Grouping and Binning in Data Science

Grouping is an important data preprocessing step in data science, used to **categorize, aggregate, and analyze data efficiently**.

It is commonly applied to **structured datasets** for tasks like **summarization, segmentation, and statistical analysis**.

1. Types of Grouping in Data Science

Grouping techniques depend on the data type and purpose:

A. Grouping by Categories (Categorical Grouping)

- Used when data has **discrete categories** (e.g., Gender, Product Type, City).
- Helps in **summarizing data** for each category.
- Example: Grouping customer purchases by product type.

B. Numerical Grouping (Binning)

- Used to **group continuous numerical values** into **bins**.
- Two common methods:
    1. **Equal-Width Binning** – Each bin has the same range (width).
    2. **Equal-Frequency Binning** – Each bin contains the same number of values.

C. Time-Based Grouping

- Used when working with **time-series data**.
- Examples:
    - Grouping sales data **by month or year**.
    - Aggregating website visits **by hour**.

D. Grouping in Clustering

- **Unsupervised Learning** technique where similar data points are grouped together.
- Example:
    - **K-Means Clustering** groups customers into similar segments based on spending habits.

2. Grouping in Pandas (Python)

Pandas provides powerful functions for **grouping and aggregating** data.

A. `groupby()` for Categorical Grouping

Example: Grouping sales data by category and finding the total sales.

```
import pandas as pd
# Sample Data
data = {'Category': ['A', 'B', 'A', 'B', 'A', 'C'],
        'Sales': [100, 200, 150, 300, 130, 400]}

df = pd.DataFrame(data)

# Group by Category and sum Sales

grouped_data = df.groupby('Category')['Sales'].sum()

print(grouped_data)
```

output:

```
Category

A    380

B    500

C    400
```

B. Numerical Grouping using cut() and qcut()

1. Equal-Width Binning:

df['Bins'] = pd.cut(df['Sales'], bins=3)

print(df)

2. Equal-Frequency Binning:

df['Bins'] = pd.qcut(df['Sales'], q=3)

print(df)

3. **Time-Based Grouping**

df['Date'] = pd.to_datetime(['2024-01-01', '2024-01-02', '2024-01-02', '2024-02-01'])

df.set_index('Date', inplace=True)

 # Group by month

monthly_sales = df.resample('M').sum()

print(monthly_sales)

Categorical Grouping Examples in Data Science

Categorical grouping is used to **summarize data based on distinct categories**. It helps analyze trends, compare groups, and perform aggregations.

Example 1: Grouping Sales by Product Category

📌 **Problem:** A store records sales for different product categories. Group the total sales by category.

Dataset:

Product	Category	Sales ($)
Laptop	Electronics	1000
Phone	Electronics	800
Shirt	Clothing	50
Jeans	Clothing	60
Headphones	Electronics	200
Jacket	Clothing	90

Solution: Group by Category & Sum Sales

import pandas as pd

Creating the dataset

```
data = {'Product': ['Laptop', 'Phone', 'Shirt', 'Jeans', 'Headphones', 'Jacket'],
        'Category': ['Electronics', 'Electronics', 'Clothing', 'Clothing', 'Electronics', 'Clothing'],
        'Sales': [1000, 800, 50, 60, 200, 90]}
df = pd.DataFrame(data)
# Group by Category and sum Sales
grouped_data = df.groupby('Category')['Sales'].sum()
print(grouped_data)
```

output:

```
Category
Clothing      200
Electronics  2000
```

Interpretation:

- **Electronics** generated **$2000** in sales.
- **Clothing** generated **$200** in sales.

Example 2: Grouping Employees by Department

- 📌 **Problem:** A company wants to know the **average salary** for each department.
- **Dataset:**

Employee	Department	Salary ($)
Alice	HR	5000
Bob	IT	7000
Charlie	IT	8000
David	HR	5500
Eve	Sales	6000

- **Solution: Group by Department & Calculate Average Salary**

```
data = {'Employee': ['Alice', 'Bob', 'Charlie', 'David', 'Eve'],
        'Department': ['HR', 'IT', 'IT', 'HR', 'Sales'],
```

'Salary': [5000, 7000, 8000, 5500, 6000]}

df = pd.DataFrame(data)

 # Group by Department and calculate average salary

grouped_data = df.groupby('Department')['Salary'].mean()

print(grouped_data)

output:

Department

HR 5250.0

IT 7500.0

Sales 6000.0

✅ **Interpretation:**

- **HR employees** have an **average salary of $5250**.
- **IT employees** earn **$7500** on average.
- **Sales employees** earn **$6000** on average.

Example 3: Grouping Students by Grade

- 📌 **Problem:** A school wants to count how many students belong to each grade.
- **Dataset:**

Student	Grade
Alice	A
Bob	B
Charlie	A
David	C
Eve	B
Frank	A

Example 4: Grouping Orders by City

- 📌 **Problem:** An e-commerce company wants to find the **total number of orders** from each city.

- **Dataset:**

Order ID	City	Orders
1001	New York	5
1002	Chicago	3
1003	New York	2
1004	Los Angeles	7
1005	Chicago	6

- **Solution: Group by City & Sum Orders**

Example 5: Grouping Patients by Disease Type

📌 **Problem:** A hospital wants to know the **number of patients per disease**.

Dataset:

Patient	Disease
John	Flu
Alice	Covid
Bob	Flu
David	Cancer
Eve	Covid
Frank	Covid

Solution: Count Patients per Disease

Summary of Use Cases for Categorical Grouping

Use Case	Grouping Category	Common Aggregation
Sales Analysis	Product Category	Sum (Total Sales)
Employee Salaries	Department	Mean (Average Salary)
Student Performance	Grade	Count (Number of Students)
Order Analysis	City	Sum (Total Orders)
Healthcare Data	Disease Type	Count (Patients per Disease)

Implementation of Binning Technique

There are two methods to implement binning in data mining, as shown below -

Equal Frequency Binning

This method involves dividing a continuous variable into a specified number of bins, each containing an equal number of observations. This method is useful for data with a large number of observations or when the data is skewed.

- **Each bin should have (approximately) the same number of values.**

For example, our input is - [6,9,14,17,19,31,54,66,81,97,164,189]

and

if we want to bin this into 3 intervals, then the output will have three bins/intervals as –

[6,9,14,17], [19,31,54,66], and [81,97,164,189].

Solve Example:

1. **You have the following list of ages:**
   ages = [5, 12, 17, 24, 33, 45, 52, 67, 75, 89]

From the above ages list, divide it into 3 bins where each bin contains (roughly) an equal number of values.

2. Exam Scores Binning

You have the following exam scores of students:
scores = [45, 55, 60, 62, 65, 70, 72, 74, 80, 85, 90, 92, 95, 98]

Divide the scores into **4 bins** with approximately equal numbers of values in each bin.

3. **Employee Salaries Binning**

A company has the following employee salaries (in dollars per month):
salaries = [2500, 2800, 3000, 3200, 3500, 3700, 4000, 4200, 4500, 4700, 5000, 5200, 5500, 6000]

Bin the salaries into **3 equal-sized groups**.

4. **House Prices Binning**

A real estate company collected house prices (in thousands of dollars):
house_prices = [120, 135, 150, 160, 175, 180, 190, 200, 215, 230, 250, 275, 300, 320, 350]

Divide the prices into **5 equal-sized bins**.

5. **Customer Purchase Amounts**

Given Purchases:
purchases = [5, 10, 15, 20, 25, 30, 40, 45, 50, 55, 60, 65, 75, 80, 85, 90, 100]
Group them into **4 bins** such that each bin contains (approximately) an equal number of values.

6. **Daily Step Count Binning**

A fitness app recorded the number of steps taken per day:
steps = [1000, 2500, 4000, 4500, 5000, 5500, 6000, 7000, 8000, 8500, 9000, 10000, 11000, 12000]

Bin these values into **4 equal-sized groups**.

Exampl Python Code for Binning:

Large Dataset

Generate **100 random numbers** from a uniform distribution between **10 and 100** and bin them into **10 equal-frequency bins**.

```
import numpy as np

import pandas as pd

data = np.random.randint(10, 100, size=100)      # Generate 100 random values

bins = pd.qcut(data, q=10)      # 10 equal-sized bins

 print(bins)
```

solutions:

Exam Scores Binning

Given Scores:
scores = [45, 55, 60, 62, 65, 70, 72, 74, 80, 85, 90, 92, 95, 98]
Steps:

1. There are **14** values, and we need **4 bins**.
2. Each bin should contain approximately **14 / 4 = 3 or 4 values**.
3. Sort the values: [45, 55, 60, 62, 65, 70, 72, 74, 80, 85, 90, 92, 95, 98].
4. Split them into 4 bins:
    - **Bin 1:** [45, 55, 60, 62]
    - **Bin 2:** [65, 70, 72, 74]
    - **Bin 3:** [80, 85, 90]
    - **Bin 4:** [92, 95, 98]

Employee Salaries Binning

Given Salaries:
salaries = [2500, 2800, 3000, 3200, 3500, 3700, 4000, 4200, 4500, 4700, 5000, 5200, 5500, 6000]
Steps:

1. **14 values** need to be split into **3 bins → each bin should contain around 4-5 values**.
2. Sort the values: [2500, 2800, 3000, 3200, 3500, 3700, 4000, 4200, 4500, 4700, 5000, 5200, 5500, 6000].
3. Split them into **3 bins**:
    - **Bin 1:** [2500, 2800, 3000, 3200, 3500]
    - **Bin 2:** [3700, 4000, 4200, 4500, 4700]
    - **Bin 3:** [5000, 5200, 5500, 6000]

House Prices Binning

Given House Prices:
house_prices = [120, 135, 150, 160, 175, 180, 190, 200, 215, 230, 250, 275, 300, 320, 350]

Customer Purchase Amounts

Given Purchases:
purchases = [5, 10, 15, 20, 25, 30, 40, 45, 50, 55, 60, 65, 75, 80, 85, 90, 100]

Daily Step Count Binning

Given Steps Data:
steps = [1000, 2500, 4000, 4500, 5000, 5500, 6000, 7000, 8000, 8500, 9000, 10000, 11000, 12000]

Note:

- **Equal-frequency binning** ensures each bin has an (almost) equal number of values.
- Python's **pd.qcut()** is great for equal-frequency binning.
- This technique is useful in **feature engineering** for machine learning.

Equal-Width B inning Explanation

In **Equal-Width Binning**, the range of values is divided into bins of equal size.
Each bin has the same width, calculated as:

$$\text{Bin Width} = \frac{(\text{Max Value - Min Value})}{\text{Number of Bins}}$$

Practice Problems

Problem 1: Exam Scores Binning

Given Exam Scores:
scores = [45, 55, 60, 62, 65, 70, 72, 74, 80, 85, 90, 92, 95, 98]

Bin them into **4 equal-width bins**.

Problem 2: Employee Salaries Binning

Given Salaries (in dollars per month):
salaries = [2500, 2800, 3000, 3200, 3500, 3700, 4000, 4200, 4500, 4700, 5000, 5200, 5500, 6000]

Bin them into **3 equal-width bins**.

Problem 3: House Prices Binning

Given House Prices (in thousands of dollars):
house_prices = [120, 135, 150, 160, 175, 180, 190, 200, 215, 230, 250, 275, 300, 320, 350]

Divide the values into **5 equal-width bins**.

Problem 4: Customer Purchase Amounts

Given Purchase Amounts (in dollars):
purchases = [5, 10, 15, 20, 25, 30, 40, 45, 50, 55, 60, 65, 75, 80, 85, 90, 100]

Group them into **4 equal-width bins**.

Problem 5: Daily Step Count Binning

Given Steps Data:
steps = [1000, 2500, 4000, 4500, 5000, 5500, 6000, 7000, 8000, 8500, 9000, 10000, 11000, 12000]

Bin them into **4 equal-width bins**.

Problem 6: Large Dataset

Generate **100 random numbers** between **10 and 100** and bin them into **10 equal-width bins**.

```
import numpy as np
import pandas as pd

data = np.random.randint(10, 100, size=100)  # Generate 100 random values
bins = pd.cut(data, bins=10)  # 10 equal-width bins
print(bins)
```

Solution 1: Exam Scores Binning

1. **Find Min & Max**:
   - Min = **45**, Max = **98**
2. **Bin Width Calculation**: $$\frac{(98-45)}{4} = 13.25$$ **Bins**:
   - **Bin 1:** 45 - 58.25 → [45, 55]
   - **Bin 2:** 58.25 - 71.5 → [60, 62, 65, 70]
   - **Bin 3:** 71.5 - 84.75 → [72, 74, 80, 85]
   - **Bin 4:** 84.75 - 98 → [90, 92, 95, 98]

Solution 2: Employee Salaries Binning

1. **Find Min & Max**:
   - Min = **2500**, Max = **6000**
2. **Bin Width Calculation**:

$$\frac{(6000-2500)}{3} = 1166.67$$

Bins:

- **Bin 1:** 2500 - 3666.67 → [2500, 2800, 3000, 3200, 3500]
- **Bin 2:** 3666.67 - 4833.33 → [3700, 4000, 4200, 4500, 4700]
- **Bin 3:** 4833.33 - 6000 → [5000, 5200, 5500, 6000]

Solution 3: House Prices Binning

Solution 4: Customer Purchase Amounts Binning

Solution 5: Daily Step Count Binning

Large Dataset (Python Solution)

```
import numpy as np
import pandas as pd

data = np.random.randint(10, 100, size=100)
bins = pd.cut(data, bins=10)  # 10 equal-width bins

df = pd.DataFrame({'Value': data, 'Bin': bins})
print(df.sort_values(by='Value'))
```