**1. Write a Python program to find the maximum and minimum value of a given flattened**

**array.**

**Expected Output:**

**Original flattened array:**

**[[0 1]**

**[2 3]]**

**Maximum value of the above flattened array:**

**3**

**Minimum value of the above flattened array:**

**0**

import numpy as np

# Define a 2x2 array

array = np.array([[0, 1], [2, 3]])

# Flatten the array

flattened\_array = array.flatten()

print("Original flattened array:")

print(array)

# Maximum and Minimum values

max\_val = np.max(flattened\_array)

min\_val = np.min(flattened\_array)

print("Maximum value of the above flattened array:")

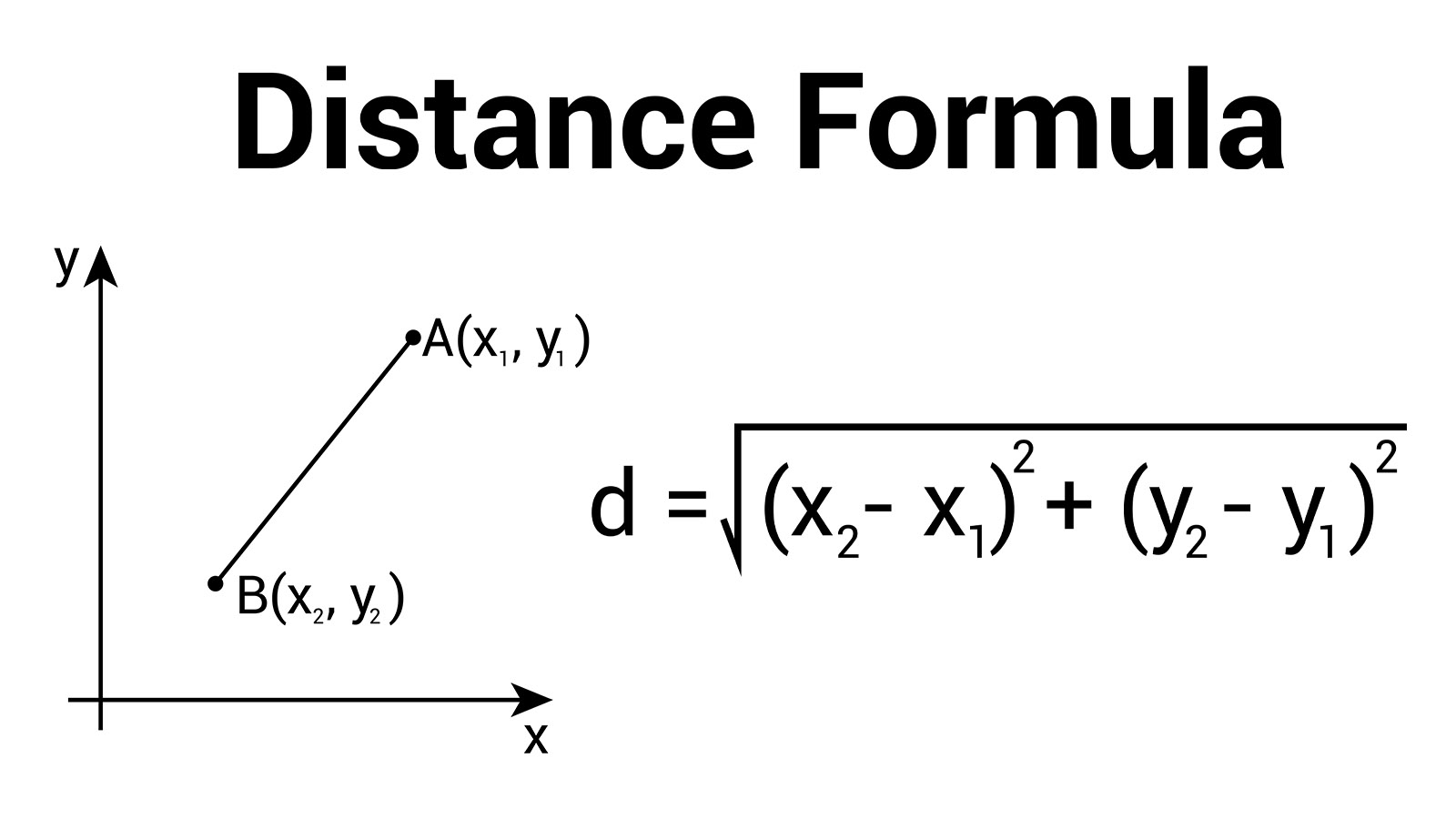
print(max\_val)

print("Minimum value of the above flattened array:")

print(min\_val)

**2. Write a python program to compute Euclidian Distance between two data points in a**

**dataset. [Hint: Use linalgo.norm function from NumPy]**



import numpy as np

# Sample data points

point1 = np.array([2, 3])

point2 = np.array([5, 7])

# Calculate Euclidean distance

distance = np.linalg.norm(point1 - point2)

# Output result

print(f"Euclidean Distance between {point1} and {point2} is: {distance}")

### **How it works:**

1. **np.array()** → Creates arrays for the two points.
2. **point1 - point2** → Finds the difference in each coordinate.
3. **np.linalg.norm()** → Calculates the magnitude (length) of the difference vector, which is the Euclidean distance.

**3. Create one dataframe of data values. Find out mean, range and IQR for this data.**

import pandas as pd

import numpy as np

# Create a DataFrame

data = {

'A': [10, 20, 30, 40, 50],

'B': [5, 15, 25, 35, 45],

'C': [2, 4, 6, 8, 10]

}

df = pd.DataFrame(data)

# Mean

mean\_values = df.mean()

# Range (max - min)

range\_values = df.max() - df.min()

# IQR (Q3 - Q1)

Q1 = df.quantile(0.25)

Q3 = df.quantile(0.75)

IQR = Q3 - Q1

# Display results

print("DataFrame:")

print(df, "\n")

print("Mean:\n", mean\_values, "\n")

print("Range:\n", range\_values, "\n")

print("Interquartile Range (IQR):\n", IQR)

### **Explanation:**

* **df.mean() → Mean of each column.**
* **df.max() - df.min() → Range for each column.**
* **df.quantile(0.25) & df.quantile(0.75) → Get Q1 and Q3.**
* **IQR = Q3 - Q1 → Measures spread of the middle 50% of data.**

**4. Write a python program to compute sum of Manhattan distance between all pairs of**

**points.**

import numpy as np

# Sample dataset (each row is a point [x, y])

points = np.array([

[1, 2],

[4, 6],

[7, 8]

])

# Function to calculate Manhattan distance

def manhattan\_distance(p1, p2):

return np.sum(np.abs(p1 - p2))

# Compute sum of Manhattan distances between all pairs

total\_distance = 0

n = len(points)

for i in range(n):

for j in range(i + 1, n): # Avoid duplicate pairs

dist = manhattan\_distance(points[i], points[j])

total\_distance += dist

print("Points:\n", points)

print("Sum of Manhattan distances between all pairs:", total\_distance)

**5. Write a NumPy program to compute the histogram of nums against the bins.**

**Sample Output:**

**nums: [0.5 0.7 1. 1.2 1.3 2.1]**

**bins: [0 1 2 3]**

**Result: (array([2, 3, 1], dtype=int64), array([0, 1, 2, 3]))**

import numpy as np

# Given data

nums = np.array([0.5, 0.7, 1.0, 1.2, 1.3, 2.1])

bins = np.array([0, 1, 2, 3])

# Compute histogram

hist, bin\_edges = np.histogram(nums, bins)

# Display results

print("nums:", nums)

print("bins:", bins)

print("Result:", (hist, bin\_edges))

**6. Create a dataframe for students’ information such name, graduation percentage and age.Display average age of students, average of graduation percentage. And, also describe**

**all basic statistics of data. (Hint: use describe()).**

import pandas as pd

# Create DataFrame

data = {

'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eva'],

'Graduation\_Percentage': [85, 78, 92, 88, 76],

'Age': [22, 23, 21, 24, 22]

}

df = pd.DataFrame(data)

# Average age

avg\_age = df['Age'].mean()

# Average graduation percentage

avg\_percentage = df['Graduation\_Percentage'].mean()

# Display results

print("Students DataFrame:")

print(df, "\n")

print(f"Average Age of Students: {avg\_age:.2f}")

print(f"Average Graduation Percentage: {avg\_percentage:.2f}\n")

# Describe basic statistics

print("Basic Statistics of Data:")

print(df.describe())

### **How it works:**

1. **df['Age'].mean()** → Calculates average age.
2. **df['Graduation\_Percentage'].mean()** → Calculates average graduation percentage.
3. **df.describe()** → Gives count, mean, std deviation, min, max, and quartiles for numeric columns.

**SET B:**

**1. Download iris dataset file. Read this csv file using read\_csv() function. Take samples**

**from entire dataset. Display maximum and minimum values of all numeric attributes.**

import pandas as pd

# Read the CSV file (assuming iris.csv is downloaded in the same folder)

df = pd.read\_csv("iris.csv")

# Display a random sample of 5 rows from entire dataset

sample\_data = df.sample(5)

# Maximum values of numeric attributes

max\_values = df.select\_dtypes(include='number').max()

# Minimum values of numeric attributes

min\_values = df.select\_dtypes(include='number').min()

# Output

print("Sample Data from Iris Dataset:\n", sample\_data, "\n")

print("Maximum values of numeric attributes:\n", max\_values, "\n")

print("Minimum values of numeric attributes:\n", min\_values)

### **Explanation:**

* **pd.read\_csv()** → Reads CSV file into DataFrame.
* **df.sample(n)** → Selects n random samples.
* **select\_dtypes(include='number')** → Selects only numeric columns for min/max calculation.
* **max() & min()** → Finds the maximum and minimum values for each numeric column.

**2. Continue with above dataset, find number of records for each distinct value of class**

**attribute. Consider entire dataset and not the samples.**

import pandas as pd

# Read the CSV file

df = pd.read\_csv("iris.csv")

# Count records for each distinct value of 'species' column

class\_counts = df['species'].value\_counts()

# Display results

print("Number of records for each class:")

print(class\_counts)

### **Explanation:**

* **df['species'].value\_counts()** → Counts how many times each class label appears.
* Works on the **entire dataset**, not just a sample.

**3. Display column-wise mean, and median for iris dataset from Q.4 (Hint: Use mean() and**

**median() functions of pandas dataframe.**

import pandas as pd

# Read the CSV file

df = pd.read\_csv("iris.csv")

# Select only numeric columns

numeric\_df = df.select\_dtypes(include='number')

# Column-wise mean

mean\_values = numeric\_df.mean()

# Column-wise median

median\_values = numeric\_df.median()

# Display results

print("Column-wise Mean:\n", mean\_values, "\n")

print("Column-wise Median:\n", median\_values)

**SET C:**

**1. Write a python program to find Minkowskii Distance between two points.**

import numpy as np

# Function to calculate Minkowski distance

def minkowski\_distance(p1, p2, order):

return np.sum(np.abs(p1 - p2) \*\* order) \*\* (1 / order)

# Example points

point1 = np.array([1, 2])

point2 = np.array([4, 6])

# Order (p) value

p = 3 # p=1 → Manhattan, p=2 → Euclidean

# Calculate Minkowski distance

distance = minkowski\_distance(point1, point2, p)

print(f"Minkowski Distance (p={p}) between {point1} and {point2} is: {distance}")

**2.Write a Python NumPy program to compute the weighted average along the specified**

**axis of a given flattened array.**

**From Wikipedia: The weighted arithmetic mean is similar to an ordinary arithmetic**

**mean (the most common type of average), except that instead of each of the data points**

**contributing equally to the final average, some data points contribute more than others.**

**The notion of weighted mean plays a role in descriptive statistics and also occurs in a**

**more general form in several other areas of mathematics.**

**Sample output:**

**Original flattened array:**

**[[0 1 2]**

**[3 4 5]**

**[6 7 8]]**

**Weighted average along the specified axis of the above flattened array:**

**[1.2 4.2 7.2]**

import numpy as np

# Original array

arr = np.arange(9).reshape(3, 3)

print("Original flattened array:")

print(arr)

# Define weights

weights = np.array([0.1, 0.5, 0.4]) # Each column gets these weights

# Compute weighted average along axis=0 (column-wise)

weighted\_avg = np.average(arr, axis=0, weights=weights)

print("Weighted average along the specified axis of the above flattened array:")

print(weighted\_avg)

### **Explanation:**

**np.arange(9).reshape(3, 3) → Creates a 3×3 array:  
  
[[0 1 2]**

**[3 4 5]**

**[6 7 8]]**

* **np.average(arr, axis=0, weights=weights) → Computes weighted average column-wise.**

**3. Write a NumPy program to compute cross-correlation of two given arrays.**

**Sample Output:**

**Original array1:**

**[0 1 3]**

**Original array2:**

**[2 4 5]**

**Cross-correlation of the said arrays:**

**[[2.33333333 2.16666667]**

**[2.16666667 2.33333333]]**

import numpy as np

# Original arrays

arr1 = np.array([0, 1, 3])

arr2 = np.array([2, 4, 5])

print("Original array1:")

print(arr1)

print("Original array2:")

print(arr2)

# Compute cross-correlation (normalized)

corr\_matrix = np.corrcoef(arr1, arr2)

print("Cross-correlation of the said arrays:")

print(corr\_matrix)

**Explanation:**

**np.corrcoef(arr1, arr2)** → Returns the Pearson correlation coefficient matrix.

The diagonal values will be **1** (correlation with itself).

The off-diagonal values are the correlation between the two arrays.

In this case, both arrays have **very similar increasing trends**, so correlation is high.

**4. Download any dataset from UCI (do not repeat it from set B). Read this csv file using**

**read\_csv() function. Describe the dataset using appropriate functions. Display mean**

**value of numeric attribute. Check if any data values are missing or not.**

import pandas as pd

# Step 1: Download dataset (Wine Quality Dataset from UCI)

# For demo purposes, we'll use the white wine quality dataset

url = "https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-white.csv"

# Step 2: Read CSV file (semicolon-separated in this dataset)

df = pd.read\_csv(url, sep=';')

# Step 3: Describe the dataset

print("Dataset Description:")

print(df.describe())

# Step 4: Display mean value of numeric attributes

print("\nMean of numeric attributes:")

print(df.mean(numeric\_only=True))

# Step 5: Check if there are any missing values

print("\nMissing values in each column:")

print(df.isnull().sum())

### **Explanation**

1. **pd.read\_csv()** → Loads CSV data into a DataFrame.
2. **df.describe()** → Gives statistical summary (count, mean, std, min, quartiles, max).
3. **df.mean()** → Calculates mean of numeric columns.
4. **df.isnull().sum()** → Checks for missing values in each column.

**5. Download nursery dataset from UCI. Split dataset on any one categorical attribute.**

**Compare the means of each split. (Use groupby)**

import pandas as pd

# Step 1: Load the dataset from UCI

url = "https://archive.ics.uci.edu/ml/machine-learning-databases/nursery/nursery.data"

col\_names = ["parents", "has\_nurs", "form", "children", "housing",

"finance", "social", "health", "class"]

df = pd.read\_csv(url, header=None, names=col\_names)

# Step 2: Check the first few rows

print("First rows:\n", df.head())

# Step 3: Split by one categorical attribute (e.g., 'parents')

grouped = df.groupby('parents')

# Step 4: For each group, compute the proportion of each target class

print("\nProportion of target classes within each 'parents' group:")

print(grouped['class'].value\_counts(normalize=True).unstack(fill\_value=0))

# Step 5 (Optional): Convert target class into dummy variables (one-hot) to compute group-wise averages

df\_onehot = pd.get\_dummies(df, columns=['class'])

grouped\_onehot = df\_onehot.groupby('parents')

print("\nMean of dummy target variables (i.e., proportion) per parents group:")

print(grouped\_onehot[[col for col in df\_onehot.columns if col.startswith('class\_')]].mean())

# Step 6: Check for missing values

print("\nMissing values present:")

print(df.isnull().sum())

### 

### **What This Does**

* **Loading: Reads the data via pd.read\_csv, naming each column appropriately.**
* **Grouping: Uses .groupby('parents') to split the dataset based on parents' category (e.g., usual, pretentious, great\_pret).**
* **Proportion Calculation:**
  + **Using value\_counts(normalize=True) computes the proportion of each class label within each group.**
* **Mean via One-Hot:**
  + **Converting class into dummy columns (e.g., class\_recommended, class\_priority, class\_not\_recom), then grouping by parents and computing the .mean() gives the same proportions in numeric form.**
* **Missing Values:**
  + **Using .isnull().sum() confirms there are no missing values in any columns.**

**6. Create one dataframe with 5 subjects and marks of 10 students for each subject. Find**

**arithmetic mean, geometric mean, and harmonic mean.**

import pandas as pd

import numpy as np

from scipy.stats import gmean, hmean

# Step 1: Create DataFrame

data = {

"Math": [78, 85, 69, 90, 88, 76, 95, 89, 84, 91],

"Physics": [82, 79, 88, 92, 75, 85, 90, 87, 80, 83],

"Chemistry": [74, 81, 77, 85, 80, 79, 88, 90, 82, 78],

"Biology": [69, 75, 80, 78, 84, 77, 85, 82, 79, 81],

"English": [88, 90, 85, 87, 82, 84, 86, 89, 91, 83]

}

df = pd.DataFrame(data)

print("Marks DataFrame:")

print(df)

# Step 2: Arithmetic Mean

arithmetic\_mean = df.mean(numeric\_only=True)

# Step 3: Geometric Mean

geometric\_mean = df.apply(lambda x: gmean(x), axis=0)

# Step 4: Harmonic Mean

harmonic\_mean = df.apply(lambda x: hmean(x), axis=0)

# Step 5: Display results

print("\nArithmetic Mean of each subject:")

print(arithmetic\_mean)

print("\nGeometric Mean of each subject:")

print(geometric\_mean)

print("\nHarmonic Mean of each subject:")

print(harmonic\_mean)

### How It Works

* df.mean() → Arithmetic mean for each subject.
* gmean() → Geometric mean (multiplicative average).
* hmean() → Harmonic mean (useful for rates/ratios).
* .apply() with axis=0 → Apply function column-wise.

**7. Download any csv file of your choice and display details about data using pandas**

**profiling. Show stats in HTML form.**

import pandas as pd

from ydata\_profiling import ProfileReport # install with: pip install ydata-profiling

# Step 1: Load CSV

df = pd.read\_csv("your\_file.csv") # replace with your CSV file name

# Step 2: Generate Profile Report

profile = ProfileReport(df, title="Dataset Profiling Report", explorative=True)

# Step 3: Save HTML report

profile.to\_file("report.html")

print("Profiling report saved as 'report.html'")