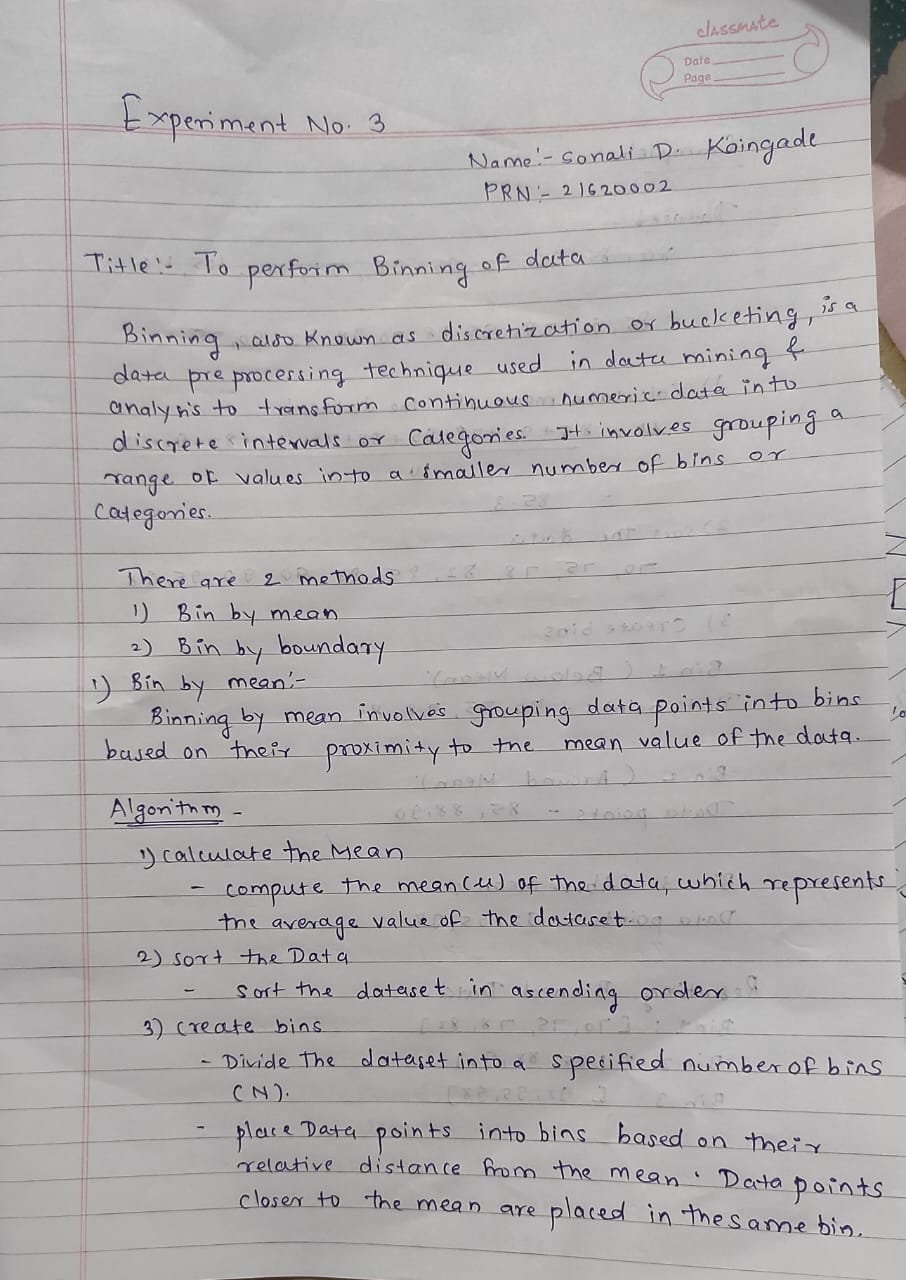
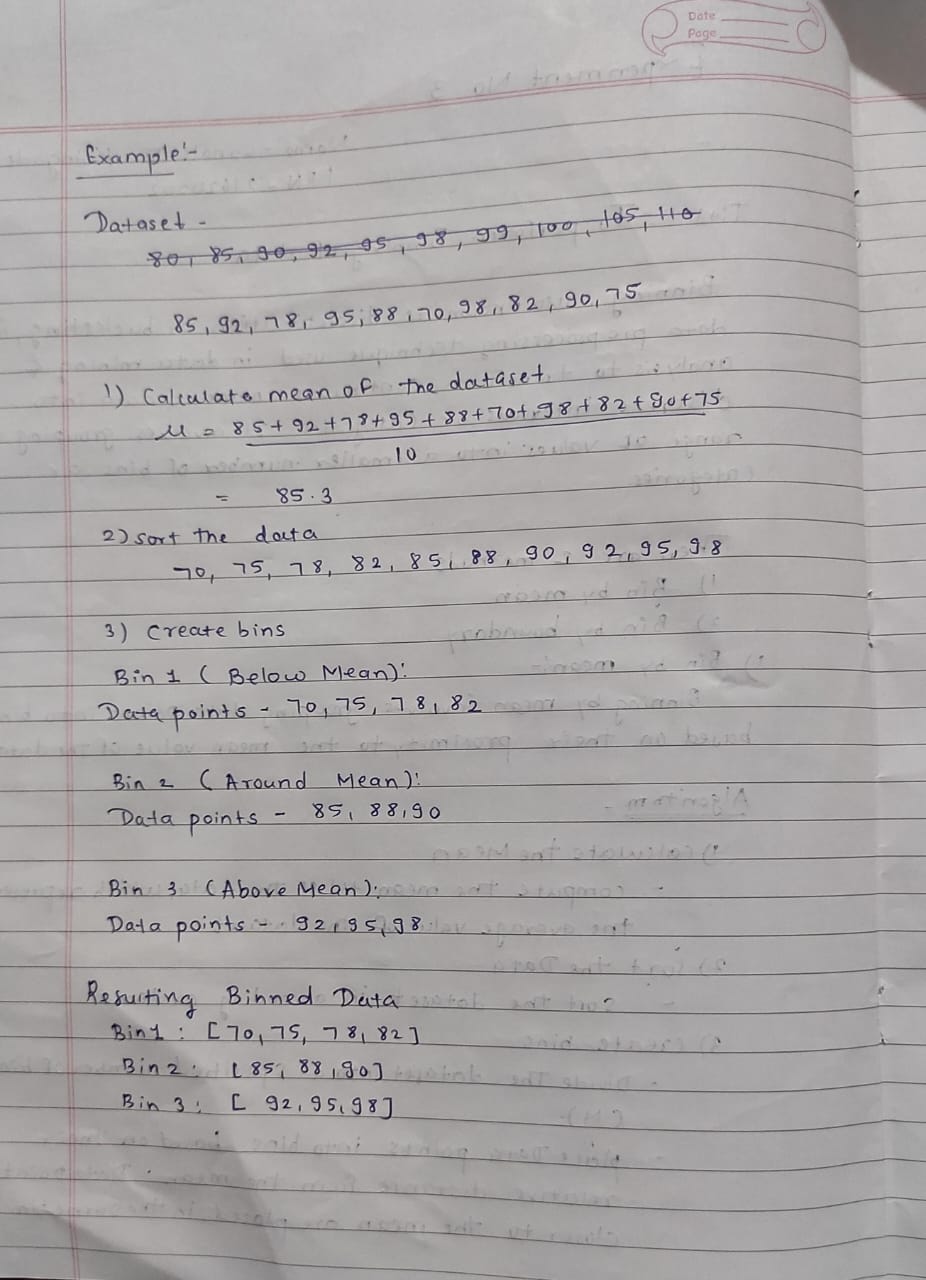
**Experiment no. 3**

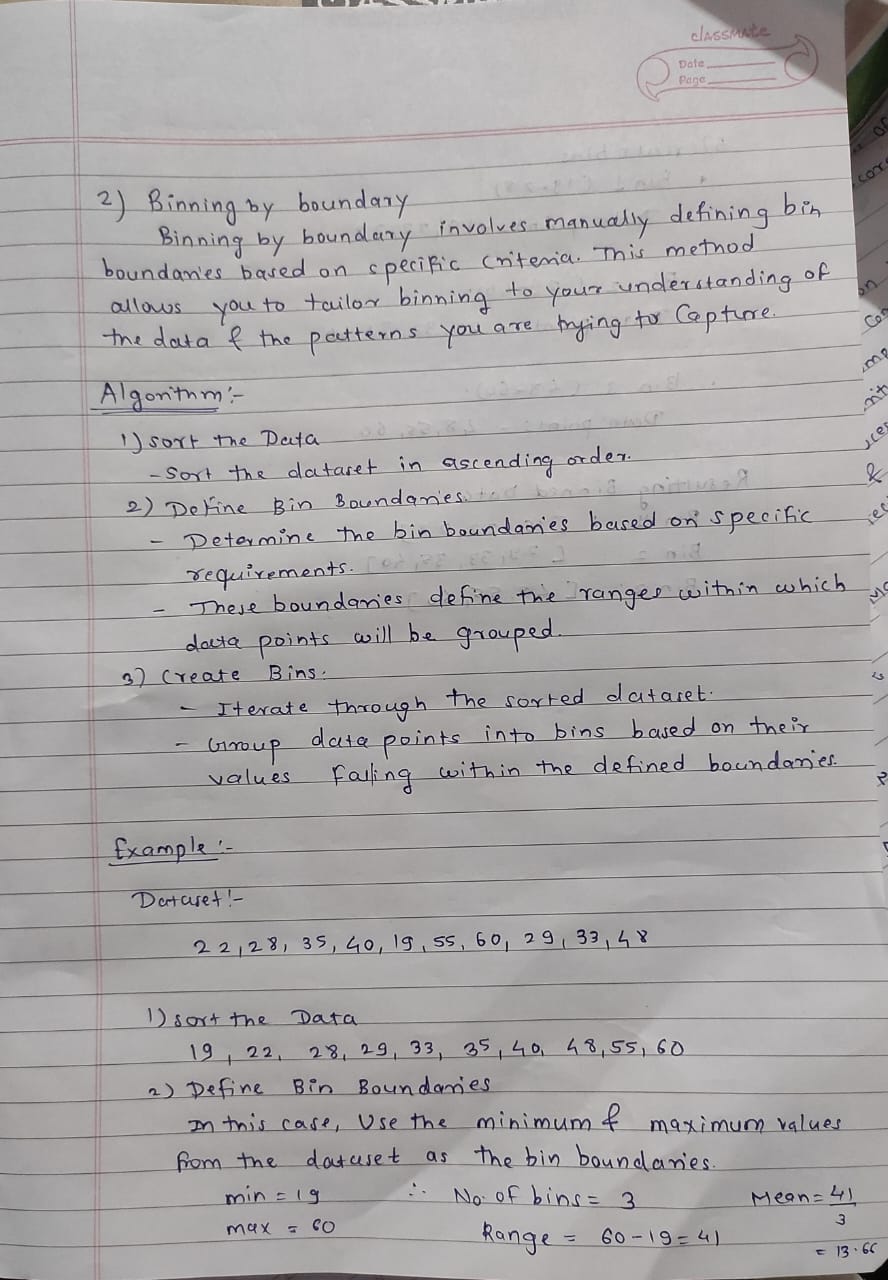
**Name: Sonali Dattatray Kaingade**

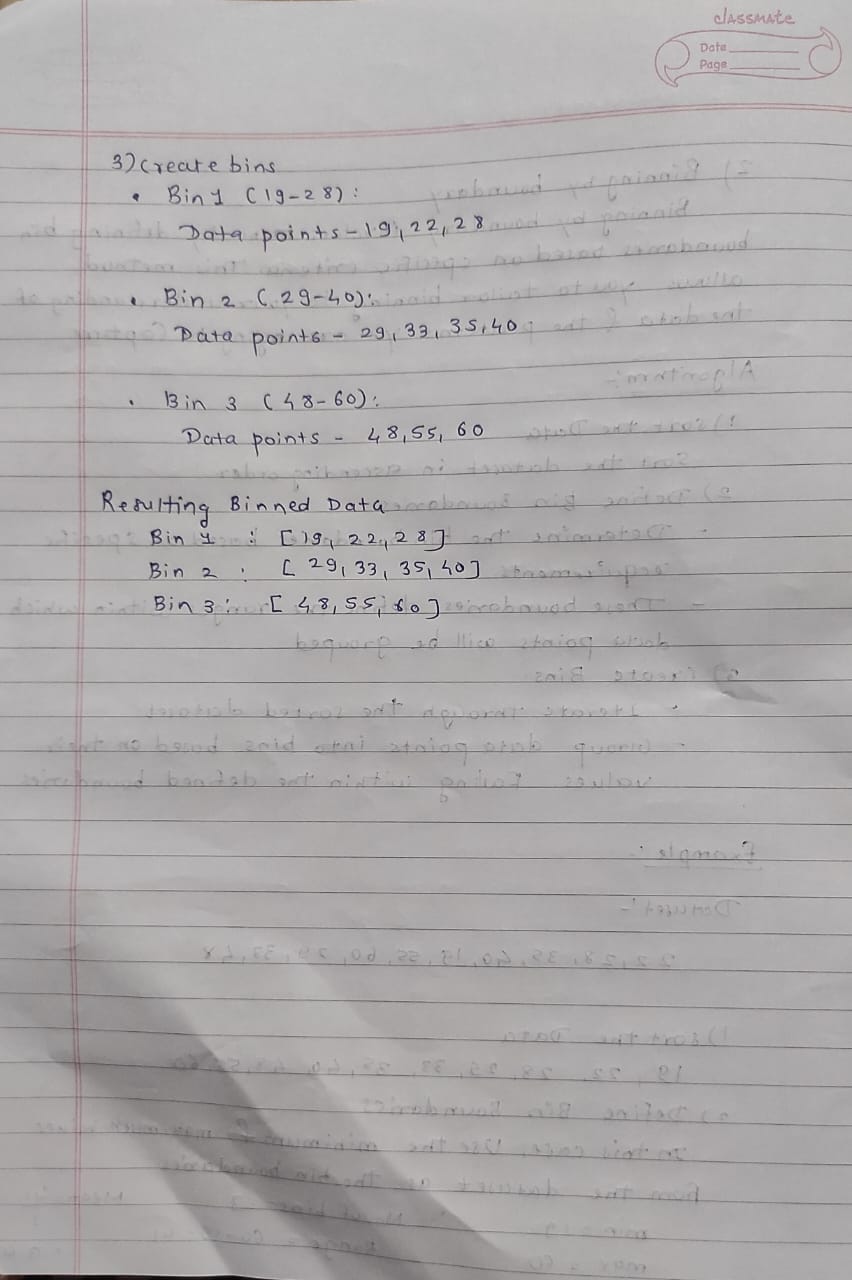
**PRN: 21620002**

**Title:** To perform Binning of data

**ti**

****

****

****

**Code:**

#include <iostream>

    #include <fstream>

    #include <vector>

    #include <algorithm>

    using namespace std;

    // Function for Bin by Mean method

    vector<int> binByMean(const vector<int>& data, int numBins) {

        int sum = 0;

        for (int x : data) {

            sum += x;

        }

        double mean = static\_cast<double>(sum) / data.size();

        vector<int> binBoundaries(numBins + 1);

        int binWidth = 0;

        // Calculate bin boundaries and bin width

        binWidth = (int)(2 \* (mean - data.front()) / numBins);

        for (int i = 0; i < numBins + 1; ++i) {

            binBoundaries[i] = data.front() + i \* binWidth;

        }

        vector<int> binAssignments(data.size());

        // Assign data points to bins based on mean-based boundaries

        for (size\_t i = 0; i < data.size(); ++i) {

            int x = data[i];

            int bin = numBins - 1; // Initialize to last bin

            // Find the appropriate bin for the current data point

            while (bin >= 0 && x < binBoundaries[bin]) {

                --bin;

            }

            binAssignments[i] = bin + 1; // Add 1 to match bin numbering (starting from 1)

        }

        return binAssignments;

    }

    // Function for Bin by Boundary method

    vector<int> binByBoundary(const vector<int>& data, int numBins) {

        int minVal = \*min\_element(data.begin(), data.end());

        int maxVal = \*max\_element(data.begin(), data.end());

        int range = maxVal - minVal;

        int binWidth = range / numBins;

        vector<int> binBoundaries(numBins + 1);

        // Calculate bin boundaries

        for (int i = 0; i < numBins + 1; ++i) {

            binBoundaries[i] = minVal + i \* binWidth;

        }

        vector<int> binAssignments(data.size());

        // Assign data points to bins based on bin boundaries

        for (size\_t i = 0; i < data.size(); ++i) {

            int x = data[i];

            int bin = 0;

            // Find the appropriate bin for the current data point

            while (bin < numBins && x >= binBoundaries[bin + 1]) {

                ++bin;

            }

            binAssignments[i] = bin + 1; // Add 1 to match bin numbering (starting from 1)

        }

        return binAssignments;

    }

    int main() {

        ifstream inputFile("input.txt");

        ofstream outputFile("output.txt");

        if (!inputFile.is\_open() || !outputFile.is\_open()) {

            cout << "Error opening files." << endl;

            return 1;

        }

        vector<int> data;

        int value;

        while (inputFile >> value) {

            data.push\_back(value);

        }

        int numBins = 3;

        // Bin by Mean

        vector<int> binByMeanResult = binByMean(data, numBins);

        outputFile << "Bin by Mean results:" << endl;

        for (size\_t i = 0; i < data.size(); ++i) {

            outputFile << data[i] << " -> Bin " << binByMeanResult[i] << endl;

        }

        outputFile << endl;

        // Bin by Boundary

        vector<int> binByBoundaryResult = binByBoundary(data, numBins);

        outputFile << "Bin by Boundary results:" << endl;

        for (size\_t i = 0; i < data.size(); ++i) {

            outputFile << data[i] << " -> Bin " << binByBoundaryResult[i] << endl;

        }

        inputFile.close();

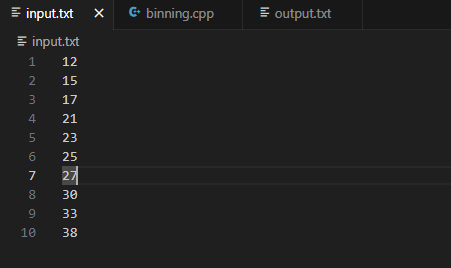
        outputFile.close();

        return 0;

    }

**Output:**

**Input.txt**



**Output.txt**

