SCIENT INSTITUTE OF TECHNOLOGY

Ibrahimpatnam, R.R Dist 501506

(NAAC Accredited, Approved by AICTE & Affiliated to JNTUH)

MACHINE LEARNING LAB MANUAL

NAME OF THE FACULTY	: SMD SHAFIULLA
DEPARTMENT	: CSE/CSM/AIML
YEAR & SEM	: III YEAR II SEM
COURSE CODE	: CS604PC
REGULATION	: R22
BRANCH & SECTION	:COMPUTER SCIENCE & ENGINEERING
ACADEMIC YEAR	: 2024-25
NO. OF CREDITS	: 1

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

VISION

The Vision of the department is to produce competent graduates suitable for industries and organizations at global level including research and development with Social responsibility.

MISSION

a) To provide an open environment to foster professional and personal growth with a strong theoretical and practical background.

b) Emphasis on hardware and software development making the graduates industry ready with social ethics.

c) Further the Department is to provide training and to partner with Global entities in education and research.

PROGRAM EDUCATIONAL OBJECTIVES(PEOs)	
PEO1	Students will establish themselves as effective professionals by solving real problems through the use of computer science knowledge and with attention to team work, effective communication, critical thinking and problem solving skills.
PEO2	Students will develop professional skills that prepare them for immediate employment and for life-long learning in advanced areas of computer science and related fields.
PEO3	Students will demonstrate their ability to adapt to a rapidly changing environment by having learned and applied new skills and new technologies
PEO4	Students will be provided with an educational foundation that prepares them for excellence, leadership roles along diverse career paths with encouragement to professional ethics and active participation needed for a successful career.

PROGRAM OUTCOMES(POs)	
PO1	Engineering knowledge: Apply the knowledge of Mathematics, Science, Engineering fundamentals, and an Engineering specialization to the solution of complex Engineering problems.
PO2	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
PO3	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.
PO4	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.
PO5	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
PO6	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.
PO7	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.
PO8	Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
PO9	Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
PO10	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.
PO11	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 multidisciplinary environments.
PO12	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.

PROGRAM SPECIFIC OUTCOMES(PSOs)	
PSO1	Understand, design and analyze computer programs in the areas related to Algorithms, System Software, Web design, Big data, Artificial Intelligence, Machine Learning and Networking.
PSO2	Focus on improving software reliability, network security or information retrieval systems.
PSO3	Make use of modern computer tools for creating innovative career paths, to be an entrepreneur and desire for higher studies

LIST OF EXPERIMENTS

Experiment 1 : Write a python program to compute

Central Tendency Measures: Mean, Median, Mode

Measure of Dispersion: Variance, Standard Deviation

Aim : To develop python program to compute central tendency measures and Measures of dispersion

Software used : Python 3.13.1

Program :

```
import statistics

def compute_central_tendency_and_dispersion(data):
    if not data:
        return "The data list is empty."

    # Central Tendency Measures

    mean = statistics.mean(data)
    median = statistics.median(data)
        try:
        mode = statistics.mode(data)
    except statistics.StatisticsError:
        mode = "No unique mode found"

    # Measure of Dispersion
    variance = statistics.variance(data)
    std_deviation = statistics.stdev(data)

    # Display results
    results = {"Mean": mean, "Median": median, "Mode": mode, "Variance": variance, "Standard Deviation": std_deviation}
    return results
```

```
# Example usage
if __name__ == "__main__":  #main() function definition
    # Sample data
    data = [10, 20, 20, 30, 40, 50, 50, 50, 60]
    results = compute_central_tendency_and_dispersion(data)
    for measure, value in results.items():
        print(f"{measure}: {value}")
```

Output :

Mean: 36.666666666666664

Median: 40

Mode: 50

Variance: 300

Standard Deviation: 17.320508075688775

Experiment 2 : Study of Python Basic Libraries such as Statistics, Math, Numpy and Scipy

Aim : To study and work on python basic libraries such as Statistics . Math , Numpy , Scipy

Software used : Python 3.13.1

Description

Statistics library

Python statistics module provides the functions to mathematical statistics of numeric data.

The four functions are common in statistics library:

1. mean() - average value
2. median() - middle value
3. mode() - most often value
4. standard deviation - spread of value

Mean() function

The mean() function is used to calculate the arithmetic mean of the numbers in the list.

Example

```
import statistics
# list of positive integer numbers
datasets = [5, 2, 7, 4, 2, 6, 8]
x = statistics.mean(datasets)
# Printing the mean
print("Mean is :", x)
```

Median() function

The median() function is used to return the middle value of the numeric data in the list.

Example

```
import statistics
# list of positive integer numbers
datasets = [5, 2, 7, 4, 2, 6, 8]
x = statistics.mean(datasets)
```

```
# Printing the median

print("Median is :", x)
```

Mode() function:

The mode() function returns the most common data that occurs in the list.

Example

```
import statistics

# list of positive integer numbers

datasets = [2, 4, 7, 7, 2, 2, 3, 6, 6, 8]

x = statistics.mode(datasets)

# Printing the mode

print("Mode is :", x)
```

stdev() function

The stdev() function is used to calculate the standard deviation on a given sample which is available in the form of the list.

Example

```
import statistics

# list of positive integer numbers

datasets = [1,0,1,1,1,1]

x = statistics.stdev(datasets)

# Printing the mode

print("Standard deviation is :", x)
```

Maths library:

- Math Module is an in-built Python library made to simplify mathematical tasks in Python.
- It consists of various mathematical constants and functions that can be used after importing the math module.

List of Functions in Python Math Module	
Function	**Description**
ceil(x)	Returns the smallest integer greater than or equal to x.
copysign(x, y)	Returns x with the sign of y
fabs(x)	Returns the absolute value of x
factorial(x)	Returns the factorial of x
floor(x)	Returns the largest integer less than or equal to x
fmod(x, y)	Returns the remainder when x is divided by y
frexp(x)	Returns the mantissa and exponent of x as the pair (m, e)
fsum(iterable)	Returns an accurate floating point sum of values in the iterable
isfinite(x)	Returns True if x is neither an infinity nor a NaN (Not a Number)
isinf(x)	Returns True if x is a positive or negative infinity
isnan(x)	Returns True if x is a NaN
ldexp(x, i)	Returns x * (2**i)
modf(x)	Returns the fractional and integer parts of x
trunc(x)	Returns the truncated integer value of x
exp(x)	Returns e**x
expm1(x)	Returns e**x - 1

Example

```
import math
print("Demonstrating ceil() function")
#Round a number upward to its nearest integer
print(math.ceil(2.1))
print(math.ceil(1.4))
print(math.ceil(-5.3))
print("Demonstrating factorial() function")
#Return factorial of a number
print(math.factorial(2))
print(math.factorial(4))
print(math.factorial(6))
```

```
print("Demonstrating floor() function")
#Round numbers to the nearest integer
print(math.floor(2.6))
print(math.floor(1.2))
print(math.floor(3.2))
print("Demonstrating gcd() function")
#find the greatest common divisor of the two integers
print(math.gcd(3,6))
print(math.gcd(1,2))
print(math.gcd(3,9))

print("Demonstrating isnan() function")
#check whether the number is Not A Number (NaN) or not
print(math.isnan(12))
print(math.isnan(-2))
print(math.isnan(math.nan))

print("Demonstrating sqrt() function")
#calculating squareroot of a value
print(math.sqrt(10))
print(math.sqrt(3))
print(math.sqrt(9))
```

Numpy :

- Numpy is Numerical Python
- NumPy(Numerical Python) is a fundamental library for Python numerical computing.
- It provides efficient multi-dimensional array objects and various mathematical functions for handling large datasets making it a critical tool for professionals in fields that require heavy computation.

Example

```
import numpy as np
# Creating a 1D array
x = np.array([1, 2, 3])
# Creating a 2D array
y = np.array([[1, 2], [3, 4]])
# Creating a 3D array
z = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])
print(x)
print(y)
print(z)
```

Create a NumPy ndarray Object

NumPy is used to work with arrays. The array object in NumPy is called **ndarray.**

We can create a NumPy ndarray object by using the array() function.

Example

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
print(arr)
print(type(arr))
```

SciPy module:

- SciPy stands for Scientific Python.
- It provides more utility functions for optimization, stats and signal processing.
- Like NumPy, SciPy is open source so we can use it freely.
- SciPy was created by NumPy's creator Travis Olliphant.

Once SciPy is installed, import the SciPy module(s) you want to use in your applications by adding the

from scipy **import** *module* statement:

Eg : from scipy import constants

#importing constants module from Scipy

Example

```
from scipy import stats
a= [99,86,87,88,111,86,103,87,94,78,77,85,86]
x = stats.mode(a)
print(x)
```

Example

```
import numpy
a= [86,87,88,86,87,85,86]
x = numpy.std(a)
print(x)
```

Experiment 3 : Study of Python Libraries for ML application such as Pandas and Matplotlib

Aim : To study Python Libraries Pandas and Matplotlib

Pandas:

- Pandas is a Python library used for working with data sets.
- It has functions for analyzing, cleaning, exploring, and manipulating data.
- Pandas allows us to analyze big data and make conclusions based on statistical theories.
- Pandas can clean messy data sets, and make them readable and relevant.
- Relevant data is very important in data science.
- The data produced by Pandas is often used as input for plotting functions in **Matplotlib,** statistical analysis in **SciPy,** and machine learning algorithms in **Scikit-learn.**

Installation of pandas:
To install python use PIP(Preferred Installer Program)
pip install pandas

Once Pandas is installed, import it in your applications by adding the import keyword as below
import pandas as pd
Here, pd is referred to as an alias for the Pandas. However, it is not necessary to import the library using the alias, it just helps in writing less code every time a method or property is called.

Data Structures in Pandas Library
Pandas generally provide two data structures for manipulating data. They are:
a)Series
b)DataFrame
Pandas Series
A Pandas Series is a one-dimensional labeled array capable of holding data of any type (integer, string, float, Python objects, etc.). The axis labels are collectively called **indexes.**
Creating a Series
Pandas Series is created by loading the datasets from existing storage (which can be a SQL database, a CSV file, or an Excel file).
Pandas Series can be created from lists, dictionaries, scalar values, etc.

Program1 :

```
import pandas as pd
a = [1, 2, 3]
x = pd.Series(a)
print(x)
```

Program 2 :

```
import pandas as pd
a = [1.2, 2.1, 3.2]
x = pd.Series(a)
print(x)
```

Program 3 :

```
import pandas as pd
a = [1, 2, 3]
var = pd.Series(a, index = ["x", "y", "z"])
print(var)
```

b)Dataframe:

A Pandas DataFrame is a 2 dimensional data structure, like a 2 dimensional array, or a table with rows and columns.

Program1 :

```
import pandas as pd
data = {"Name": ['siva', 'kiran', 'lucky'],"Marks": [50, 40, 45]}
#load data into a DataFrame object:
df = pd.DataFrame(data)
print(df)
```

Reading CSV files using Pandas

- A simple way to store big data sets is to use CSV files (comma separated files).
- CSV files contains plain text and is a well know format that can be read by everyone including Pandas.
- **to_string()** to print the entire DataFrame.
- **read_csv() is used to read csv file.**

Program 2 :

```
import pandas as pd
df = pd.read_csv('scient.csv')
print(df.to_string())
```

Program 3 :

```
import pandas as pd
df = pd.read_csv('scient.csv')
print(df)
```

MATPLOTLIB

Matplotlib is a low level graph plotting library in python that serves as a visualization utility.

Matplotlib was created by John D. Hunter.

Matplotlib is open source and we can use it freely.

Matplotlib Pyplot

Most of the Matplotlib utilities lies under the pyplot submodule, and are usually imported under the plt alias:

```
import matplotlib.pyplot as plt
```

Program 1 :

```
import matplotlib.pyplot as plt
import numpy as np
xpoints = np.array([0, 6])
ypoints = np.array([0, 250])
plt.plot(xpoints, ypoints)
plt.show()
```

Matplotlib Plotting

Plotting x and y points

- The plot() function is used to draw points (markers) in a diagram.
- By default, the plot() function draws a line from point to point.
- The function takes parameters for specifying points in the diagram.
- Parameter 1 is an array containing the points on the x-axis.
- Parameter 2 is an array containing the points on the y-axis.

Program 2 : Draw a line in a diagram from position (1, 3) to position (10, 20):

```
import matplotlib.pyplot as plt
import numpy as np
xpoints = np.array([1, 10])
ypoints = np.array([3, 20])
plt.plot(xpoints, ypoints)
plt.show()
```

Program 3 : Using markers

```
import matplotlib.pyplot as plt
import numpy as np
xpoints = np.array([1, 8])
ypoints = np.array([3, 10])

plt.plot(xpoints, ypoints)
plt.plot(ypoints, marker = '*')
plt.plot(xpoints, marker = '.')

plt.show()
```

Matplotlib Scatter

Creating Scatter Plots

With Pyplot, you can use the scatter() function to draw a scatter plot.

The scatter() function plots one dot for each observation. It needs two arrays of the same length, one for the values of the x-axis, and one for values on the y-axis

Program 4 :

```
import matplotlib.pyplot as plt
import numpy as np
x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
plt.scatter(x, y)
plt.show()
```

Matplotlib Pie Charts

With Pyplot, you can use the **pie()** function to draw pie charts:

Program 5 :

```
import matplotlib.pyplot as plt
import numpy as np
y = np.array([35, 25, 25, 15])
plt.pie(y)
plt.show()
```

Matplotlib Bars

Creating Bars

With Pyplot, you can use the bar() function to draw bar graphs:

Program 6 :

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

x = np.array(["CSE", "CSM", "AIML", "ECE","EEE"])
y = np.array([288, 120,60 ,60,30])

plt.bar(x,y)
plt.show()
```

Experiment 4 : Write a Python program to implement Simple Linear Regression

Aim : To develop a python program to implement simple linear regression

Software used : Jupyterlab

Description

Linear regression:

- Linear regression is a statistical method used to model the relationship between a dependent variable and one or more independent variables.
- It provides valuable insights for prediction and data analysis.
- Linear regression is also a type of supervised machine-learning algorithm that learns from the labelled datasets and maps the data points with most optimized linear functions which can be used for prediction on new datasets.
- It computes the linear relationship between the dependent variable and one or more independent features by fitting a linear equation with observed data.
- It predicts the continuous output variables based on the independent input variable.

Program:

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt


# To read data from Age_Income.csv file

dataFrame = pd.read_csv('Age_Income.csv')


# To place data in to age and income vectors

age = dataFrame['Age']

income = dataFrame['Income']


# number of points

num = np.size(age)


# To find the mean of age and income vector

mean_age = np.mean(age)
```

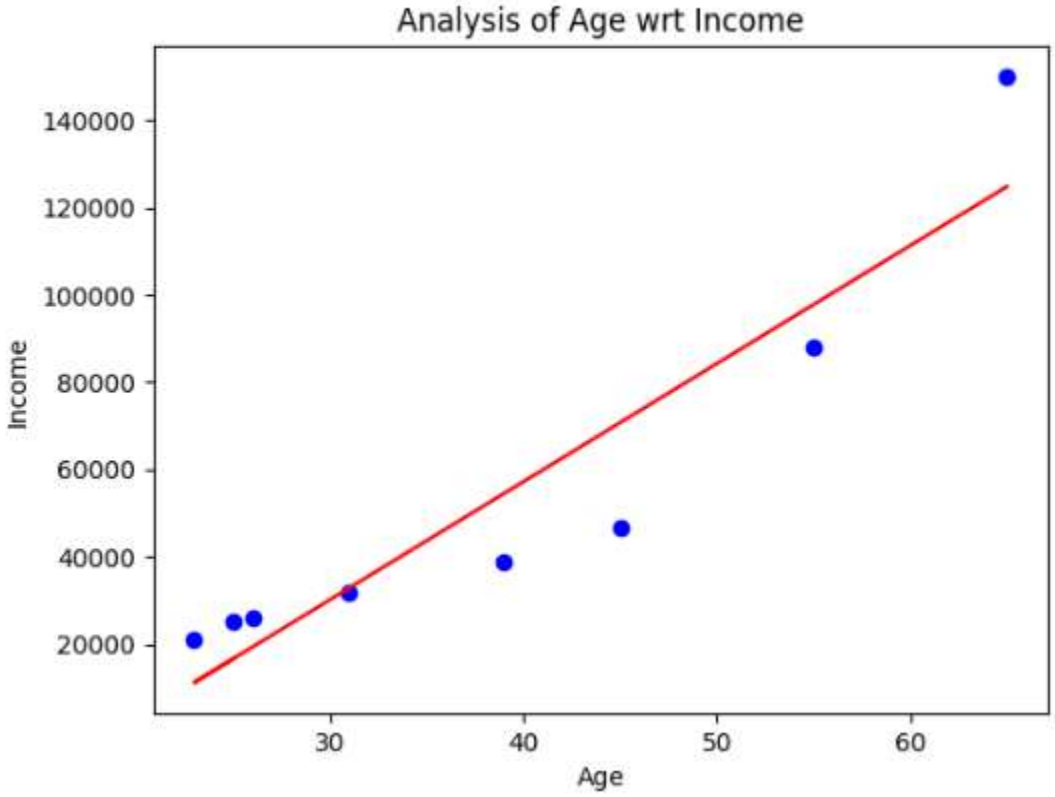
```
mean_income = np.mean(income)

# calculating cross-deviation and deviation about age
CD_ageincome = np.sum(income*age) - num*mean_income*mean_age
CD_ageage = np.sum(age*age) - num*mean_age*mean_age

# calculating regression coefficients
b1 = CD_ageincome / CD_ageage
b0 = mean_income - b1*mean_age

# to display coefficients
print("Estimated Coefficients :")
print("b0 = ",b0,"\nb1 = ",b1)

# To plot the actual points as scatter plot
plt.scatter(age, income, color = "b",marker = "o")

# To predict response vector
response_Vec = b0 + b1*age

# To plot the regression line
plt.plot(age, response_Vec, color = "r")

# Title of the graph
plt.title('Analysis of Age wrt Income')
# Placing labels
plt.xlabel('Age')
plt.ylabel('Income')
```

 # To display plot

plt.show()

Output :

```
Estimated Coefficients :
b0 =  -51019.52326901248
b1 =  2704.3889519485433
```



Experiment 5 : Implementation of Multiple Linear Regression for House Price Prediction using sklearn

Aim : To develop a python program to implement multiple Linear Regression for House Price Prediction using sklearn

Software used : Jupyterlab

Description

Multiple Linear regression:

Multiple linear regression is a statistical technique that models the relationship between a dependent variable (house price) and multiple independent variables (features like size, location, number of bedrooms, etc.).

Program:

```
# Importing modules and packages
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error


# Importing the dataset
df = pd.read_csv('Real-estate1.csv')
print(df.head())  # Display the first few rows of the dataset
print(df.columns)  # Display column names


# Plotting a scatterplot
sns.scatterplot(
    x='X4 number of convenience stores',
    y='Y house price of unit area',
    data=df
```

```
)
plt.title('Convenience Stores vs House Price')
plt.xlabel('Number of Convenience Stores')
plt.ylabel('House Price per Unit Area')
plt.show()

# Creating feature and target variables
X = df.drop('Y house price of unit area', axis=1)  # Feature variables
y = df['Y house price of unit area']  # Target variable
print(X.head())  # Display the first few rows of features
print(y.head())  # Display the first few rows of the target variable

# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=101)

# Creating and fitting the regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Making predictions
predictions = model.predict(X_test)

# Model evaluation
print('Mean Squared Error:', mean_squared_error(y_test, predictions))
print('Mean Absolute Error:', mean_absolute_error(y_test, predictions))
```

Output :

X1 transaction date X2 house age … X6 longitude Y house price of unit area

0 2012.917 32.0 … 121.54024 37.9

1 2012.917 19.5 … 121.53951 42.2

2 2013.583 13.3 … 121.54391 47.3

3 2013.500 13.3 … 121.54391 54.8

4 2012.833 5.0 … 121.54245 43.1

[5 rows x 7 columns]

Index([‘X1 transaction date’, ‘X2 house age’,

‘X3 distance to the nearest MRT station’,

‘X4 number of convenience stores’, ‘X5 latitude’, ‘X6 longitude’,

‘Y house price of unit area’],

dtype=’object’)

X1 transaction date X2 house age … X5 latitude X6 longitude

0 2012.917 32.0 … 24.98298 121.54024

1 2012.917 19.5 … 24.98034 121.53951

2 2013.583 13.3 … 24.98746 121.54391

3 2013.500 13.3 … 24.98746 121.54391

4 2012.833 5.0 … 24.97937 121.54245

.. … … … … …

409 2013.000 13.7 … 24.94155 121.50381

410 2012.667 5.6 … 24.97433 121.54310

411 2013.250 18.8 … 24.97923 121.53986

412 2013.000 8.1 … 24.96674 121.54067

413 2013.500 6.5 … 24.97433 121.54310

[414 rows x 6 columns]

0 37.9

1 42.2

2 47.3

3 54.8

4 43.1

…

409 15.4

410 50.0

411 40.6

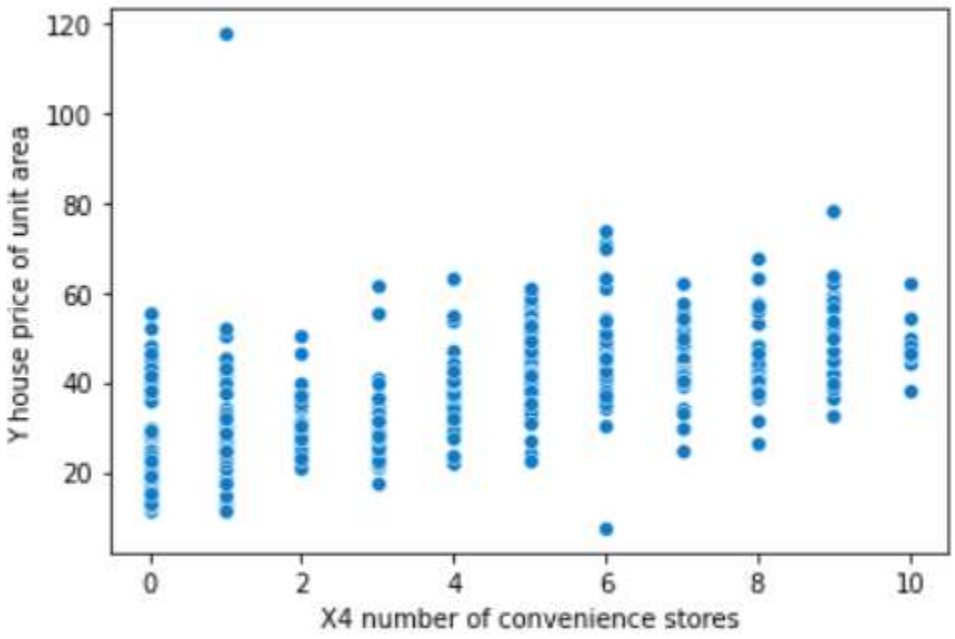
412 52.5

413 63.9

Name: Y house price of unit area, Length: 414, dtype: float64

mean_squared_error : 46.21179783493418

mean_absolute_error : 5.392293684756571



Experiment 6 : Implementation of Decision tree using sklearn and its parameter tuning

Aim : To develop a python program to implement Decision tree using sklearn and its parameter tuning

Software used : Jupyterlab

Description

Decision Trees

Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. A tree can be seen as a piecewise constant approximation. For instance, in the example below, decision trees learn from data to approximate a sine curve with a set of if-then-else decision rules. The deeper the tree, the more complex the decision rules and the fitter the model.

Program:

```
# Importing the required packages
import numpy as np
import pandas as pd
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
import matplotlib.pyplot as plt
from sklearn.tree import plot_tree


# Function to import the dataset
def importdata():
    balance_data = pd.read_csv(
        'https://archive.ics.uci.edu/ml/machine-learning-databases/balance-scale/balance-scale.data',
        sep=',', header=None
    )
    # Displaying dataset information
    print("Dataset Length: ", len(balance_data))
```

```
    print("Dataset Shape: ", balance_data.shape)
    print("Dataset: ", balance_data.head())
    return balance_data

# Function to split the dataset into features and target variables
def splitdataset(balance_data):
    # Separating the target variable
    X = balance_data.values[:, 1:5]
    Y = balance_data.values[:, 0]

    # Splitting the dataset into train and test
    X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3, random_state=100)
    return X, Y, X_train, X_test, y_train, y_test

# Function to train the model using Gini index
def train_using_gini(X_train, X_test, y_train):
    # Creating the classifier object
    clf_gini = DecisionTreeClassifier(criterion="gini", max_depth=3, min_samples_leaf=5)
    # Performing training
    clf_gini.fit(X_train, y_train)
    return clf_gini

# Function to train the model using Entropy
def train_using_entropy(X_train, X_test, y_train):
    # Decision tree with entropy
    clf_entropy = DecisionTreeClassifier(
        criterion="entropy", random_state=100, max_depth=3, min_samples_leaf=5
    )
```

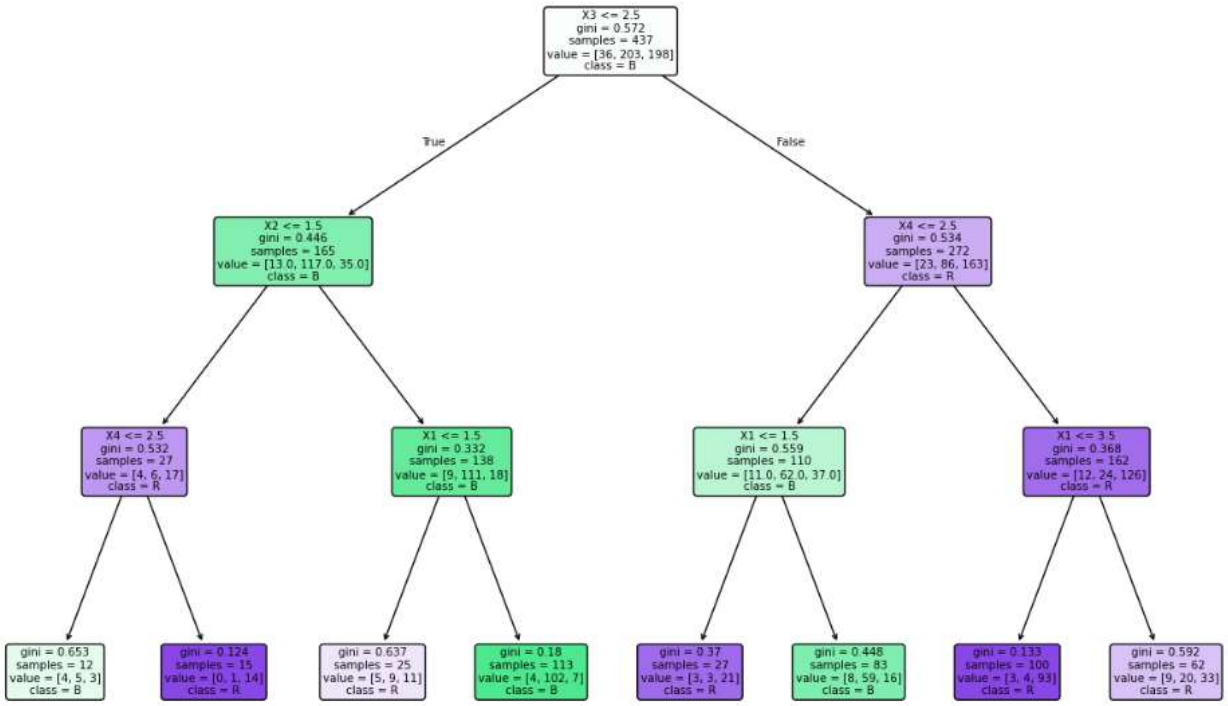
```
    # Performing training
    clf_entropy.fit(X_train, y_train)
    return clf_entropy


# Function to make predictions
def prediction(X_test, clf_object):
    y_pred = clf_object.predict(X_test)
    print("Predicted values:")
    print(y_pred)
    return y_pred
# Function to calculate accuracy
def cal_accuracy(y_test, y_pred):
    print("Confusion Matrix: ", confusion_matrix(y_test, y_pred))
    print("Accuracy : ", accuracy_score(y_test, y_pred) * 100)
    print("Report : ", classification_report(y_test, y_pred))
# Function to plot the decision tree
def plot_decision_tree(clf_object, feature_names, class_names):
    plt.figure(figsize=(15, 10))
    plot_tree(clf_object, filled=True, feature_names=feature_names, class_names=class_names,
rounded=True)
    plt.show()
# Main execution
if __name__ == "__main__":
    data = importdata()
    X, Y, X_train, X_test, y_train, y_test = splitdataset(data)
# Training using Gini Index
    clf_gini = train_using_gini(X_train, X_test, y_train)
    # Training using Entropy
```

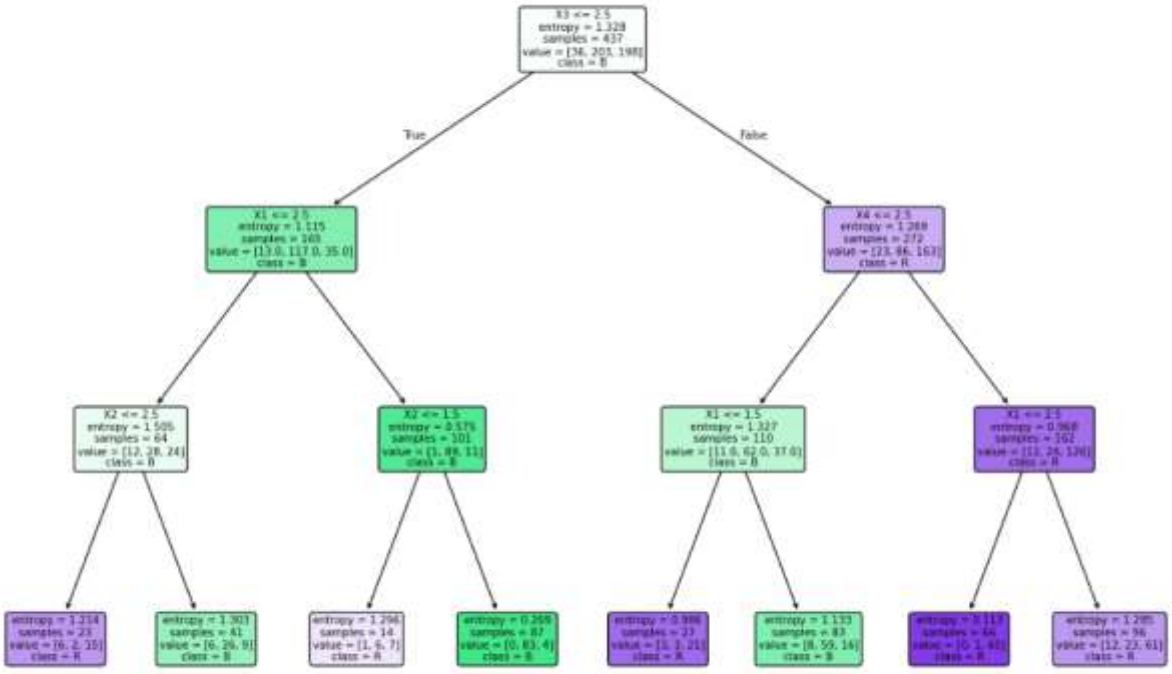
```
clf_entropy = train_using_entropy(X_train, X_test, y_train)
# Visualizing the Decision Trees
plot_decision_tree(clf_gini, ['X1', 'X2', 'X3', 'X4'], ['L', 'B', 'R'])
plot_decision_tree(clf_entropy, ['X1', 'X2', 'X3', 'X4'], ['L', 'B', 'R'])
# Operational Phase
print("Results Using Gini Index:")
y_pred_gini = prediction(X_test, clf_gini)
cal_accuracy(y_test, y_pred_gini)
```

Output :

```
Dataset Length:  625
Dataset Shape:  (625, 5)
Dataset:     0  1  2  3  4
0  B  1  1  1  1
1  R  1  1  1  2
2  R  1  1  1  3
3  R  1  1  1  4
4  R  1  1  1  5
```

```
Results Using Gini Index:
Predicted values:
['R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'R' 'L' 'L' 'L' 'R' 'L' 'R' 'L'
 'L' 'R' 'L' 'R' 'L' 'L' 'R' 'L' 'L' 'L' 'R' 'L' 'L' 'L' 'R' 'L' 'L' 'L'
 'L' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'R' 'L' 'R'
 'R' 'L' 'R' 'R' 'L' 'L' 'R' 'R' 'L' 'L' 'L' 'L' 'L' 'R' 'R' 'L' 'L' 'R'
 'R' 'L' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'R' 'R' 'L' 'R' 'L'
 'R' 'R' 'L' 'L' 'L' 'R' 'R' 'L' 'L' 'L' 'R' 'L' 'R' 'R' 'R' 'R' 'R' 'R'
 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'L'
 'L' 'L' 'L' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R'
 'L' 'L' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'R' 'R'
 'L' 'L' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'R' 'R'
 'L' 'R' 'R' 'L' 'L' 'R' 'R' 'R']
Confusion Matrix:  [[ 0  6  7]
 [ 0 67 18]
 [ 0 19 71]]
Accuracy :  73.40425531914893
Report :                precision    recall  f1-score   support

           B       0.00      0.00      0.00        13
           L       0.73      0.79      0.76        85
           R       0.74      0.79      0.76        90

    accuracy                           0.73       188
   macro avg       0.49      0.53      0.51       188
weighted avg       0.68      0.73      0.71       188
```


X3 <= 2.5
gini = 0.572
samples = 437
value = [36, 203, 198]
class = B
True
False
X2 <= 1.5
gini = 0.446
samples = 165
value = [13.0, 117.0, 35.0]
class = B
X4 <= 2.5
gini = 0.534
samples = 272
value = [23, 86, 163]
class = R
X4 <= 2.5
gini = 0.532
samples = 27
value = [4, 6, 17]
class = R
X1 <= 1.5
gini = 0.332
samples = 138
value = [9, 111, 18]
class = B
X1 <= 1.5
gini = 0.559
samples = 110
value = [11.0, 62.0, 37.0]
class = B
X1 <= 3.5
gini = 0.368
samples = 162
value = [12, 24, 126]
class = R
gini = 0.653
samples = 12
value = [4, 5, 3]
class = B
gini = 0.124
samples = 15
value = [0, 1, 14]
class = R
gini = 0.637
samples = 25
value = [5, 9, 11]
class = R
gini = 0.18
samples = 113
value = [4, 102, 7]
class = B
gini = 0.37
samples = 27
value = [3, 3, 21]
class = R
gini = 0.448
samples = 83
value = [8, 59, 16]
class = B
gini = 0.133
samples = 100
value = [3, 4, 93]
class = R
gini = 0.592
samples = 62
value = [9, 20, 33]
class = R


True
False

Experiment 7 : Implementation of K Nearest Neighbor using sklearn

Aim : To develop a python program to implement K Nearest Neighbor using sklearn

Software used : Jupyterlab

Program:

```
# Import necessary modules
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
import random
# Loading data
data_iris = load_iris()
# To get list of target names
label_target = data_iris.target_names
print()
print("Sample Data from Iris Dataset")
print("*"*30)
# to display the sample data from the iris dataset
for i in range(10):
        rn = random.randint(0,120)
        print(data_iris.data[rn],"===>",label_target[data_iris.target[rn]])

# Create feature and target arrays
X = data_iris.data
y = data_iris.target

# Split into training and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state=1)
print("The Training dataset length: ",len(X_train))
```

```
print("The Testing dataset length: ",len(X_test))
try:
        nn = int(input("Enter number of neighbors :"))
        knn = KNeighborsClassifier(nn)
        knn.fit(X_train, y_train)
        # to display the score
        print("The Score is :",knn.score(X_test, y_test))
        # To get test data from the user
        test_data = input("Enter Test Data :").split(",")
        for i in range(len(test_data)):
                test_data[i] = float(test_data[i])
        print()
        v = knn.predict([test_data])
        print("Predicted output is :",label_target[v])
except:
        print("Please supply valid input.....")
```

Output :

```
Sample Data from Iris Dataset
*****************************
[5.9 3.  4.2 1.5] ===> versicolor
[4.4 3.2 1.3 0.2] ===> setosa
[5.1 3.8 1.5 0.3] ===> setosa
[5.4 3.4 1.5 0.4] ===> setosa
[4.8 3.  1.4 0.3] ===> setosa
[5.7 2.5 5.  2. ] ===> virginica
[5.  3.  1.6 0.2] ===> setosa
[4.6 3.4 1.4 0.3] ===> setosa
```

[4.8 3. 1.4 0.3] ===> setosa

[4.9 3.6 1.4 0.1] ===> setosa

The Training dataset length: 105

The Testing dataset length: 45

Enter number of neighbors : 2

The Score is : 0.9777777777777777

Enter Test Data : 6.2,2.6,3.4,0.6

Predicted output is : ['versicolor']

Experiment 8 : Implementation of Logistic Regression using sklearn

Aim : To develop a python program to implement Logistic Regression using sklearn

Software used : Jupyterlab

Program:

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn import metrics
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Load dataset
diab_df = pd.read_csv("diabetes.csv")
diab_df.head()

# Split dataset into features and target variable
diab_cols = ['Pregnancies', 'Insulin', 'BMI', 'Age', 'Glucose', 'BloodPressure', 'DiabetesPedigreeFunction']

X = diab_df[diab_cols]  # Features
y = diab_df.Outcome  # Target variable

# Splitting Data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=0)

# Model Development and Prediction
logreg = LogisticRegression(solver='liblinear')  # Instantiate the model
logreg.fit(X_train, y_train)  # Fit the model with data
```

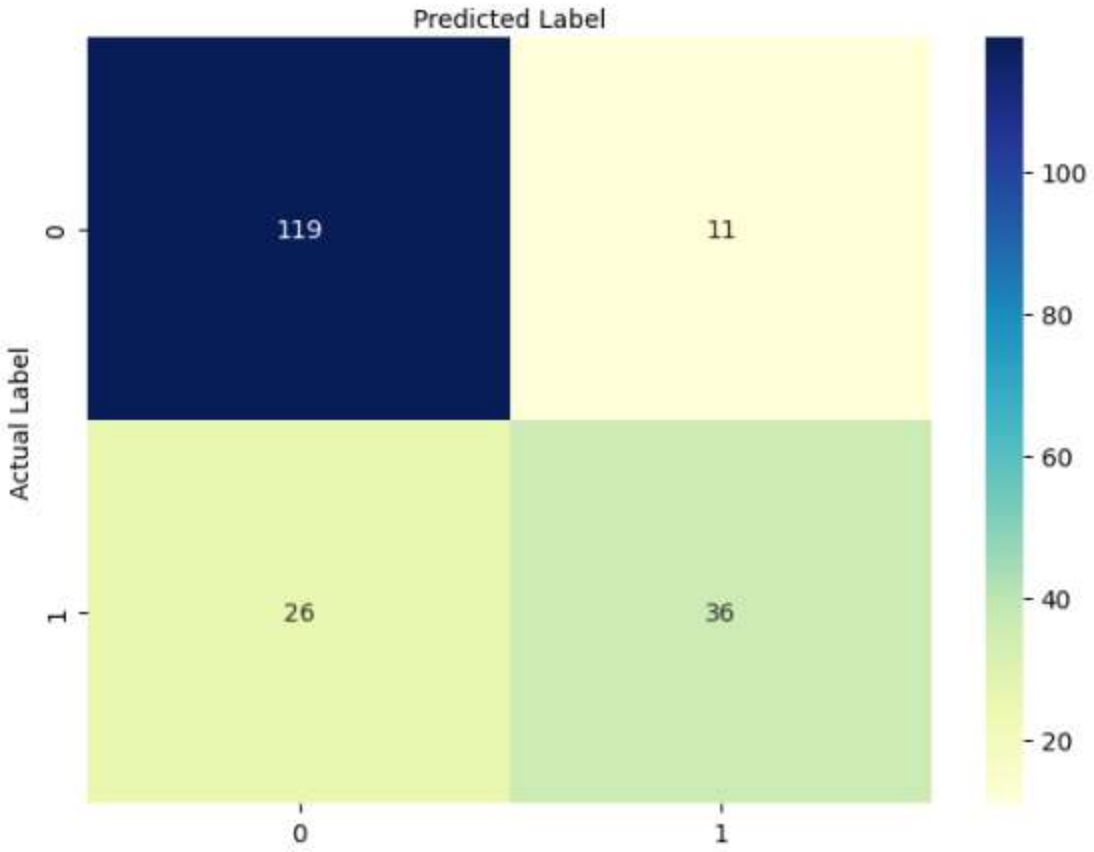
```
y_pred = logreg.predict(X_test)  # Predicting y_pred

# Model Evaluation using Confusion Matrix
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
cnf_matrix

# Visualizing Confusion Matrix using Heatmap
class_names = [0, 1]  # Name of classes
fig, ax = plt.subplots()

tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)
# Create heatmap
sns.heatmap(pd.DataFrame(cnf_matrix), annot=True, cmap="YlGnBu", fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion Matrix', y=1.1)
plt.ylabel('Actual Label')
plt.xlabel('Predicted Label')
# Confusion Matrix Evaluation Metrics
print("Accuracy:", metrics.accuracy_score(y_test, y_pred))
print("Precision:", metrics.precision_score(y_test, y_pred))
print("Recall:", metrics.recall_score(y_test, y_pred))
```

Output :

Accuracy: 0.8072916666666666

Precision: 0.7659574468085106

Recall: 0.5806451612903226


Predicted Label
Actual Label
0
1
119
11
26
36
0
1
100
80
60
40
20

Experiment 9 : Implementation of K – Means clustering

Aim : To develop a python program to implement K – Means clustering

Software used : Jupyterlab

Program:

```
import numpy as np
from sklearn.datasets import make_blobs
class KMeans:
    def __init__(self, n_clusters, max_iters=100):
        self.n_clusters = n_clusters
        self.max_iters = max_iters
    def fit(self, X):
        # Initialize centroids randomly
        self.centroids = X[np.random.choice(X.shape[0], self.n_clusters, replace=False)]
        for _ in range(self.max_iters):
            # Assign each data point to the nearest centroid
            labels = self._assign_labels(X)
            # Update centroids
            new_centroids = self._update_centroids(X, labels)
            # Check for convergence
            if np.all(self.centroids == new_centroids):
                break
            self.centroids = new_centroids
    def _assign_labels(self, X):
        # Compute distances from each data point to centroids
        distances = np.linalg.norm(X[:, np.newaxis] - self.centroids, axis=2)
        # Assign labels based on the nearest centroid
        return np.argmin(distances, axis=1)
    def _update_centroids(self, X, labels):
```

```
    new_centroids = np.array([X[labels == i].mean(axis=0) for i in range(self.n_clusters)])
    return new_centroids
# Generate synthetic data using sklearn.datasets.make_blobs
X, _ = make_blobs(n_samples=300, centers=3, random_state=42)
# Create a K-Means instance with 3 clusters
kmeans = KMeans(n_clusters=3)
# Fit the model to the data
kmeans.fit(X)
# Get cluster assignments for each data point
labels = kmeans._assign_labels(X)
print("Cluster Assignments:", labels)
print("Final Centroids:", kmeans.centroids)
```

Output :

```
Cluster Assignments: [1 1 0 2 1 2 0 2 0 0 0 2 0 0 1 0 1 2 0 0 0 0 2 1 0 1 1 2 2 0 0 0 1 0 1 0 1
 2 1 2 2 0 1 2 0 0 1 2 1 2 2 1 1 0 1 2 1 0 2 0 1 2 2 1 1 2 2 1 1 0 2 1 1 0
 0 1 1 2 0 2 0 0 1 0 2 1 1 0 2 0 1 0 1 0 0 1 1 0 1 1 2 0 2 0 0 0 0 0 2 1 2
 0 0 0 0 2 1 2 1 2 2 2 0 1 1 1 1 0 1 1 0 0 0 0 0 2 2 1 0 1 0 0 1 0 2 2 2 0
 2 0 0 1 2 1 0 2 2 1 1 0 0 1 1 1 0 1 2 0 0 0 0 0 2 0 2 2 2 0 2 2 1 0 1 2 2
 1 2 0 2 2 1 1 2 1 2 2 2 2 0 1 0 0 2 2 0 2 1 1 2 0 0 1 2 2 1 1 1 1 0 1 1 2
 1 1 0 2 1 1 2 0 0 1 0 1 2 2 1 2 1 1 1 2 2 0 1 2 2 2 1 2 1 2 1 2 2 1 2 0 1
 0 0 0 1 0 2 2 1 2 2 0 0 2 2 2 1 1 1 0 0 0 2 2 2 2 1 2 1 2 2 1 0 2 2 0 1 0
 2 0 1 1]


Final Centroids:
[
 [-2.63323268  9.04356978]
 [-6.88387179 -6.98398415]
 [ 4.74710337  2.01059427] ]
```