**Practical 1**

**Aim : Write a program which implements-“Data Cleaning” Smoothing by binning techniques mean, median and boundaries.**

**📌 Notes for Your Original Code**

These notes break down the structure and functionality of your Python program, making it easy for anyone to understand **how it works** and **why each part is necessary**.

**🔹 Overview of the Program**

Your program implements **data binning** using **Equi-Depth and Equi-Width binning** techniques.  
It also supports **three different smoothing methods**:  
1️⃣ **Mean Binning**  
2️⃣ **Median Binning**  
3️⃣ **Boundary Binning**

🔹 **User Input:**

* The program first asks the user to enter:
  + The number of bins
  + The binning method (Equi-Depth or Equi-Width)
  + The smoothing method (Mean, Median, or Boundary)

🔹 **Processing:**

* The program **divides data into bins** using the selected method.
* The selected **smoothing technique** is applied to replace bin values.

🔹 **Output:**

* The program prints the **original data, bins before smoothing, bins after smoothing, and final smoothed data.**

**🔹 Functions in the Code**

**1️⃣ makeEquiDepthBin(data, numOfBin) → list[list]**

📌 **Purpose:**

* Divides the dataset into **equal-sized bins** (Equi-Depth Binning).

📌 **How it Works:**

* The total number of elements is divided by numOfBin to determine binDepth (size of each bin).
* Each bin gets approximately the same number of elements.
* If the number of elements is not perfectly divisible, the last bin takes the remaining elements.

📌 **Example:**

python

CopyEdit

data = [10, 20, 30, 40, 50, 60]

numOfBin = 2

# Output: [[10, 20, 30], [40, 50, 60]]

**2️⃣ makeEquiWidthBin(data, numOfBin) → list[list]**

📌 **Purpose:**

* Divides the dataset into bins **with equal range (width)** (Equi-Width Binning).

📌 **How it Works:**

* The range of data is calculated using max(data) - min(data).
* Bin width is computed as (dataRange / numOfBin).
* Data points are assigned to bins based on these calculated bin edges.

📌 **Example:**

python

CopyEdit

data = [5, 15, 25, 35, 45, 55]

numOfBin = 2

# Output: [[5, 15, 25], [35, 45, 55]]

**3️⃣ binByMean(data, numOfBin, method) → list**

📌 **Purpose:**

* Applies **mean smoothing** by replacing each value in a bin with the **bin’s mean**.

📌 **How it Works:**

* Calls makeEquiDepthBin() or makeEquiWidthBin() based on the chosen method.
* Computes the **mean** of each bin using sum(bin) / len(bin).
* Replaces all values in that bin with the **computed mean**.

📌 **Example:**

python

CopyEdit

data = [10, 20, 30, 40, 50, 60]

numOfBin = 2

# Bins before smoothing: [[10, 20, 30], [40, 50, 60]]

# Bins after smoothing: [[20, 20, 20], [50, 50, 50]]

# Smoothed Data: [20, 20, 20, 50, 50, 50]

**4️⃣ binByMedian(data, numOfBin, method) → list**

📌 **Purpose:**

* Applies **median smoothing** by replacing each value in a bin with the **bin’s median**.

📌 **How it Works:**

* Calls makeEquiDepthBin() or makeEquiWidthBin() based on the chosen method.
* Computes the **median** of each bin:
  + If the bin length is odd, the median is the middle value.
  + If the bin length is even, the median is the average of the two middle values.
* Replaces all values in that bin with the **computed median**.

📌 **Example:**

python

CopyEdit

data = [10, 20, 30, 40, 50, 60]

numOfBin = 2

# Bins before smoothing: [[10, 20, 30], [40, 50, 60]]

# Bins after smoothing: [[20, 20, 20], [50, 50, 50]]

# Smoothed Data: [20, 20, 20, 50, 50, 50]

**5️⃣ binByBoundary(data, numOfBin, method) → list**

📌 **Purpose:**

* Applies **boundary smoothing** by replacing each value in a bin with the **nearest boundary value (min or max of bin).**

📌 **How it Works:**

* Calls makeEquiDepthBin() or makeEquiWidthBin() based on the chosen method.
* For each element in a bin:
  + If it is closer to the **minimum** value, replace it with min(bin).
  + If it is closer to the **maximum** value, replace it with max(bin).

📌 **Example:**

python

CopyEdit

data = [10, 20, 30, 40, 50, 60]

numOfBin = 2

# Bins before smoothing: [[10, 20, 30], [40, 50, 60]]

# Bins after smoothing: [[10, 10, 30], [40, 40, 60]]

# Smoothed Data: [10, 10, 30, 40, 40, 60]

**🔹 User Input & Execution Flow**

📌 **How User Input is Handled:**

* The user enters:
  1. **Number of bins**
  2. **Binning method:**
     + 'D' for **Equi-Depth Binning**
     + 'W' for **Equi-Width Binning**
  3. **Smoothing method:**
     + 'X' for **Mean Smoothing**
     + 'M' for **Median Smoothing**
     + 'B' for **Boundary Smoothing**

📌 **How Execution Flow Works:**

* Based on user input, the corresponding function (binByMean, binByMedian, or binByBoundary) is called.
* The function performs binning and smoothing.
* The results (bins before/after smoothing and final smoothed data) are printed.

📌 **Example User Input & Output:**

rust

CopyEdit

Enter no. of bins: 3

Enter method for Binning:

'D' for EquiDepth

'W' for EquiWidth

D

Enter parameter for Binning:

'X' for Mean

'M' for Median

'B' for Boundary

X

💡 **This means:**

* **Equi-Depth Binning** will be used.
* **Mean Smoothing** will be applied to each bin.

**🔹 Key Takeaways**

✅ **Equi-Depth vs. Equi-Width:**

* **Equi-Depth** divides data into **bins with equal number of elements**.
* **Equi-Width** divides data into **bins with equal range**.

✅ **Smoothing Methods:**

* **Mean Smoothing:** Replaces all values with the **bin’s mean**.
* **Median Smoothing:** Replaces all values with the **bin’s median**.
* **Boundary Smoothing:** Replaces values with **nearest boundary value**.

✅ **User-Friendly Input Handling:**

* The program **asks for input** and smoothly handles errors.

**💡 Why This Code is Useful?**

* **Data Preprocessing:** Helps in handling noisy data.
* **Data Mining & Analytics:** Used in histogram analysis and clustering.
* **Machine Learning:** Helps in feature engineering for better model performance.