming the data of dataset with Scaler
M_scaled = scalerFD.fit_transform(M)
# To make sure that data will follow gaussian distribution
# We will normalize the scaled data with normalize() function
M_normalized = normalize(M_scaled)
# Now we will convert numpy arrays in the dataset into dataframes of panda
M_normalized = pds.DataFrame(M_normalized)
```

Step 4: Reduce the dimensionality of the data:

```
# Initializing a variable with the PCA() function
pcaFD = PCA(n_components = 2) # components of data
# Transforming the normalized data with PCA
M_principal = pcaFD.fit_transform(M_normalized)
# Making dataframes from the transformed data
M_principal = pds.DataFrame(M_principal)
# Creating two columns in the transformed data
M_principal.columns = ['C1', 'C2']
# Printing the head of the transformed data
print(M_principal.head())
```

```
output:
    C1        C2
0 -0.489949 -0.679976
1 -0.519099  0.544828
2  0.330633  0.268877
3 -0.481656 -0.097610
4 -0.563512 -0.482506
```

Step 5: Build a clustering model:
```
# Creating clustering model of the data using the DBSCAN function and providing parameters
in it
db_default = DBSCAN(eps = 0.0375, min_samples = 3).fit(M_principal)
# Labelling the clusters we have created in the dataset
labeling = db_default.labels_
```

**Step 6: Visualize the clustering model:**
```
# Visualization of clustering model by giving different colours
colours = {}
# First colour in visualization is green
colours[0] = 'g'
# Second colour in visualization is black
colours[1] = 'k'
# Third colour in visualization is red
colours[2] = 'r'
# Last colour in visualization is blue
```

```
colours[-1] = 'b'
# Creating a colour vector for each data point in the dataset cluster
cvec = [colours[label] for label in labeling]
# Construction of the legend
# Scattering of green colour
g = pplt.scatter(M_principal['C1'], M_principal['C2'], color ='g');
# Scattering of black colour
k = pplt.scatter(M_principal['C1'], M_principal['C2'], color ='k');
# Scattering of red colour
r = pplt.scatter(M_principal['C1'], M_principal['C2'], color ='r');
# Scattering of green colour
b = pplt.scatter(M_principal['C1'], M_principal['C2'], color ='b');
# Plotting C1 column on the X-Axis and C2 on the Y-Axis
# Fitting the size of the figure with figure function
pplt.figure(figsize =(9, 9))
# Scattering the data points in the Visualization graph
pplt.scatter(M_principal['C1'], M_principal['C2'], c = cvec)
# Building the legend with the coloured data points and labelled
pplt.legend((g, k, r, b), ('Label M.0', 'Label M.1', 'Label M.2', 'Label M.-1'))
# Showing Visualization in the output
pplt.show()
```

output:
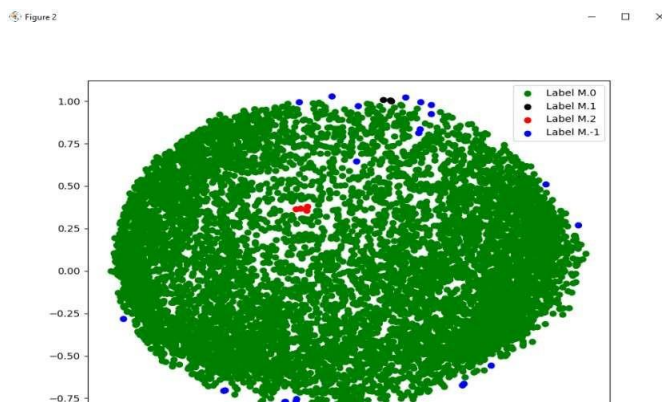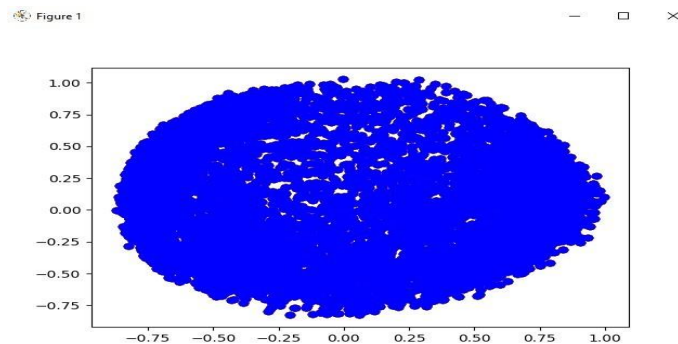
Step 7: Tuning the parameters:

```
# Tuning the parameters of the model inside the DBSCAN function
dts = DBSCAN(eps = 0.0375, min_samples = 50).fit(M_principal)
# Labelling the clusters of data points
labeling = dts.labels_
```

Step 8: Visualization of the changes:

```
# Labelling with different colours
colours1 = {}
# labelling with Red colour
colours1[0] = 'r'
# labelling with Green colour
colours1[1] = 'g'
# labelling with Blue colour
colours1[2] =  'b'
colours1[3] = 'c'
# labelling with Yellow colour
colours1[4] = 'y'
# Magenta colour
colours1[5] = 'm'
# labelling with Black colour
colours1[-1] = 'k'
# Labelling the data points with the colour variable we have defined
cvec = [colours1[label] for label in labeling]
# Defining all colour that we will use
colors = ['r', 'g', 'b', 'c', 'y', 'm', 'k' ]
# Scattering the colours onto the data points
r = pplt.scatter(
    M_principal['C1'], M_principal['C2'], marker ='o', color = colors[0])
g = pplt.scatter(
    M_principal['C1'], M_principal['C2'], marker ='o', color = colors[1])
b = pplt.scatter(
    M_principal['C1'], M_principal['C2'], marker ='o', color = colors[2])
c = pplt.scatter(
    M_principal['C1'], M_principal['C2'], marker ='o', color = colors[3])
y = pplt.scatter(
    M_principal['C1'], M_principal['C2'], marker ='o', color = colors[4])
m = pplt.scatter(
    M_principal['C1'], M_principal['C2'], marker ='o', color = colors[5])
k = pplt.scatter(
    M_principal['C1'], M_principal['C2'], marker ='o', color = colors[6])
# Fitting the size of the figure with figure function
pplt.figure(figsize =(9, 9))
# Scattering column 1 into X-axis and column 2 into y-axis
pplt.scatter(M_principal['C1'], M_principal['C2'], c = cvec)
# Constructing a legend with the colours we have defined
```

```
pplt.legend((r, g, b, c, y, m, k),
        ('Label M.0', 'Label M.1', 'Label M.2', 'Label M.3', 'Label M.4','Label M.5', 'Label M.-1'),
# Using different labels for data points
        scatterpoints = 1, # Defining the scatter point
        loc ='upper left', # Location of cluster scattering
        ncol = 3, # Number of columns
        fontsize = 10) # Size of the font
# Displaying the visualisation of changes in cluster scattering
pplt.show()
```

output:
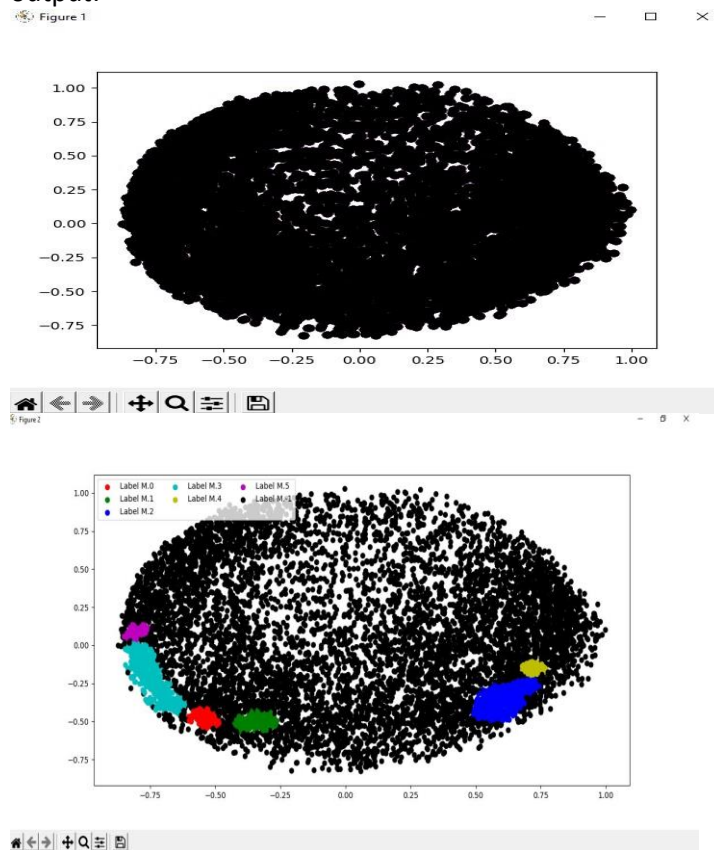
10. Implement and demonstrate the two hidden layer multilayer perceptron neural network to any given dataset for classification. Apply two different optimizers or activation functions and compare the results.

```python
#Importing the essential modules in the hidden layer
import tensorflow as tf
import numpy as np
import mat
plotlib.pyplot as plt
import math, random


np.random.seed(1000)
function_to_learn = lambda x: np.cos(x) + 0.1*np.random.randn(*x.shape)
layer_1_neurons = 10
NUM_points = 1000

#Train the parameters of hidden layer
batch_size = 100
NUM_EPOCHS = 1500

all_x = np.float32(np.random.uniform(-2*math.pi, 2*math.pi, (1, NUM_points))).T
  np.random.shuffle(all_x)

train_size = int(900)
#Train the first 700 points in the set
 x_training = all_x[:train_size]
y_training = function_to_learn(x_training)

#Training the last 300 points in the given set x_validation = all_x[train_size:]
y_validation = function_to_learn(x_validation)

plt.figure(1)
plt.scatter(x_training, y_training, c = 'blue', label = 'train')
plt.scatter(x_validation, y_validation, c = 'pink', label = 'validation')
plt.legend()
plt.show()

X = tf.placeholder(tf.float32, [None, 1], name = "X")
Y = tf.placeholder(tf.float32, [None, 1], name = "Y")

#first layer
#Number of neurons = 10
w_h = tf.Variable(
```

```
    tf.random_uniform([1, layer_1_neurons],\ minval = -1, maxval = 1, dtype = tf.float32))
b_h = tf.Variable(tf.zeros([1, layer_1_neurons], dtype = tf.float32))
h = tf.nn.sigmoid(tf.matmul(X, w_h) + b_h)

#output layer
#Number of neurons = 10
w_o = tf.Variable(
    tf.random_uniform([layer_1_neurons, 1],\ minval = -1, maxval = 1, dtype = tf.float32))
b_o = tf.Variable(tf.zeros([1, 1], dtype = tf.float32))

#building the model
model = tf.matmul(h, w_o) + b_o

#minimize the cost function (model - Y)
train_op = tf.train.AdamOptimizer().minimize(tf.nn.l2_loss(model - Y))

#Starting the Learning phase
sess = tf.Session() sess.run(tf.initialize_all_variables())

errors = []
for i in range(NUM_EPOCHS):
    for start, end in zip(range(0, len(x_training), batch_size),\
        range(batch_size, len(x_training), batch_size)):
        sess.run(train_op, feed_dict = {X: x_training[start:end],\ Y: y_training[start:end]})
    cost = sess.run(tf.nn.l2_loss(model - y_validation),\ feed_dict = {X:x_validation})
    errors.append(cost)

    if i%100 == 0:
        print("epoch %d, cost = %g" % (i, cost))

plt.plot(errors,label='MLP Function Approximation') plt.xlabel('epochs')
plt.ylabel('cost')
plt.legend()
plt.show()
```
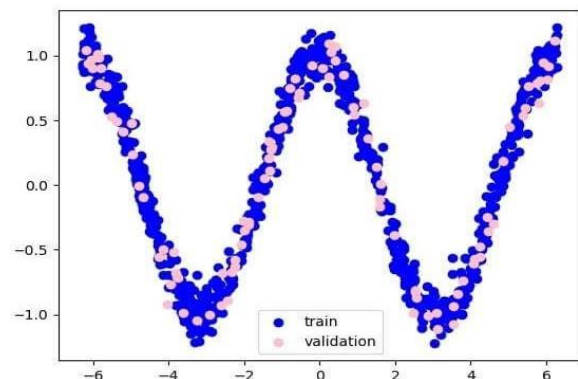
output:

The two data are: **train** and **validation**, which are described in distinct colors as visible in the legend section.

```
Instructions for updating:
Use `tf.global_variables_initializer` instead.
epoch 0, cost = 523.278
epoch 100, cost = 25.7673
epoch 200, cost = 24.9066
epoch 300, cost = 24.7239
epoch 400, cost = 24.3757
epoch 500, cost = 23.6071
epoch 600, cost = 22.3059
epoch 700, cost = 20.4841
epoch 800, cost = 17.8938
epoch 900, cost = 14.1825
epoch 1000, cost = 9.82508
epoch 1100, cost = 5.91628
epoch 1200, cost = 3.18464
epoch 1300, cost = 1.68528
epoch 1400, cost = 1.04172
```

ADDITIONAL QUESTIONS & ANSWERS:

1. What is the difference between a Perceptron and Logistic Regression?

   A Multi-Layer Perception (MLP) is one of the most basic neuralnetworks that we use for classification. For a binary classification problem, we know that the output can be either 0 or 1. This is just like our simple logistic regression, where we use a logit function to generate a probability between 0 and 1. So, what's the difference between the two?

2. What do you understand by Machine learning?

   Machine learning is the form of Artificial Intelligence that deals with system programming and automates data analysis to enable computers to learn and act through experiences without being explicitly programmed.

3. What is the meaning of Overfitting in Machine learning?

   Overfitting can be seen in machine learning when a statistical model describes random error or noise instead of the underlying relationship. Overfitting is usually observed when a model is excessively complex. It happens because of having too many parameters concerning the number of training data types. The model displays poor performance, which has been overfitted.

4. What is the method to avoid overfitting?

   Overfitting occurs when we have a small dataset, and a model is trying to learn from it. By using a large amount of data, overfitting can be avoided. But if we have a small database and are forced to build a model based on that, then we can use a technique known as **cross-validation**. In this method, a model is usually given a dataset of a known data on which training data set is run and dataset of unknown data against which the model is tested. The primary aim of cross-validation is to define a dataset to "test" the model in the training phase. If there is sufficient data, '**Isotonic Regression**' is used to prevent overfitting.

5. Differentiate supervised and unsupervised machine learning.

   o   In supervised machine learning, the machine is trained using labeled data. Then a new dataset is given into the learning model so that the algorithm provides a positive outcome by analyzing the labeled data. For example, we first require to label the data which is necessary to train the model while performing classification.

   o   In the unsupervised machine learning, the machine is not trained using labeled data and let the algorithms make the decisions without any corresponding output variables.

6. What are the different types of Algorithm methods in Machine Learning?

   The different types of algorithm methods in machine earning are:

   o   Supervised Learning

   o   Semi-supervised Learning

- o Unsupervised Learning

- o Transduction

- o Reinforcement Learning

7. What is the trade-off between bias and variance?

Both bias and variance are errors. Bias is an error due to erroneous or overly simplistic assumptions in the learning algorithm. It can lead to the model under-fitting the data, making it hard to have high predictive accuracy and generalize the knowledge from the training set to the test set.

Variance is an error due to too much complexity in the learning algorithm. It leads to the algorithm being highly sensitive to high degrees of variation in the training data, which can lead the model to overfit the data.

To optimally reduce the number of errors, we will need to tradeoff bias and variance.