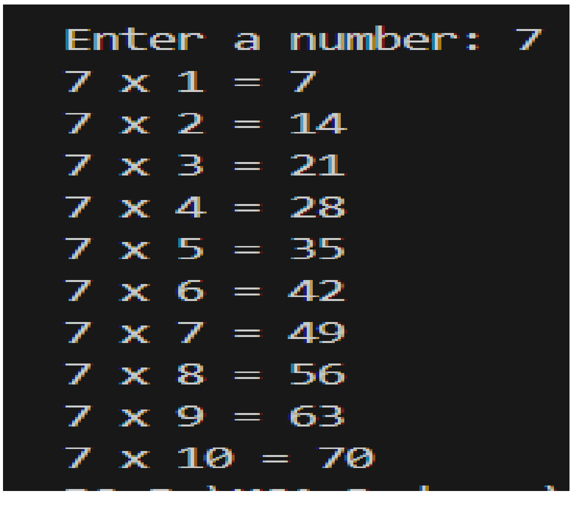
INDEX

PROGRAM 1

a) Write a python program to print the multiplication table for the given number.

```
def multiplication_table(num):
    for i in range(1, 11):
        print(f"{num} x {i} = {num * i}")


number = int(input("Enter a number: "))


multiplication_table(number)
```

OUTPUT: -



```
Enter a number: 7
7 x 1 = 7
7 x 2 = 14
7 x 3 = 21
7 x 4 = 28
7 x 5 = 35
7 x 6 = 42
7 x 7 = 49
7 x 8 = 56
7 x 9 = 63
7 x 10 = 70
```

b) Write a python program to check whether the given number is prime or not.

```python
def is_prime(number):
    if number <= 1:
        return False
    for i in range(2, int(number ** 0.5) + 1):
        if number % i == 0:
            return False
    return True

num = int(input("Enter a number: "))
if is_prime(num):
    print(f"{num} is a prime number.")
else:
    print(f"{num} is not a prime number.")
```

<u>OUTPUT: -</u>

```
PS B:\MCA 3rd sem\AI & ML\AIML
Enter a number: 123
123 is not a prime number.
```

```
PS B:\MCA 3rd sem\AI & ML\
Enter a number: 31
31 is a prime number.
```

c) Write a python program to find factorial of the given number.

```
def factorial(n):
    if n == 0 or n == 1:
        return 1
    else:
        return n * factorial(n - 1)

# Input from user
num = int(input("Enter a number: "))
result = factorial(num)

print(f"The factorial of {num} is {result}")
```

<u>OUTPUT: -</u>

```
PS B:\MCA 3rd sem\AI & ML\AIML> pytho
Enter a number: 12
The factorial of 12 is : 479001600
```

PROGRAM 2

Write a python program to implement simple Chatbot.

```
def chatbot():
    print("Hello! I'm ChatBot. How can I help you today?")
    while True:
        user_input = input("You: ").lower()
        if "hi" in user_input or "hello" in user_input:
            print("ChatBot: Hello! How can I assist you?")
        elif "how are you" in user_input:
            print("ChatBot: I'm just a program, but I'm functioning perfectly! How about you?")
        elif "your name" in user_input:
            print("ChatBot: My name is ChatBot!")
        elif "who is the prime minister of india" in user_input:
            print("ChatBot: Mr. Narendra Modi!")
        elif "bye" in user_input or "exit" in user_input:
            print("ChatBot: Goodbye! Have a great day!")
            break
        else:
            print("ChatBot: I'm sorry, I don't understand that. Can you ask something else?")
chatbot()
```

OUTPUT: -

```
PS B:\MCA 3rd sem\AI & ML\AIML> python -u "b:\MC
Hello! I'm ChatBot. How can I help you today?
You: hi
ChatBot: Hello! How can I assist you?
You: who is the prime minister of india
ChatBot: Mr. Narendra Modi!
You: bye
ChatBot: Goodbye! Have a great day!
```

PROGRAM 3

Write a python program to generate Calendar for the given month and year.

```
import calendar


def generate_calendar(year, month):
    # Print the calendar for the given month and year
    print(calendar.month(year, month))


# Input from user
year = int(input("Enter year: "))
month = int(input("Enter month (1-12): "))


generate_calendar(year, month)
```

OUTPUT: -

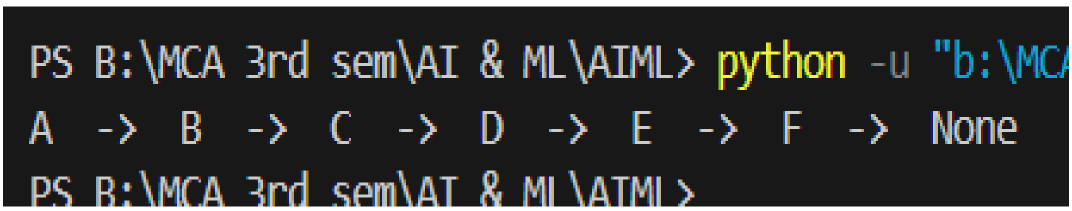
```
PS B:\MCA 3rd sem\AI & ML\AIM
Enter year: 2024
Enter month (1-12): 11
   November 2024
Mo Tu We Th Fr Sa Su
             1  2  3
 4  5  6  7  8  9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30
```

PROGRAM 4

Write a python program to implement Breadth First Search Traversal.

```
from collections import deque
def bfs(graph, start_node):
    visited = set()
    queue = deque([start_node])
    while queue:
        node = queue.popleft()
        if node not in visited:
            print(node," -> ", end=" ")
            visited.add(node)

            # Add all unvisited neighbors to the queue
            for neighbor in graph[node]:
                if neighbor not in visited:
                    queue.append(neighbor)
graph = {
    'A': ['B', 'C'],    'B': ['D', 'E'],    'C': ['F'],    'D': [],    'E': ['F'],    'F': []
}
bfs(graph, 'A')
print("None")
```

OUTPUT: -



PROGRAM 5

Write a python program to implement Depth First Search Traversal.

```
def dfs(graph, node, visited):
    if node not in visited:
        print(node, “->” , end=" ")
        visited.add(node)

        for neighbor in graph[node]:
            dfs(graph, neighbor, visited)
graph = {
    'U': ['V', 'W'],
    'V': ['X', 'Y'],
    'W': ['Z'],
    'X': [],
    'Y': ['Z'],
    'Z': []
}
visited = set()
dfs(graph, 'U', visited)
print("None")
```

OUTPUT: -

```
PS B:\MCA 3rd sem\AI & ML\AIML> python -
U -> V -> X -> Y -> Z -> W -> None
PS B:\MCA 3rd sem\AI & ML\AIML>
```

PROGRAM 6

Write a python program to implement Water Jug Problem.

```
from collections import deque


def is_visited(state, visited_states):
    return state in visited_states


def print_solution(path):
    for state in path:
        print(f"Jug1: {state[0]} liters, Jug2: {state[1]} liters")
    print("\nGoal reached!\n")


def water_jug_bfs(jug1_capacity, jug2_capacity, target):
    initial_state = (0, 0)

    # Queue to store paths to explore
    queue = deque([(initial_state, [])])

    visited_states = set()

    while queue:
        (jug1, jug2), path = queue.popleft()

        # If we have already visited this state, skip it
        if is_visited((jug1, jug2), visited_states):
            continue
```

```
        visited_states.add((jug1, jug2))

        # Add the current state to the path
        path = path + [(jug1, jug2)]

        if jug1 == target or jug2 == target:
            print_solution(path)
            return True

        # All possible operations (transitions)
        possible_states = [
            (jug1_capacity, jug2),   # Fill Jug1
            (jug1, jug2_capacity),   # Fill Jug2
            (0, jug2),               # Empty Jug1
            (jug1, 0),               # Empty Jug2
            (min(jug1 + jug2, jug1_capacity), max(0, jug2 - (jug1_capacity - jug1))), # Pour Jug2
into Jug1
            (max(0, jug1 - (jug2_capacity - jug2)), min(jug1 + jug2, jug2_capacity)) # Pour Jug1
into Jug2
        ]

        # Enqueue all possible states if they haven't been visited
        for state in possible_states:
            if not is_visited(state, visited_states):
                queue.append((state, path))

    print("No solution found.")
    return False

jug1_capacity = int(input("Enter the capacity of Jug1: "))
```

jug2_capacity = int(input("Enter the capacity of Jug2: "))

target = int(input("Enter the target amount of water: "))

water_jug_bfs(jug1_capacity, jug2_capacity, target)

OUTPUT: -

```
PS B:\MCA 3rd sem\AI & ML\AIML> python -
Enter the capacity of Jug1: 3
Enter the capacity of Jug2: 4
Enter the target amount of water: 2
Jug1: 0 liters, Jug2: 0 liters
Jug1: 3 liters, Jug2: 0 liters
Jug1: 0 liters, Jug2: 3 liters
Jug1: 3 liters, Jug2: 3 liters
Jug1: 2 liters, Jug2: 4 liters

Goal reached!
```

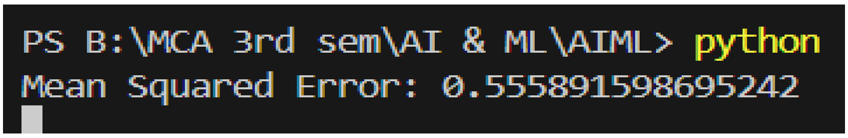
PROGRAM 7

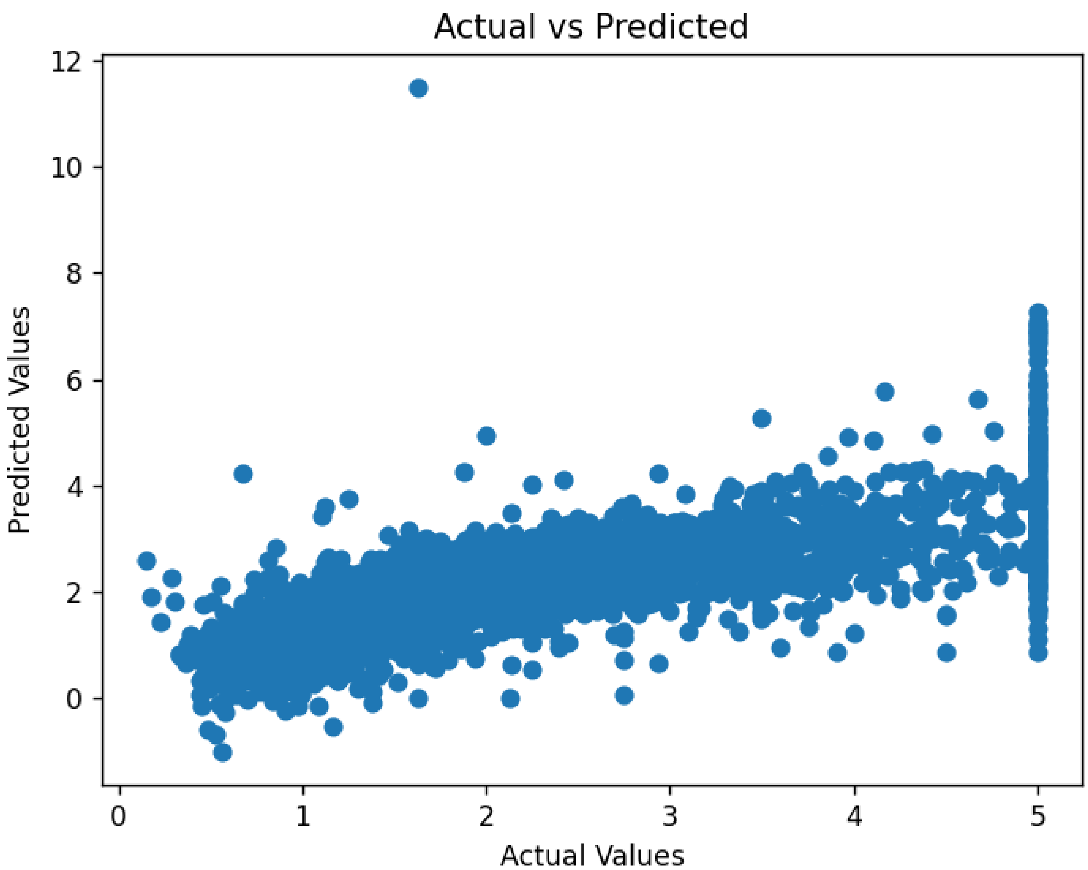
Implement Linear regression using any real data set.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.datasets import fetch_california_housing
from sklearn.metrics import mean_squared_error

# Load the California Housing dataset
california = fetch_california_housing()
X = california.data
y = california.target

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

# Create a Linear Regression model
model = LinearRegression()

# Train the model
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Print the Mean Squared Error
```

```
print("Mean Squared Error:", mean_squared_error(y_test, y_pred))

# Plotting actual vs predicted values
plt.scatter(y_test, y_pred)
plt.xlabel("Actual Values")
plt.ylabel("Predicted Values")
plt.title("Actual vs Predicted")
plt.show()
```

OUTPUT: -





PROGRAM 8

Implement Logistic regression using any real data set.

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn import datasets
from sklearn.metrics import (confusion_matrix, precision_score, recall_score, f1_score,
                             accuracy_score)

import matplotlib.pyplot as plt
iris = datasets.load_iris()
X = iris.data
y = iris.target

# We will classify only two classes for binary classification
X = X[y != 2]
y = y[y != 2]

# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Create a Logistic Regression model
log_reg = LogisticRegression()

# Train the model
log_reg.fit(X_train, y_train)
y_pred = log_reg.predict(X_test)
y_prob = log_reg.predict_proba(X_test)[:, 1]  # Probability estimates for ROC
```

```
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", conf_matrix)
precision = precision_score(y_test, y_pred)
print("Precision:", precision)
recall = recall_score(y_test, y_pred)
print("Recall:", recall)
f1 = f1_score(y_test, y_pred)
print("F1-Score:", f1)
```

OUTPUT: -

```
PS B:\MCA 3rd sem\AI & ML\AIML> python -u
Accuracy: 1.0
Confusion Matrix:
 [[12  0]
 [ 0  8]]
Precision: 1.0
Recall: 1.0
F1-Score: 1.0
PS B:\MCA 3rd sem\AI & ML\AIML>
```

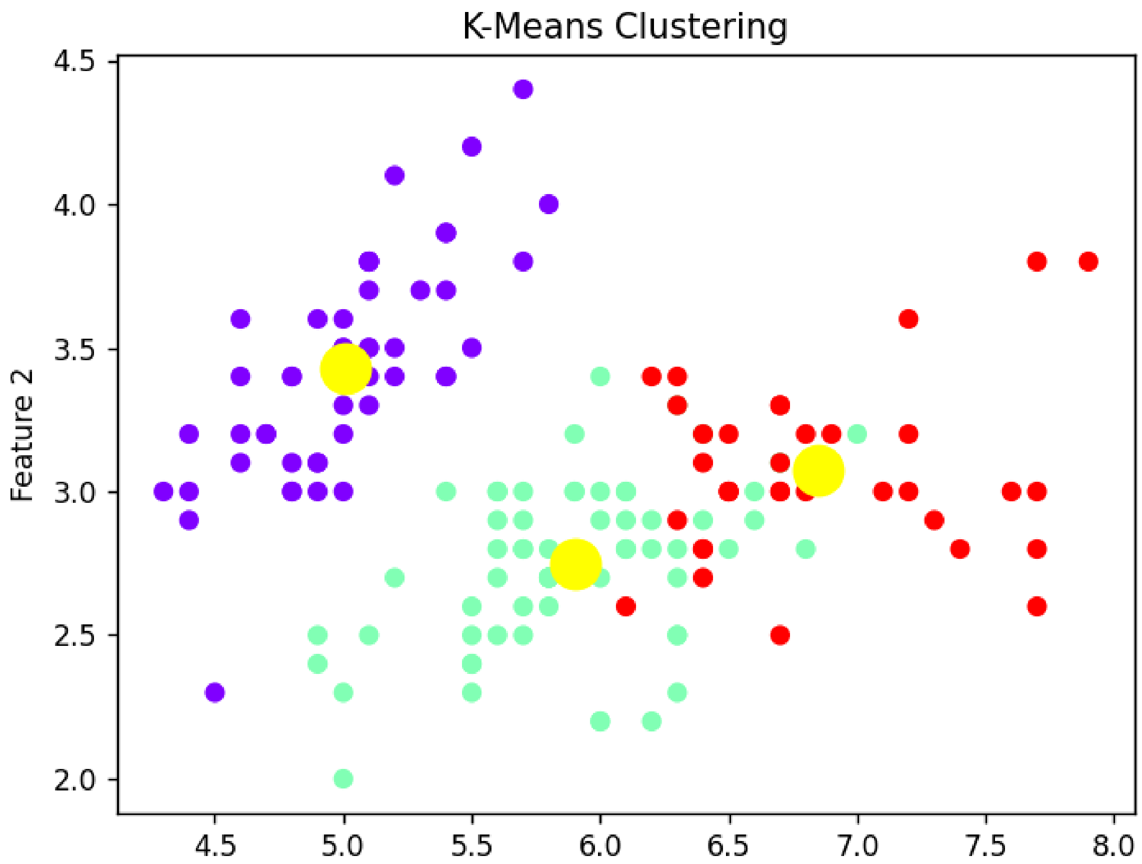
PROGRAM 9

Use a real-life data set to implement K-means clustering.

```
import numpy as np

from sklearn.datasets import load_iris

from sklearn.cluster import KMeans

import matplotlib.pyplot as plt


# Load the Iris dataset

iris = load_iris()

X = iris.data


# Create the KMeans model

kmeans = KMeans(n_clusters=3)


# Fit the model

kmeans.fit(X)


# Get the cluster centroids

centroids = kmeans.cluster_centers_

labels = kmeans.labels_


# Plot the clusters

plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='rainbow')

plt.scatter(centroids[:, 0], centroids[:, 1], s=300, c='yellow')

plt.xlabel("Feature 1")

plt.ylabel("Feature 2")

plt.title("K-Means Clustering")
```

plt.show()

OUTPUT: -



PROGRAM 10

Implementing Numpy in lab and use it in: Joining Numpy arrays , Intersection & Difference , Mean , median , Standard Deviation.

```
import numpy as np

# Joining Numpy arrays

array1 = np.array([11, 22, 33, 44])

array2 = np.array([45, 56, 67,78])

joined_array = np.concatenate((array1, array2))

print("Joined Array:", joined_array)

# Intersection and Difference

array3 = np.array([11, 22, 32, 16, 52])

array4 = np.array([41, 22, 16, 71])

intersection = np.intersect1d(array3, array4)

difference = np.setdiff1d(array3, array4)

print("Intersection:", intersection)

print("Difference:", difference)

# Mean, Median, Standard Deviation

data = np.array([10, 20, 60, 40, 50])

print("Mean:", np.mean(data))

print("Median:", np.median(data))

print("Standard Deviation:", np.std(data))
```

OUTPUT: -

```
PS B:\MCA 3rd sem\AI & ML\AIML> python -u "b:\
Joined Array: [11 22 33 44 45 56 67 78]
Intersection: [16 22]
Difference: [11 32 52]
Mean: 36.0
Median: 40.0
Standard Deviation: 18.547236990991408
PS B:\MCA 3rd sem\AI & ML\AIML>
```

PROGRAM 11

Implementing Pandas Library in lab with available data base and simple commands: Head, Tail. Describe, tail, iloc, loc, drop, mean, median, maximum, minimum.

```
import pandas as pd

# Create a sample DataFrame
data = {'A': [1, 2, 3, 4, 5],
        'B': [10, 20, 30, 40, 50],
        'C': [100, 200, 300, 400, 500]}
df = pd.DataFrame(data)

# Basic Pandas Commands
print(df.head()) # First 5 rows
print(df.tail()) # Last 5 rows
print(df.describe())  # Summary of data

# Using iloc and loc
print(df.iloc[0]) # First row using iloc
print(df.loc[0])  # First row using loc

# Dropping a column
df_dropped = df.drop('B', axis=1)
print(df_dropped)

# Mean, Median, Max, Min
print("Mean:", df.mean())
```

print("Median:", df.median())

print("Max:", df.max())

print("Min:", df.min())

OUTPUT: -

```
PS B:\MCA 3rd sem\AI & ML\AIML> python -u "
   A   B    C
0  1  10  100
1  2  20  200
2  3  30  300
3  4  40  400
4  5  50  500
   A   B    C
0  1  10  100
1  2  20  200
2  3  30  300
3  4  40  400
4  5  50  500
              A          B           C
count  5.000000   5.000000    5.000000
mean   3.000000  30.000000  300.000000
std    1.581139  15.811388  158.113883
min    1.000000  10.000000  100.000000
25%    2.000000  20.000000  200.000000
50%    3.000000  30.000000  300.000000
75%    4.000000  40.000000  400.000000
max    5.000000  50.000000  500.000000
A      1
B     10
C    100
Name: 0, dtype: int64
A      1
B     10
C    100
Name: 0, dtype: int64
```

```
   A    C
0  1  100
1  2  200
2  3  300
3  4  400
4  5  500
Mean: A      3.0
B     30.0
C    300.0
dtype: float64
Median: A      3.0
B     30.0
C    300.0
dtype: float64
Max: A      5
dtype: float64
Median: A      3.0
B     30.0
C    300.0
dtype: float64
Median: A      3.0
B     30.0
dtype: float64
Median: A      3.0
dtype: float64
Median: A      3.0
B     30.0
dtype: float64
Median: A      3.0
B     30.0
dtype: float64
```

```
Median: A      3.0
dtype: float64
Median: A      3.0
dtype: float64
dtype: float64
Median: A      3.0
dtype: float64
Median: A      3.0
B     30.0
C    300.0
dtype: float64
Median: A      3.0
dtype: float64
dtype: float64
dtype: float64
Median: A      3.0
B     30.0
C    300.0
dtype: float64
Max: A      5
B     50
C    500
dtype: int64
Min: A      1
B     10
C    100
dtype: int64
```

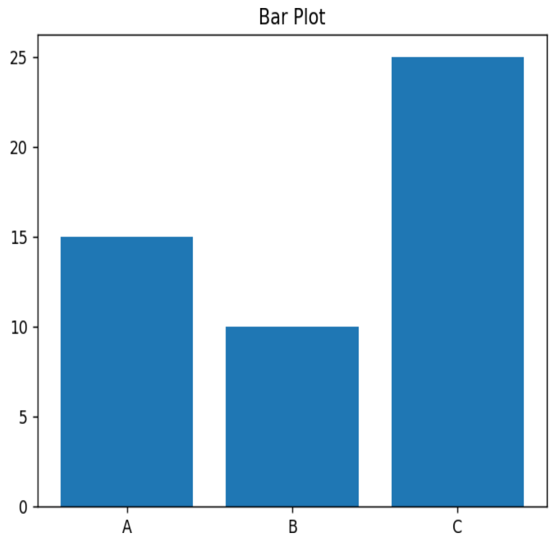
PROGRAM 12

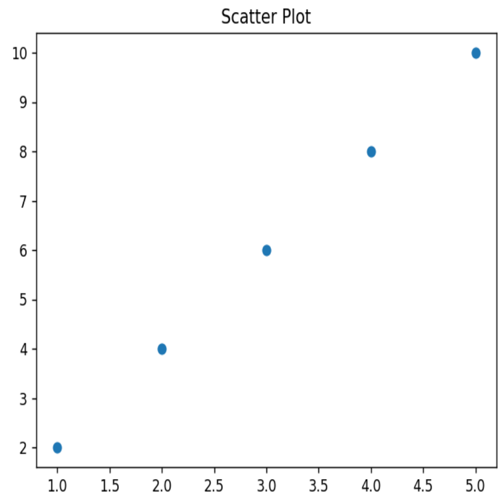
Implementing Matplotlib library in lab and use the charts: Bar plot, Scatter plot, Pie-chart, Donut- chart.

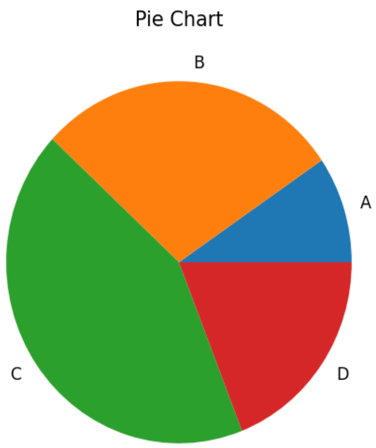
```
import matplotlib.pyplot as plt

# Bar Plot
categories = ['A', 'B', 'C']
values = [15, 10, 25]
plt.bar(categories, values)
plt.title("Bar Plot")
plt.show()

# Scatter Plot
x = [1, 2, 3, 4, 5]
y = [2, 4, 6, 8, 10]
plt.scatter(x, y)
plt.title("Scatter Plot")
plt.show()

# Pie Chart
sizes = [10, 30, 45, 20]
labels = ['A', 'B', 'C', 'D']
plt.pie(sizes, labels=labels)
plt.title("Pie Chart")
plt.show()

# Donut Chart
```

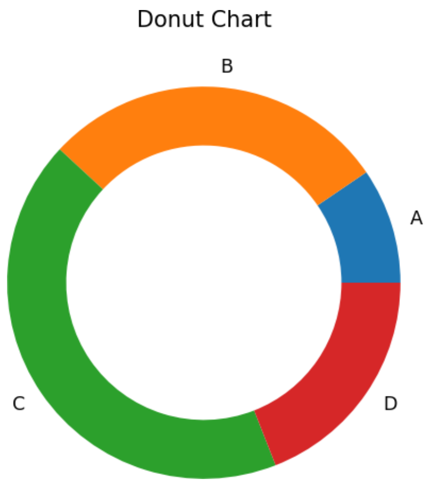
```python
plt.pie(sizes, labels=labels, wedgeprops=dict(width=0.3))
plt.title("Donut Chart")
plt.show()
```

OUTPUT: -









PROGRAM 13

Explain preprocessing of data on any real-life dataset.

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler

# Load the Iris dataset
iris = load_iris()

# Convert to a pandas DataFrame
data = pd.DataFrame(iris.data, columns=iris.feature_names)

# Add the target column to the DataFrame
data['species'] = iris.target

# Checking for missing values
print(data.isnull().sum())

# Standardizing the data (excluding the target column)
scaler = StandardScaler()
scaled_data = scaler.fit_transform(data.iloc[:, :-1])

# Print the standardized features
print(scaled_data)
```

OUTPUT: -

```
PS B:\MCA 3rd sem\AI & ML\AIML> python -u "b:\MCA 3rd sem\AI & ML\AIML\Preprocess
sepal length (cm)    0
sepal width (cm)     0
petal length (cm)    0
petal width (cm)     0
species              0
dtype: int64
[[-9.00681170e-01  1.01900435e+00 -1.34022653e+00 -1.31544430e+00]
 [-1.14301691e+00 -1.31979479e-01 -1.34022653e+00 -1.31544430e+00]
 [-1.38535265e+00  3.28414053e-01 -1.39706395e+00 -1.31544430e+00]
 [-1.50652052e+00  9.82172869e-02 -1.28338910e+00 -1.31544430e+00]
 [-1.02184904e+00  1.24920112e+00 -1.34022653e+00 -1.31544430e+00]
 [-5.37177559e-01  1.93979142e+00 -1.16971425e+00 -1.05217993e+00]
 [-1.50652052e+00  7.88807586e-01 -1.34022653e+00 -1.18381211e+00]
 [-1.02184904e+00  7.88807586e-01 -1.28338910e+00 -1.31544430e+00]
 [-1.74885626e+00 -3.62176246e-01 -1.34022653e+00 -1.31544430e+00]
 [-1.14301691e+00  9.82172869e-02 -1.28338910e+00 -1.44707648e+00]
 [-5.37177559e-01  1.47939788e+00 -1.28338910e+00 -1.31544430e+00]
 [-1.26418478e+00  7.88807586e-01 -1.22655167e+00 -1.31544430e+00]
 [-1.26418478e+00 -1.31979479e-01 -1.34022653e+00 -1.44707648e+00]
 [-1.87002413e+00 -1.31979479e-01 -1.51073881e+00 -1.44707648e+00]
 [-5.25060772e-02  2.16998818e+00 -1.45390138e+00 -1.31544430e+00]
 [-1.73673948e-01  3.09077525e+00 -1.28338910e+00 -1.05217993e+00]
 [-5.37177559e-01  1.93979142e+00 -1.39706395e+00 -1.05217993e+00]
 [-9.00681170e-01  1.01900435e+00 -1.34022653e+00 -1.18381211e+00]
 [-1.73673948e-01  1.70959465e+00 -1.16971425e+00 -1.18381211e+00]
 [-9.00681170e-01  1.70959465e+00 -1.28338910e+00 -1.18381211e+00]
 [-5.37177559e-01  7.88807586e-01 -1.16971425e+00 -1.31544430e+00]
 [-9.00681170e-01  1.47939788e+00 -1.28338910e+00 -1.05217993e+00]
 [-1.50652052e+00  1.24920112e+00 -1.56757623e+00 -1.31544430e+00]
 [-9.00681170e-01  5.58610819e-01 -1.16971425e+00 -9.20547742e-01]
 [-1.26418478e+00  7.88807586e-01 -1.05603939e+00 -1.31544430e+00]
 [-1.02184904e+00 -1.31979479e-01 -1.22655167e+00 -1.31544430e+00]
 [-1.02184904e+00  7.88807586e-01 -1.22655167e+00 -1.05217993e+00]
 [-7.79513300e-01  1.01900435e+00 -1.28338910e+00 -1.31544430e+00]
 [-7.79513300e-01  7.88807586e-01 -1.34022653e+00 -1.31544430e+00]
 [-1.38535265e+00  3.28414053e-01 -1.22655167e+00 -1.31544430e+00]
 [-1.26418478e+00  9.82172869e-02 -1.22655167e+00 -1.31544430e+00]
 [-5.37177559e-01  7.88807586e-01 -1.28338910e+00 -1.05217993e+00]
 [-7.79513300e-01  2.40018495e+00 -1.28338910e+00 -1.44707648e+00]
 [-4.16009689e-01  2.63038172e+00 -1.34022653e+00 -1.31544430e+00]
 [-1.14301691e+00  9.82172869e-02 -1.28338910e+00 -1.31544430e+00]
 [-1.02184904e+00  3.28414053e-01 -1.45390138e+00 -1.31544430e+00]
```

PROGRAM 14

Write a program to implement the graph coloring problem.

```
def graph_coloring(graph, num_colors):
    color = [-1] * len(graph)
    # Assign the first color to the first node
    color[0] = 0
    for node in range(1, len(graph)):
        # Find colors that are assigned to the neighboring nodes
        available_colors = [True] * num_colors
        for neighbor in graph[node]:
            if color[neighbor] != -1:
                available_colors[color[neighbor]] = False
        # Assign the first available color
        for clr in range(num_colors):
            if available_colors[clr]:
                color[node] = clr
                break
    return color
# Example graph as adjacency list
graph = {    0: [1, 2],    1: [0, 2],    2: [0, 1, 3],    3: [2]}
result = graph_coloring(graph, 3)
print("Assigned Colors:", result)
```

OUTPUT: -

```
PS B:\MCA 3rd sem\AI & ML\AIML> python -u "b
Assigned Colors: [0, 1, 2, 0]
PS B:\MCA 3rd sem\AI & ML\AIML>
```