1. (a) Create a text file named ”data.txt” and write the following data into it:

1, John, 25
2, Jane, 30
3, Bob, 22
4, Alice, 28

```
#Method 1:
#Syntax: fileObject = open(filename,mode)
file = open("data.txt","w")
try:
    file.write("1, John, 25\n2, Jane, 30\n3, Bob, 22\n4, Alice,
28")
finally:
    file.close()


#Method 2:
with open("data.txt","w") as file:
    file.write("1, John, 25\n")
    file.write("2, Jane, 30\n")
    file.write("3, Bob, 22\n")
    file.write("4, Alice, 28")
```

(b) Write a Python function `read_data` that reads the data from *“data.txt”* and returns a list of dictionaries, where each dictionary represents a record with keys *‘ID’, ‘Name’,* and *‘Age’.*

```
def read_data():
  data = []
  with open("data.txt", "r") as file:
     for line in file:
     fields = line.split(", ")
     data.append({
                    "ID": fields[0],
                    "Name": fields[1],
                    "Age": int(fields[2]),
                })
  return data
```

```
data = read_data()
for dicts in data:
     print(dicts)
```

(c) Write another function `write_data` that takes the file path and a list of dictionaries (similar to the output of the previous function) and writes the data into the file. Ensure that the file is overwritten with the new data.

```
def write_data(filename, data):
     with open(filename, "w") as file:
     for record in data:

file.write(f"{record['ID']},{record['Name']},{record['Age']}\n")

data = [
     {"ID": "5", "Name": "Johnny", "Age": 15},
     {"ID": "6", "Name": "SarahJane", "Age": 20},
     {"ID": "7", "Name": "Rob", "Age": 12},
     {"ID": "8", "Name": "Alyce", "Age": 18},
     ]

write_data("data.txt", data)
```

(d) Implement a function update age(file path, id, new age) that updates the age of the person with the given ID in the ”data.txt” file.

```
def read_data():
  data = []
  with open("data.txt", "r") as file:
     for line in file:
     fields = line.split(", ")
     data.append({
                    "ID": fields[0],
                    "Name": fields[1],
                    "Age": int(fields[2]),
                    })
  return data
```

```
def write_data(filename, data):
     with open(filename, "w") as file:
     for record in data:

file.write(f"{record['ID']},{record['Name']},{record['Age']}\n")

def update_age(file_path, id, new_age):
  data = read_data()


  for person in data:
     if person["ID"] == id:
     person["Age"] = new_age
     break

  write_data(file_path, data)

update_age("data.txt", "2", 35)
```

---

2. Implement a function called process_data(input_filename, output_filename) that reads a list of numbers from the file specified by input filename, squares each number, and writes the squared numbers to the file specified by output filename.

```
def process_data(input_filename, output_filename):

     with open(input_filename, 'r') as input_file:
     numbers = [float(line.strip()) for line in input_file]

     squared_numbers = [number**2 for number in numbers]

     with open(output_filename, 'w') as output_file:
     output_file.writelines(f"{number}\n" for number in
squared_numbers)

process_data("input.txt", "output.txt")
```

(a) Consider the following data stored in a file named data1.txt and Implement the fol-
lowing tasks in your Python program:

1,2,3,4,5
6,7,8,9,10
11,12,13,14,15
i. Read the data from data1.txt and store it in a NumPy array.

Solution :

```
import numpy as np

file_path = 'data1.txt'

try:
    data_array = np.loadtxt(file_path, delimiter=',')
    print("Data loaded successfully:")
    print(data_array)
except Exception as e:
    print(f"Error reading data from {file_path}: {e}")
```

ii. Calculate the mean and standard deviation for each set of numbers in the array.

solution :

```
 import numpy as np

file_path = 'data1.txt'

try:
    data_array = np.loadtxt(file_path, delimiter=',')
    print("Data loaded successfully:")
    print(data_array)

    means = np.mean(data_array, axis=1)
    std_devs = np.std(data_array, axis=1)

    for i in range(len(means)):
       print(f"Set {i + 1}: Mean = {means[i]}, Standard
Deviation = {std_devs[i]}")
```

```
except Exception as e:
    print(f"Error reading data from {file_path}: {e}")
```

iii.Create a new NumPy array that contains the mean and standard deviation for each set of numbers. Each row should represent one set of numbers, and the columns should be labeled appropriately.

Solution:

```
import numpy as np

file_path = 'data1.txt'

try:
    data_array = np.loadtxt(file_path, delimiter=',')
    print("Data loaded successfully:")
    print(data_array)

       means = np.mean(data_array, axis=1)
    std_devs = np.std(data_array, axis=1)

       result_array = np.column_stack((means, std_devs))

      print("\nNew array with mean and standard deviation:")
    print(result_array)

except Exception as e:
    print(f"Error reading data from {file_path}: {e}")
```

iv. Write the new array to a new text file named ”results.txt”. Each line in the file should correspond to one set of numbers, and the values should be separated by commas.
Test your program with the provided ”data.txt” file and ensure that the ”results.txt” file is generated correctly.

Solution :

```
import numpy as np

input_file_path = 'data1.txt'
output_file_path = 'results.txt'
```

```
try:
    data_array = np.loadtxt(input_file_path, delimiter=',')
    print("Data loaded successfully:")
    print(data_array)

       means = np.mean(data_array, axis=1)
    std_devs = np.std(data_array, axis=1)

      result_array = np.column_stack((means, std_devs))

       print("\nNew array with mean and standard deviation:")
    print(result_array)

      np.savetxt(output_file_path, result_array, delimiter=',',
header='Mean, Standard Deviation', comments='')

    print(f"\nResults written to {output_file_path}
successfully.")

except Exception as e:
    print(f"Error processing data from {input_file_path}: {e}")
```

(b) Write a function called calculate statistics(data) that takes a NumPy array as input
and returns a dictionary containing the following statistics:Median,Minimum,Maximum

Solution :

```
import numpy as np

def calculate_statistics(data):
    """
    Calculate median, minimum, and maximum statistics for a
NumPy array.

    Parameters:
    - data (numpy.ndarray): Input NumPy array.
```

```
    Returns:
    - dict: Dictionary containing the calculated statistics.
    """
    statistics = {
        'Median': np.median(data),
        'Minimum': np.min(data),
        'Maximum': np.max(data)
    }
    return statistics

try:
     data_array = np.loadtxt('data1.txt', delimiter=',')
    print("Data loaded successfully:")
    print(data_array)

    result_statistics = calculate_statistics(data_array)

       print("\nCalculated statistics:")
    for stat, value in result_statistics.items():
        print(f"{stat}: {value}")

except Exception as e:
    print(f"Error processing data: {e}")
```

(c) Write a function called multiply matrices(matrix1, matrix2) that takes two NumPy
matrices as input and returns their product.

Solution :

```
import numpy as np

def multiply_matrices(matrix1, matrix2):
    """
    Multiply two NumPy matrices.

    Parameters:
    - matrix1 (numpy.ndarray): First input matrix.
    - matrix2 (numpy.ndarray): Second input matrix.

    Returns:
```

```
    - numpy.ndarray: Resultant matrix after multiplication.
    """
    result_matrix = np.dot(matrix1, matrix2)
    return result_matrix

try:
      matrix1 = np.array([[1, 2], [3, 4]])
    matrix2 = np.array([[5, 6], [7, 8]])

       result_matrix = multiply_matrices(matrix1, matrix2)

       print("Matrix 1:")
    print(matrix1)

    print("\nMatrix 2:")
    print(matrix2)

    print("\nResultant matrix after multiplication:")
    print(result_matrix)

except Exception as e:
    print(f"Error multiplying matrices: {e}")
```

(d) Create a NumPy array arr with 10 random integers between 1 and 100 (inclusive).
Write a function called filter odd numbers(arr) that takes this array as input and returns a new array containing only the odd numbers from the original array.

Solution :

```
import numpy as np

def filter_odd_numbers(arr):
    """
    Filter odd numbers from a NumPy array.

    Parameters:
    - arr (numpy.ndarray): Input array.

    Returns:
    - numpy.ndarray: New array containing only the odd numbers.
    """
```

```
    odd_numbers = arr[arr % 2 != 0]
    return odd_numbers

arr = np.random.randint(1, 101, 10)

print("Original array:")
print(arr)

result_odd_numbers = filter_odd_numbers(arr)

print("\nFiltered array containing only odd numbers:")
print(result_odd_numbers)
```

(e) Create a NumPy array with 20 integers.
i. Use array slicing to extract the first half and the second half of the array.

Solution:

```
import numpy as np

original_array = np.random.randint(1, 101, 20)

print("Original array:")
print(original_array)

first_half = original_array[:10]
second_half = original_array[10:]

print("\nFirst half of the array:")
print(first_half)

print("\nSecond half of the array:")
print(second_half)
```

ii. Modify specific elements in the array using indexing.

Solution:

```
import numpy as np

original_array = np.random.randint(1, 101, 20)
```

```
print("Original array:")
print(original_array)

original_array[:3] = [100, 200, 300]

print("\nArray after modification:")
print(original_array)
```

(f) Generate two random NumPy arrays of any dimension with the same shape. Display both arrays.
i. Concatenate the two arrays horizontally and vertically. Display the results of both concatenations.

Solution:

```
import numpy as np

array1 = np.random.randint(1, 11, size=(3, 4))
array2 = np.random.randint(11, 21, size=(3, 4))

print("Array 1:")
print(array1)

print("\nArray 2:")
print(array2)

horizontal_concatenation = np.concatenate((array1, array2),
axis=1)

vertical_concatenation = np.concatenate((array1, array2),
axis=0)

print("\nHorizontal Concatenation:")
print(horizontal_concatenation)

print("\nVertical Concatenation:")
print(vertical_concatenation)
```

ii. Sort both arrays along a specified axis (choose any axis). Display the sorted Arrays.

Solution :

```
import numpy as np

array1 = np.random.randint(1, 11, size=(3, 4))
array2 = np.random.randint(11, 21, size=(3, 4))
print("Array 1:")
print(array1)

print("\nArray 2:")
print(array2)

sorted_array1 = np.sort(array1, axis=1)
sorted_array2 = np.sort(array2, axis=1)

print("\nSorted Array 1 along axis 1:")
print(sorted_array1)

print("\nSorted Array 2 along axis 1:")
print(sorted_array2)
```

iii. Perform element-wise addition, subtraction, and multiplication on the two arrays. Display the results of each operation.

Solution:

```
import numpy as np

array1 = np.random.randint(1, 11, size=(3, 4))
array2 = np.random.randint(11, 21, size=(3, 4))

print("Array 1:")
print(array1)

print("\nArray 2:")
print(array2)

addition_result = array1 + array2

subtraction_result = array1 - array2

multiplication_result = array1 * array2
```

```
print("\nElement-wise Addition:")
print(addition_result)

print("\nElement-wise Subtraction:")
print(subtraction_result)

print("\nElement-wise Multiplication:")
print(multiplication_result)
```