Name-Omkar Patil

Batch-B1

PRN-22610079

Assignment 1

Title:

Normalize the dataset using the three normalization types

Theory:

When working with real-world datasets, numerical attributes often come with very different ranges. For instance, the attribute “age” might take values between 20 and 70, while “income” could vary from 10,000 to 200,000. If such features are used directly, those with larger scales tend to overshadow the smaller ones in algorithms that rely on distance, similarity, or optimization steps.

To avoid this imbalance, we apply a process called normalization. Normalization modifies the scale of numerical values so that all features lie in a comparable range. This does not distort the relative difference between values but ensures that every feature has a balanced influence on the model. As a result, training becomes more stable, and predictive performance often improves.

Popular Normalization Methods:

1. Min-Max Scaling
    - Maps values to a predefined interval, usually between 0 and 1.
    - Maintains the original distribution but compresses the scale.
2. Standardization (Z-Score Method)
    - Transforms values so the mean becomes 0 and the standard deviation becomes 1.
    - Helps when attributes follow a normal distribution and is less affected by extreme values.
3. Decimal Scaling
    - Divides each value by a power of 10 until the largest absolute value lies within -1 and 1.
    - A simpler method, often used for quick rescaling.

Mathematical Expressions:

- **Min-Max Scaling:**

$$X' = \frac{X - Xmin}{Xmax - Xmin}$$

- **Z-Score:**

$$z = \frac{x - \mu}{\sigma}$$

μ = Mean

σ = Standard Deviation

- **Decimal Scaling:**

$$v' = \frac{v}{10^j}$$

Algorithm:

Input: A CSV file (expt1.csv) containing dataset
Output: Normalized dataset saved in normalized_output.csv

Steps:

1. **Start**
2. Open the input CSV file.
3. Read the header row (column names) and clean any spaces or quotes.
4. Display the list of column names with index numbers.
5. Ask the user to select one or more column numbers for normalization.
6. Ask the user to choose a normalization technique:
    - Option 1 → Min-Max Normalization
    - Option 2 → Z-Score Normalization
    - Option 3 → Decimal Scaling Normalization
7. For each selected column:
   a. Extract all numeric values from that column.
   b. Calculate the required statistics depending on the chosen method.
   c. Apply the normalization technique to transform each value.
   d. Replace the old values with the new normalized values.
8. Save the updated dataset into a new CSV file named normalized_output.csv.
9. Display a confirmation message that normalization is completed.
10. **Stop**

Code:

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <vector>
#include <string>
#include <cmath>
#include <algorithm>
using namespace std;

// ---------- Normalization Functions ----------

// Range Normalization (Min-Max)
void normalizeRange(vector<double>& values, double low, double high) {
    double mn = *min_element(values.begin(), values.end());
    double mx = *max_element(values.begin(), values.end());
    for (size_t i = 0; i < values.size(); i++) {
        if (mx == mn) values[i] = (low + high) / 2.0;
        else values[i] = ((values[i] - mn) / (mx - mn)) * (high - low) + low;
    }
}

// Standardization (Z-Score)
void normalizeStandard(vector<double>& values) {
    double sum = 0.0;
    for (double v : values) sum += v;
```

```
    double mean = sum / values.size();

    double sq_sum = 0.0;
    for (double v : values) sq_sum += (v - mean) * (v - mean);
    double sd = sqrt(sq_sum / values.size());

    for (size_t i = 0; i < values.size(); i++) {
        if (sd == 0) values[i] = 0.0;
        else values[i] = (values[i] - mean) / sd;
    }
}

// Decimal Scaling
void normalizeDecimal(vector<double>& values) {
    double maxAbs = 0;
    for (double v : values) maxAbs = max(maxAbs, fabs(v));

    int j = 0;
    while (maxAbs >= 1) {
        maxAbs /= 10;
        j++;
    }
    for (size_t i = 0; i < values.size(); i++) {
        values[i] = values[i] / pow(10, j);
    }
}
```

```
// ---------- CSV Writing ----------
void saveCSV(const string& file,
             const vector<double>& a,
             const vector<double>& b,
             const vector<double>& c,
             const vector<double>& d,
             const vector<string>& label) {
    ofstream out(file);
    out << "SepalLength,SepalWidth,PetalLength,PetalWidth,Class\n";
    for (size_t i = 0; i < a.size(); i++) {
        out << a[i] << "," << b[i] << "," << c[i] << "," << d[i] << "," << label[i] << "\n";
    }
    out.close();
}

// ---------- Main ----------
int main() {
    ifstream fin("expt1.csv");
    if (!fin.is_open()) {
        cout << "Error: expt1.csv not found!\n";
        return 1;
    }

    string row;
    getline(fin, row); // Skip header
    vector<double> sl, sw, pl, pw;
    vector<string> labels;
```

```
// Read dataset
while (getline(fin, row)) {
    stringstream ss(row);
    string val;
    vector<string> parts;
    while (getline(ss, val, ',')) parts.push_back(val);

    if (parts.size() == 5) {
        sl.push_back(stod(parts[0]));
        sw.push_back(stod(parts[1]));
        pl.push_back(stod(parts[2]));
        pw.push_back(stod(parts[3]));
        labels.push_back(parts[4]);
    }
}
fin.close();

if (sl.empty()) {
    cout << "Dataset is empty!\n";
    return 1;
}

// ---- Min-Max ----
double low, high;
cout << "Enter Min value: ";
cin >> low;
```

```
cout << "Enter Max value: ";
cin >> high;

vector<double> sl_mm = sl, sw_mm = sw, pl_mm = pl, pw_mm = pw;
normalizeRange(sl_mm, low, high);
normalizeRange(sw_mm, low, high);
normalizeRange(pl_mm, low, high);
normalizeRange(pw_mm, low, high);
saveCSV("iris_minmax.csv", sl_mm, sw_mm, pl_mm, pw_mm, labels);

// ---- Z-Score ----
vector<double> sl_z = sl, sw_z = sw, pl_z = pl, pw_z = pw;
normalizeStandard(sl_z);
normalizeStandard(sw_z);
normalizeStandard(pl_z);
normalizeStandard(pw_z);
saveCSV("iris_zscore.csv", sl_z, sw_z, pl_z, pw_z, labels);

// ---- Decimal Scaling ----
vector<double> sl_d = sl, sw_d = sw, pl_d = pl, pw_d = pw;
normalizeDecimal(sl_d);
normalizeDecimal(sw_d);
normalizeDecimal(pl_d);
normalizeDecimal(pw_d);
saveCSV("iris_decimal.csv", sl_d, sw_d, pl_d, pw_d, labels);

cout << "Normalization done. Files generated:\n";
```

```
    cout << " - iris_minmax.csv\n";
    cout << " - iris_zscore.csv\n";
    cout << " - iris_decimal.csv\n";

    return 0;
}
```

Output:

Min-Max Normalization:

```
SepalLength,SepalWidth,PetalLength,PetalWidth,Class
0.222222,0.625,0.0677966,0.0416667,Iris-setosa
0.166667,0.416667,0.0677966,0.0416667,Iris-setosa
0.111111,0.5,0.0508475,0.0416667,Iris-setosa
0.0833333,0.458333,0.0847458,0.0416667,Iris-setosa
0.194444,0.666667,0.0677966,0.0416667,Iris-setosa
0.305556,0.791667,0.118644,0.125,Iris-setosa
0.0833333,0.583333,0.0677966,0.0833333,Iris-setosa
0.194444,0.583333,0.0847458,0.0416667,Iris-setosa
0.0277778,0.375,0.0677966,0.0416667,Iris-setosa
0.166667,0.458333,0.0847458,0,Iris-setosa
0.305556,0.708333,0.0847458,0.0416667,Iris-setosa
0.138889,0.583333,0.101695,0.0416667,Iris-setosa
0.138889,0.416667,0.0677966,0,Iris-setosa
0,0.416667,0.0169492,0,Iris-setosa
0.416667,0.833333,0.0338983,0.0416667,Iris-setosa
0.388889,1,0.0847458,0.125,Iris-setosa
0.305556,0.791667,0.0508475,0.125,Iris-setosa
0.222222,0.625,0.0677966,0.0833333,Iris-setosa
0.388889,0.75,0.118644,0.0833333,Iris-setosa
```

Z-score output:

```
SepalLength,SepalWidth,PetalLength,PetalWidth,Class
-0.900681,1.03206,-1.34127,-1.31298,Iris-setosa
-1.14302,-0.124958,-1.34127,-1.31298,Iris-setosa
-1.38535,0.337848,-1.39814,-1.31298,Iris-setosa
-1.50652,0.106445,-1.28441,-1.31298,Iris-setosa
-1.02185,1.26346,-1.34127,-1.31298,Iris-setosa
-0.537178,1.95767,-1.17068,-1.05003,Iris-setosa
-1.50652,0.800654,-1.34127,-1.1815,Iris-setosa
-1.02185,0.800654,-1.28441,-1.31298,Iris-setosa
-1.74886,-0.356361,-1.34127,-1.31298,Iris-setosa
-1.14302,0.106445,-1.28441,-1.44445,Iris-setosa
-0.537178,1.49486,-1.28441,-1.31298,Iris-setosa
-1.26418,0.800654,-1.22754,-1.31298,Iris-setosa
-1.26418,-0.124958,-1.34127,-1.44445,Iris-setosa
-1.87002,-0.124958,-1.51187,-1.44445,Iris-setosa
-0.0525061,2.18907,-1.455,-1.31298,Iris-setosa
-0.173674,3.11468,-1.28441,-1.05003,Iris-setosa
-0.537178,1.95767,-1.39814,-1.05003,Iris-setosa
-0.900681,1.03206,-1.34127,-1.1815,Iris-setosa
-0.173674,1.72627,-1.17068,-1.1815,Iris-setosa
```

Decimal Scaling output:

```
SepalLength,SepalWidth,PetalLength,PetalWidth,Class
0.51,0.35,0.14,0.02,Iris-setosa
0.49,0.3,0.14,0.02,Iris-setosa
0.47,0.32,0.13,0.02,Iris-setosa
0.46,0.31,0.15,0.02,Iris-setosa
0.5,0.36,0.14,0.02,Iris-setosa
0.54,0.39,0.17,0.04,Iris-setosa
0.46,0.34,0.14,0.03,Iris-setosa
0.5,0.34,0.15,0.02,Iris-setosa
0.44,0.29,0.14,0.02,Iris-setosa
0.49,0.31,0.15,0.01,Iris-setosa
0.54,0.37,0.15,0.02,Iris-setosa
0.48,0.34,0.16,0.02,Iris-setosa
0.48,0.3,0.14,0.01,Iris-setosa
0.43,0.3,0.11,0.01,Iris-setosa
0.58,0.4,0.12,0.02,Iris-setosa
0.57,0.44,0.15,0.04,Iris-setosa
0.54,0.39,0.13,0.04,Iris-setosa
0.51,0.35,0.14,0.03,Iris-setosa
0.57,0.38,0.17,0.03,Iris-setosa
```

Conclusion:

Data normalization is an essential preprocessing step in data mining and machine learning. It ensures that all features contribute equally by bringing them to a comparable scale. Techniques like Min-Max, Z-Score, and Decimal Scaling improve the performance of algorithms that rely on distance, similarity, or gradients. By applying normalization to datasets, we achieve more accurate, reliable, and meaningful results.

Assignment 2

Title- To perform Data Cube operations (Slice, Dice, Roll-Up,Dril-Down,Pivot)

Theory:

OLAP is a powerful technology in data warehousing and data mining used for analyzing data from multiple perspectives. It helps organizations in decision-making by providing quick and interactive access to large amounts of data.

In OLAP, data is usually organized in a **multidimensional structure**, where information can be viewed and analyzed across different dimensions such as time, department, region, or product. Instead of just storing raw data, OLAP focuses on summarizing and analyzing it through various operations.

1. **Slice Operation**
    - The slice operation selects a single dimension of the data cube and creates a sub-cube for analysis.
    - Example: If a cube has dimensions **Product, Region, and Time**, slicing it by "Region = Asia" will give you a smaller cube with only data from Asia.
    - It helps in focusing on a single aspect while keeping other dimensions intact.
2. **Dice Operation**
    - The dice operation selects data based on **two or more dimensions** and creates a smaller sub-cube.
    - Example: If you choose "Region = Asia AND Product = Electronics," you get data only for Electronics in Asia.
    - It provides more flexibility than slice since multiple conditions can be applied.
3. **Roll-Up Operation**
    - Roll-up performs **data aggregation**, moving from detailed data to summarized data.
    - It works by climbing up a **hierarchy of dimensions**.
    - Example: Sales data can be rolled up from **Day → Month → Year**, or from **City → State → Country**.
    - This is useful for high-level decision-making.
4. **Drill-Down Operation**

- Drill-down is the reverse of roll-up, moving from summarized data to **more detailed data**.
- Example: A yearly sales report can be drilled down to see sales per month, then per week, and finally per day.
- It provides deeper insights by exploring the fine details of data.

5. **Pivot (Rotate) Operation**
   - Pivot allows reorienting the multidimensional view of data.
   - It changes how data is presented so that users can look at it from **different perspectives**.
   - Example: In a sales cube, you may initially view sales by **Product vs. Region**, and then pivot to view it by **Region vs. Product**.
   - It improves visualization and makes comparisons easier.

Algorithm :

Step 1: Start the program.

Step 2: Ask the user to enter the CSV file name.

Step 3: Open the CSV file.

- If the file is not found → Display error and stop.
- Else, read the file line by line.

Step 4: For each line (ignoring the header):

- Split the line into fields → Semester, University, Department, Subject, Score.
- Store the fields into a StudentRecord structure.
- Add the record to a vector data.

Step 5: If no records are loaded → Display “No data loaded” and stop.

Step 6: Display the **OLAP Menu** with options:

1. Show Original Data
2. Slice
3. Dice
4. Roll-Up

5. Drill-Down
6. Pivot
7. Exit

Operations

Option 1: Original Data

- Print all records from data.

Option 2: Slice

- Ask for a field (Semester / University / Department / Subject).
- Ask for a value.
- Display all records where the selected field matches the value.

Option 3: Dice

- Ask how many filters to apply.
- For each filter:
    - Ask for field name and value.
- Display only the records that match all the filters simultaneously.

Option 4: Roll-Up

- Ask for a field to group by (Semester / University / Department / Subject).
- For each unique value of the chosen field → Compute total score.
- Display totals grouped by the field.

Option 5: Drill-Down

- Ask the user where to start:
    - **Semester Totals** → Then show details of a chosen semester.
    - **University Totals** → Then show details of a chosen university.
    - **Department Totals** → Then show details of a chosen department.
- Display records that belong to the chosen subgroup.

Option 6: Pivot

- Ask for row field and column field.

- Create a pivot table where rows = values of row field, columns = values of column field.
- Fill each cell with the total score.
- Print the pivot table.

Option 0: Exit

- Terminate the program.

Step 7: Repeat menu until the user chooses Exit.

Step 8: End the program.

Code:

```
void sliceOperation(const vector<StudentRecord>& data, string field, string value) {
    cout << "\nSlice: " << field << " = " << value << endl;

    for (const auto& r : data) {
        string fieldValue;
        if (field == "Semester") fieldValue = r.semester;
        else if (field == "University") fieldValue = r.university;
        else if (field == "Department") fieldValue = r.department;
        else if (field == "Subject") fieldValue = r.subject;

        if (toLower(fieldValue) == toLower(value)) {
            printRecord(r);
        }
    }
}

// Dice operation
```

```
void diceOperation(const vector<StudentRecord>& data, map<string, string> filters) {
    cout << "\nDice" << endl;

    for (const auto& r : data) {
        bool match = true;
        for (const auto& filter : filters) {
            string fieldValue;
            if (filter.first == "Semester") fieldValue = r.semester;
            else if (filter.first == "University") fieldValue = r.university;
            else if (filter.first == "Department") fieldValue = r.department;
            else if (filter.first == "Subject") fieldValue = r.subject;

            if (toLower(fieldValue) != toLower(filter.second)) {
                match = false;
                break;
            }
        }

        if (match) {
            printRecord(r);
        }
    }
}

// Roll-up operation
void rollupOperation(const vector<StudentRecord>& data, string groupField) {
    cout << "\nRoll up: (Total Score by " << groupField << ")" << endl;
```

```
    map<string, int> totals;

    for (const auto& r : data) {
        string key;
        if (groupField == "Semester") key = r.semester;
        else if (groupField == "University") key = r.university;
        else if (groupField == "Department") key = r.department;
        else if (groupField == "Subject") key = r.subject;

        totals[key] += r.score;
    }

    for (const auto& pair : totals) {
        cout << pair.first << " -> " << pair.second << endl;
    }
}

// Drill-down operation
void drilldownOperation(const vector<StudentRecord>& data) {
    cout << "\nDrill-down" << endl;
    cout << "Select a level to drill down from:" << endl;
    cout << "1. Semester totals" << endl;
    cout << "2. University totals" << endl;
    cout << "3. Department totals" << endl;

    string choice;
```

```
cout << "Enter your choice (1-3): ";
cin >> choice;

if (choice == "1") {
    // Show semester totals first, then drill down
    cout << "\nSemester Totals:" << endl;
    map<string, int> semesterTotals;
    for (const auto& r : data) {
        semesterTotals[r.semester] += r.score;
    }
    for (const auto& pair : semesterTotals) {
        cout << pair.first << " -> " << pair.second << endl;
    }

    cout << "\nEnter semester to drill down (e.g., 2023-1): ";
    string selectedSemester;
    cin >> selectedSemester;

    cout << "\nDetailed data for " << selectedSemester << ":" << endl;
    for (const auto& r : data) {
        if (r.semester == selectedSemester) {
            printRecord(r);
        }
    }
}
else if (choice == "2") {
    // Show university totals first, then drill down
```

```
    cout << "\nUniversity Totals:" << endl;
    map<string, int> universityTotals;
    for (const auto& r : data) {
        universityTotals[r.university] += r.score;
    }
    for (const auto& pair : universityTotals) {
        cout << pair.first << " -> " << pair.second << endl;
    }

    cout << "\nEnter university to drill down: ";
    string selectedUniversity;
    cin.ignore();
    getline(cin, selectedUniversity);

    cout << "\nDetailed data for " << selectedUniversity << ":" << endl;
    for (const auto& r : data) {
        if (toLower(r.university) == toLower(selectedUniversity)) {
            printRecord(r);
        }
    }
}
else if (choice == "3") {
    // Show department totals first, then drill down
    cout << "\nDepartment Totals:" << endl;
    map<string, int> departmentTotals;
    for (const auto& r : data) {
        departmentTotals[r.department] += r.score;
```

```
        }
        for (const auto& pair : departmentTotals) {
            cout << pair.first << " -> " << pair.second << endl;
        }

        cout << "\nEnter department to drill down: ";
        string selectedDepartment;
        cin.ignore();
        getline(cin, selectedDepartment);

        cout << "\nDetailed data for " << selectedDepartment << ":" << endl;
        for (const auto& r : data) {
            if (toLower(r.department) == toLower(selectedDepartment)) {
                printRecord(r);
            }
        }
    }
    else {
        cout << "Invalid choice!" << endl;
    }
}

// Pivot operation
void pivotOperation(const vector<StudentRecord>& data, string rowField, string colField) {
    cout << "\nPivot: " << rowField << " vs " << colField << endl;

    map<string, map<string, int>> pivot;
```

```
set<string> colValues;

for (const auto& r : data) {
    string row, col;

    if (rowField == "Semester") row = r.semester;
    else if (rowField == "University") row = r.university;
    else if (rowField == "Department") row = r.department;
    else if (rowField == "Subject") row = r.subject;

    if (colField == "Semester") col = r.semester;
    else if (colField == "University") col = r.university;
    else if (colField == "Department") col = r.department;
    else if (colField == "Subject") col = r.subject;

    colValues.insert(col);
    pivot[row][col] += r.score;
}

// Print header
cout << rowField;
for (const auto& col : colValues) {
    cout << "\t" << col;
}
cout << endl;

// Print data
```

```
    for (const auto& rowPair : pivot) {
        cout << rowPair.first;
        for (const auto& col : colValues) {
            auto it = rowPair.second.find(col);
            cout << "\t" << (it != rowPair.second.end() ? it->second : 0);
        }
        cout << endl;
    }
}
```

Algorithm:

Step 1: Start the Program

- Begin execution.

Step 2: Input CSV File

- Prompt the user to enter the name of the CSV file.
- Open the file for reading.
- If the file is not found → display an error and stop.
- Else, read each line of the CSV file.
- Skip the header line.
- For each record:
    - Extract **Semester, University, Department, Subject, Score**.
    - Store them in a StudentRecord structure.
    - Add the record to the list data.

Step 3: Display Menu

- Show the user the available OLAP operations:
    1. Original Data
    2. Slice
    3. Dice

4. Roll-up
5. Drill-down
6. Pivot
7. Exit

Step 4: Menu Operations

(a) Original Data

- Print all records stored in data.

(b) Slice Operation

- Ask the user for a field (Semester / University / Department / Subject).
- Ask for the value to filter.
- Traverse all records:
    - If the record's field matches the value → display it.

(c) Dice Operation

- Ask the user how many filters they want.
- Collect field-value pairs (e.g., Department = "CSE", Semester = "2023-1").
- Traverse all records:
    - Check if the record satisfies all filter conditions.
    - If yes → display it.

(d) Roll-up Operation

- Ask the user for a grouping field (Semester / University / Department / Subject).
- Create a dictionary to store totals.
- For each record:
    - Add the score to the total of its group.
- Display total scores grouped by the chosen field.

(e) Drill-down Operation

- Ask the user for a level (Semester, University, Department).
- Show total scores at that level.

- Ask the user to select one specific item (e.g., a semester).
- Display detailed records for that selection.

(f) Pivot Operation

- Ask the user for **row field** and **column field**.
- Build a 2D structure (row vs column).
- For each record:
    - Add score to the correct row-column cell.
- Print the pivot table (rows as chosen row field, columns as chosen column field).

Step 5: Repeat Menu

- After completing an operation, return to the menu.
- Continue until the user selects **0. Exit**.

Step 6: End Program

- Display "Exiting…" and stop execution.

Output:

1.Original data :

```
* OLAP Menu *
1. Original Data
2. Slice
3. Dice
4. Roll-Up
5. Drill-Down
6. Pivot
0. Exit
Enter your choice: 1

Original Data
2023-1 | IIT Bombay | Computer Science | Data Structures | 85
2023-1 | IIT Bombay | Computer Science | Algorithms | 78
2023-1 | IIT Bombay | Computer Science | Database Systems | 92
2023-1 | IIT Delhi | Mathematics | Calculus | 88
2023-1 | IIT Delhi | Mathematics | Linear Algebra | 91
2023-1 | IIT Madras | Physics | Mechanics | 76
2023-1 | IIT Madras | Physics | Electromagnetism | 83
2023-2 | IIT Bombay | Computer Science | Data Structures | 87
2023-2 | IIT Bombay | Computer Science | Algorithms | 82
2023-2 | IIT Bombay | Computer Science | Database Systems | 89
```

1.Slice :

```
Enter your choice: 2
Enter field to slice by (Semester/University/Department/Subject): Semester
Enter value for Semester: 2023-1

Slice: Semester = 2023-1
2023-1 | IIT Bombay | Computer Science | Data Structures | 85
2023-1 | IIT Bombay | Computer Science | Algorithms | 78
2023-1 | IIT Bombay | Computer Science | Database Systems | 92
2023-1 | IIT Delhi | Mathematics | Calculus | 88
2023-1 | IIT Delhi | Mathematics | Linear Algebra | 91
2023-1 | IIT Madras | Physics | Mechanics | 76
2023-1 | IIT Madras | Physics | Electromagnetism | 83
2023-1 | BITS Pilani | Computer Science | Machine Learning | 89
2023-1 | BITS Pilani | Computer Science | Computer Networks | 84
2023-1 | NIT Trichy | Electronics | Digital Electronics | 91
2023-1 | NIT Trichy | Electronics | Microprocessors | 86
```

2.Dice :

```
Enter your choice: 3
How many filters? 2
Enter field name (Semester/University/Department/Subject): University
Enter value for University: IIT Bombay
Enter field name (Semester/University/Department/Subject): Department
Enter value for Department: Computer Science

Dice
2023-1 | IIT Bombay | Computer Science | Data Structures | 85
2023-1 | IIT Bombay | Computer Science | Algorithms | 78
2023-1 | IIT Bombay | Computer Science | Database Systems | 92
2023-2 | IIT Bombay | Computer Science | Data Structures | 87
2023-2 | IIT Bombay | Computer Science | Algorithms | 82
2023-2 | IIT Bombay | Computer Science | Database Systems | 89
2024-1 | IIT Bombay | Computer Science | Data Structures | 90
2024-1 | IIT Bombay | Computer Science | Algorithms | 85
2024-1 | IIT Bombay | Computer Science | Database Systems | 93
```

3.Roll up

```
Enter your choice: 4
Enter field to group by (Semester/University/Department/Subject): Semester

Roll up: (Total Score by Semester)
2023-1 -> 943
2023-2 -> 965
2024-1 -> 995
```

4.Drill down :

```
Enter semester to drill down (e.g., 2023-1): 2024-1

Detailed data for 2024-1:
2024-1 | IIT Bombay | Computer Science | Data Structures | 90
2024-1 | IIT Bombay | Computer Science | Algorithms | 85
2024-1 | IIT Bombay | Computer Science | Database Systems | 93
2024-1 | IIT Delhi | Mathematics | Calculus | 87
2024-1 | IIT Delhi | Mathematics | Linear Algebra | 96
2024-1 | IIT Madras | Physics | Mechanics | 81
2024-1 | IIT Madras | Physics | Electromagnetism | 88
2024-1 | BITS Pilani | Computer Science | Machine Learning | 95
2024-1 | BITS Pilani | Computer Science | Computer Networks | 91
2024-1 | NIT Trichy | Electronics | Digital Electronics | 96
2024-1 | NIT Trichy | Electronics | Microprocessors | 93
```

5.Pivot :

```
Enter your choice: 6
Enter row field: University
Enter column field: Semester

Pivot: University vs Semester
University      2023-1  2023-2  2024-1
BITS Pilani     173     180     186
IIT Bombay      255     258     268
IIT Delhi       179     179     183
IIT Madras      159     165     169
NIT Trichy      177     183     189
```

Conclusion :

Through this study, we explored how different **OLAP operations** such as Slice, Dice, Roll-up, Drill-down, and Pivot can be applied on datasets using C++. These operations help in analyzing data from different perspectives, making it easier to summarize, filter, and explore details. Implementing these operations shows the importance of structured data handling and highlights how concepts of data warehousing and analytics can be practically applied to real-world datasets.

Assignment -3

Title-

Calculation of T-Weight and D-Weight for Dataset Analysis

Theory:

In dataset analysis, especially in education, business, or survey-based studies, it is often necessary to understand both the **overall contribution of each record to the dataset** and the **internal distribution of values within each record**. To achieve this, two important measures are used: **T-Weight (Total Weight)** and **D-Weight (Distribution Weight)**.

---

1. T-Weight (Total Weight)

- **Definition**: T-Weight represents the proportion of an individual's value in a subject or category **relative to the total dataset value of that subject**.
- **Purpose**: It helps identify *how much an individual contributes to the collective dataset*.
- **Interpretation**: A higher T-Weight means that the individual has a greater share in that category compared to others.

◆ Example:

If the total marks in Hindi for the whole class are 500 and a student scores 50 in Hindi, then the T-Weight of that student in Hindi is **10%**.

This measure gives a **global perspective**, showing how an individual compares to the entire dataset.

---

2. D-Weight (Distribution Weight)

- **Definition**: D-Weight represents the proportion of a value in a subject **relative to the individual's own total across all subjects**.
- **Purpose**: It helps understand the **internal distribution of an individual's performance**.
- **Interpretation**: A higher D-Weight in a subject means that the subject forms a larger part of that individual's overall total.

◆ Example:

If a student's total marks are 100, with 40 in Hindi and 60 in English:

- Hindi D-Weight = 40%
- English D-Weight = 60%

This measure gives a **personal perspective**, showing how the individual divides their performance among different categories.

---

3. Significance in Dataset Analysis

- **T-Weight** allows comparisons across a group, making it useful for identifying top contributors, trends, or imbalances at the dataset level.
- **D-Weight** highlights internal patterns within each record, useful for understanding strengths, weaknesses, or preferences.
- Together, they provide both a **macro view (dataset-wide contribution)** and a **micro view (individual distribution)** of data.

Algorithm:

Step 1: Input Dataset

- Collect the dataset in tabular form where rows represent individuals (e.g., students, products) and columns represent attributes (e.g., subjects, features).

Step 2: Compute Column Totals

- For each column (attribute), calculate the sum of values across all individuals.
- This gives the **overall dataset total for each attribute**.

Step 3: Compute Row Totals

- For each row (individual), calculate the sum of values across all attributes.
- This gives the **individual total across attributes**.

Step 4: Calculate T-Weight

- For each cell, compare the individual's value with the total of its column.
- Store this proportion as the **T-Weight of that value**.

Step 5: Calculate D-Weight

- For each cell, compare the individual's value with the total of its row.
- Store this proportion as the **D-Weight of that value**.

Step 6: Store Results

- Arrange both T-Weight and D-Weight values in separate tables for interpretation.

Step 7: Analysis and Interpretation

- Use T-Weight table to analyze contribution of each individual to the overall dataset.
- Use D-Weight table to analyze distribution pattern within each individual.

Code:

```
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <iomanip>
#include <sstream>

using namespace std;

class Student {
public:
    string name;
    double hindi_marks;
    double english_marks;

    Student(string n, double h, double e) : name(n), hindi_marks(h), english_marks(e) {}
};

int main() {
    vector<Student> students;

    // Read data from CSV file
```

```
ifstream file("students_study.csv");
if (!file.is_open()) {
   cout << "Error: Could not open students_study.csv" << endl;
   return 1;
}

string line;
// Skip header
getline(file, line);

while (getline(file, line)) {
   stringstream ss(line);
   string name, hindi_str, english_str;

   getline(ss, name, ',');
   getline(ss, hindi_str, ',');
   getline(ss, english_str, ',');

   double hindi = stod(hindi_str);
   double english = stod(english_str);

   students.push_back(Student(name, hindi, english));
}
file.close();

// Calculate totals
double total_hindi = 0, total_english = 0, total_all = 0;
```

```
for (const auto& s : students) {
    total_hindi += s.hindi_marks;
    total_english += s.english_marks;
    total_all += s.hindi_marks + s.english_marks;
}

// Write output to CSV
ofstream outfile("output.csv");
if (!outfile.is_open()) {
    cout << "Error: Could not create output.csv" << endl;
    return 1;
}

// Write header
outfile << "Student,Hindi_Marks,T-Weight(Hindi),D-Weight(Hindi),"
        << "English_Marks,T-Weight(English),D-Weight(English),"
        << "Both,T-Weight(Both),D-Weight(Both)" << endl;

// Process each student
for (const auto& s : students) {
    double total_student = s.hindi_marks + s.english_marks;

    // Calculate T-weights
    double t_weight_hindi = (total_hindi > 0) ? (s.hindi_marks / total_hindi) * 100 : 0;
    double t_weight_english = (total_english > 0) ? (s.english_marks / total_english) * 100 : 0;
    double t_weight_both = (total_all > 0) ? (total_student / total_all) * 100 : 0;
```

```
    // Calculate D-weights
    double d_weight_hindi = (total_student > 0) ? (s.hindi_marks / total_student) * 100 : 0;
    double d_weight_english = (total_student > 0) ? (s.english_marks / total_student) * 100 : 0;
    double d_weight_both = 100.0;

    // Write student data
    outfile << s.name << ","
            << fixed << setprecision(1) << s.hindi_marks << ","
            << fixed << setprecision(2) << t_weight_hindi << ","
            << fixed << setprecision(2) << d_weight_hindi << ","
            << fixed << setprecision(1) << s.english_marks << ","
            << fixed << setprecision(2) << t_weight_english << ","
            << fixed << setprecision(2) << d_weight_english << ","
            << fixed << setprecision(1) << total_student << ","
            << fixed << setprecision(2) << t_weight_both << ","
            << fixed << setprecision(2) << d_weight_both << endl;
}

// Write totals
outfile << "Total,"
        << fixed << setprecision(1) << total_hindi << ",100.00,100.00,"
        << fixed << setprecision(1) << total_english << ",100.00,100.00,"
        << fixed << setprecision(1) << total_all << ",100.00,100.00" << endl;

outfile.close();

cout << "Output saved to output.csv" << endl;
```

```
    return 0;
}
```

Output:

	A	B	C	D	E	F	G	H	I	J	K
1	Student	Hindi_Mar	T-Weight(H	D-Weight(H	English_Ma	T-Weight(E	D-Weight(E	Both	T-Weight(E	D-Weight(Both)	
2	Priya	85	9.83	52.15	78	9.04	47.85	163	9.43	100	
3	Rahul	92	10.64	51.11	88	10.2	48.89	180	10.42	100	
4	Anjali	76	8.79	48.1	82	9.5	51.9	158	9.14	100	
5	Vikram	89	10.29	49.44	91	10.54	50.56	180	10.42	100	
6	Meera	94	10.87	52.51	85	9.85	47.49	179	10.36	100	
7	Arjun	81	9.36	50.62	79	9.15	49.38	160	9.26	100	
8	Kavya	87	10.06	48.33	93	10.78	51.67	180	10.42	100	
9	Aditya	90	10.4	51.14	86	9.97	48.86	176	10.19	100	
10	Zara	83	9.6	48.26	89	10.31	51.74	172	9.95	100	
11	Rohan	88	10.17	48.89	92	10.66	51.11	180	10.42	100	
12	Total	865	100	100	863	100	100	1728	100	100	
13											

Conclusion:

The calculation of T-Weight and D-Weight provides an effective method for analyzing datasets from two perspectives. T-Weight highlights the contribution of each individual entry towards the overall dataset, while D-Weight emphasizes the internal distribution of values within a single record. Together, they allow a balanced understanding of both global and local data patterns. By applying these measures, complex datasets can be better interpreted, comparisons become clearer, and decision-making becomes more data-driven.

Assignment-4

Title-

Calculation of Five-Number Summary and Detection of Outliers in a Dataset

Theory:

The five number summary is a basic but powerful descriptive statistic used to understand the spread and characteristics of data. It condenses a dataset into five key values that give an overall picture of its distribution.

1. **Minimum** – the lowest observation that shows the starting point of the data.
2. **First Quartile (Q1)** – marks the value below which one-fourth of the data lies.
3. **Median** – represents the middle position of the data, dividing it into two equal halves.
4. **Third Quartile (Q3)** – indicates the value below which three-fourths of the data falls.
5. **Maximum** – the highest observation that shows the end point of the data.

Together, these values describe the center, spread, and range of the dataset. They are also used to identify variability and detect extreme observations known as outliers.

The interquartile range, calculated from Q1 and Q3, is important because it focuses on the central portion of the data while ignoring extreme values. Using this, limits are set to check whether any values fall too far from the normal spread.

The method is widely used in statistics, data analysis, and machine learning as it quickly summarizes data without needing complex calculations. It also serves as the basis for graphical tools like boxplots, which visually represent the five number summary and highlight outliers clearly.

Algorithm: Five-Number Summary and Outlier Detection

1. **Start**
2. Read the dataset from a CSV file.
3. Store each entry with category, name, value, and description.
4. Extract values from the dataset based on required categories.
5. For each category:
    - Sort the values in ascending order.
    - Find the median of the entire dataset.
    - Divide the dataset into lower half and upper half.

- Calculate Q1 from the lower half and Q3 from the upper half.
- Compute the interquartile range as Q3 – Q1.
- Determine lower bound as Q1 – 1.5 × IQR and upper bound as Q3 + 1.5 × IQR.
- Identify the lower whisker as the smallest value within the bound.
- Identify the upper whisker as the largest value within the bound.
- Mark values below the lower bound or above the upper bound as outliers.

6. Display the results including Q1, median, Q3, IQR, whiskers, and outliers.
7. **Stop**

Code:

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <iomanip>
#include <string>
#include <numeric>
#include <fstream>
#include <sstream>

using namespace std;

// Structure to hold the five-number summary
struct FiveNumberSummary {
    double Q1;
    double median;
    double Q3;
    double IQR;
```

```
    double lowerWhisker;
    double upperWhisker;
    vector<double> outliers;
};

// Structure to hold data entries
struct DataEntry {
    string category;
    string name;
    double value;
    string description;
};

// Function to calculate median
double median(vector<double>& data) {
    int n = data.size();
    if (n % 2 == 0) {
        return (data[n/2 - 1] + data[n/2]) / 2.0;
    } else {
        return data[n/2];
    }
}

// Function to calculate five-number summary with outliers
FiveNumberSummary fiveNumberSummaryWithOutliers(vector<double>& data) {
    vector<double> sortedData = data;
    sort(sortedData.begin(), sortedData.end());
```

```
int n = sortedData.size();
double Q2 = median(sortedData);

vector<double> lowerHalf, upperHalf;

if (n % 2 == 0) {
    lowerHalf = vector<double>(sortedData.begin(), sortedData.begin() + n/2);
    upperHalf = vector<double>(sortedData.begin() + n/2, sortedData.end());
} else {
    lowerHalf = vector<double>(sortedData.begin(), sortedData.begin() + n/2);
    upperHalf = vector<double>(sortedData.begin() + n/2 + 1, sortedData.end());
}

double Q1 = median(lowerHalf);
double Q3 = median(upperHalf);
double IQR = Q3 - Q1;

double lowerBound = Q1 - 1.5 * IQR;
double upperBound = Q3 + 1.5 * IQR;

// Find whiskers
double lowerWhisker = sortedData[0];
double upperWhisker = sortedData[n-1];

for (double value : sortedData) {
    if (value >= lowerBound) {
        lowerWhisker = value;
        break;
```

```
        }
    }

    for (int i = n-1; i >= 0; i--) {
        if (sortedData[i] <= upperBound) {
            upperWhisker = sortedData[i];
            break;
        }
    }

    // Find outliers
    vector<double> outliers;
    for (double value : sortedData) {
        if (value < lowerBound || value > upperBound) {
            outliers.push_back(value);
        }
    }

    return {Q1, Q2, Q3, IQR, lowerWhisker, upperWhisker, outliers};
}

// Function to print simple summary (like the image)
void printSimpleSummary(const string& title, const FiveNumberSummary& summary) {
    cout << "\n" << title << " Summary:" << endl;
    cout << "Q1: " << fixed << setprecision(1) << summary.Q1 << endl;
    cout << "Median (Q2): " << summary.median << endl;
    cout << "Q3: " << summary.Q3 << endl;
    cout << "IQR: " << summary.IQR << endl;
```

```
    cout << "Lower Whisker: " << summary.lowerWhisker << endl;
    cout << "Upper Whisker: " << summary.upperWhisker << endl;
}

// Function to read CSV file
vector<DataEntry> readCSV(const string& filename) {
    vector<DataEntry> data;
    ifstream file(filename);

    if (!file.is_open()) {
        cout << "Error: Could not open file " << filename << endl;
        return data;
    }

    string line;
    // Skip header
    getline(file, line);

    while (getline(file, line)) {
        stringstream ss(line);
        string category, name, value, description;

        getline(ss, category, ',');
        getline(ss, name, ',');
        getline(ss, value, ',');
        getline(ss, description, ',');

        data.push_back({category, name, stod(value), description});
```

```
    }

    file.close();
    return data;
}

// Function to extract data by category
vector<double> extractDataByCategory(const vector<DataEntry>& data, const string& category) {
    vector<double> values;
    for (const auto& entry : data) {
        if (entry.category == category) {
            values.push_back(entry.value);
        }
    }
    return values;
}

int main() {
    cout << "Five-Number Summary Analysis" << endl;

    // Read data from CSV file
    vector<DataEntry> allData = readCSV("indian_data.csv");

    if (allData.empty()) {
        cout << "Error: No data found in CSV file." << endl;
        return 1;
    }
```

```
    // Extract data by categories
    vector<double> cricketAverages = extractDataByCategory(allData, "Cricket");
    vector<double> gdpPerCapita = extractDataByCategory(allData, "GDP");
    vector<double> literacyRates = extractDataByCategory(allData, "Literacy");

    // Calculate and display summaries
    FiveNumberSummary cricketSummary = fiveNumberSummaryWithOutliers(cricketAverages);
    FiveNumberSummary gdpSummary = fiveNumberSummaryWithOutliers(gdpPerCapita);
    FiveNumberSummary literacySummary = fiveNumberSummaryWithOutliers(literacyRates);

    printSimpleSummary("Cricket Batting Averages", cricketSummary);
    printSimpleSummary("GDP Per Capita", gdpSummary);
    printSimpleSummary("Literacy Rates", literacySummary);

    return 0;
}
```

Output:

```
$ ./dm4.exe
Five-Number Summary Analysis

Cricket Batting Averages Summary:
Q1: 22.1
Median (Q2): 28.3
Q3: 37.6
IQR: 15.6
Lower Whisker: 16.3
Upper Whisker: 52.4

GDP Per Capita Summary:
Q1: 5150.0
Median (Q2): 7100.0
Q3: 9500.0
IQR: 4350.0
Lower Whisker: 3500.0
Upper Whisker: 12500.0
```

Conclusion :

The five-number summary is a simple yet powerful statistical tool that provides a clear picture of data distribution. By calculating the minimum, Q1, median, Q3, and maximum, it becomes easier to understand the center, spread, and variability of a dataset. The inclusion of the interquartile range and whiskers allows us to identify outliers, which are important in detecting unusual or extreme values. This method is highly useful in data analysis, as it reduces complex datasets into meaningful insights and supports better decision-making.