Arun M

23122110

3MScDS B

Predicting Protein4 Expression in Breast Cancer using Linear Regression

Introduction

Understanding breast cancer development and progression is essential for improving treatments and outcomes. Protein4 is a key marker linked to breast cancer, and predicting its levels can help in better diagnosis and treatment planning. In this study, we use a machine learning technique called linear regression to predict the expression levels of Protein4 in breast cancer patients. Linear regression is a simple yet effective method for understanding how different factors influence Protein4 levels. We use data from patients, including the expression levels of Protein1, Protein2, and Protein3, as well as other features like age, gender, tumour stage, histology, ER status, PR status, HER2 status, surgery type, and patient status. By incorporating all these variables, we build a linear regression model that identifies the key factors affecting Protein4 levels. This model helps us uncover patterns and correlations that might not be obvious otherwise. Our goal is to create a tool that helps doctors make better decisions about patient care by providing deeper insights into the biology of breast cancer. This research demonstrates how machine learning can enhance the precision and personalization of cancer treatments, ultimately benefiting patient outcomes.

Dataset Description

a) **Protein1:** Expression levels of Protein1, measured through relevant biochemical assays. These values provide insight into one of the potential regulators or correlates of Protein4 expression.

b) **Protein2:** Expression levels of Protein2, another biomarker potentially interacting with Protein4 pathways.

c) **Protein3:** Expression levels of Protein3, adding to the network of proteins that might influence or indicate changes in Protein4 levels.

d) **Age:** The patient's age at the time of diagnosis, recorded in years. Age can be a significant factor in cancer progression and response to treatment.

e) **Gender:** The patient's gender. While breast cancer predominantly affects females, including gender ensures comprehensive data analysis for any male patients.

f) **Tumour_Stage:** The stage of the tumour at diagnosis, categorized according to standard staging criteria (e.g., Stage I, II, III, IV). Tumour stage is crucial for understanding the extent of cancer spread and its aggressiveness.

g) **Histology:** The microscopic structure of the tumour tissue, classified into types such as ductal, lobular, and others. Histology provides detailed information about tumour characteristics and potential behaviour.

h) **ER Status (Estrogen Receptor Status):** Indicates whether the cancer cells have receptors for the hormone estrogen (positive or negative). ER status is essential for determining hormone therapy suitability.

i) **PR Status (Progesterone Receptor Status):** Indicates whether the cancer cells have receptors for the hormone progesterone (positive or negative). Like ER status, PR status is vital for hormone therapy decisions.

j) **HER2 Status (Human Epidermal Growth Factor Receptor 2 Status):** Indicates overexpression or amplification of the HER2 gene in cancer cells (positive or negative). HER2 status guides the use of targeted therapies such as trastuzumab.

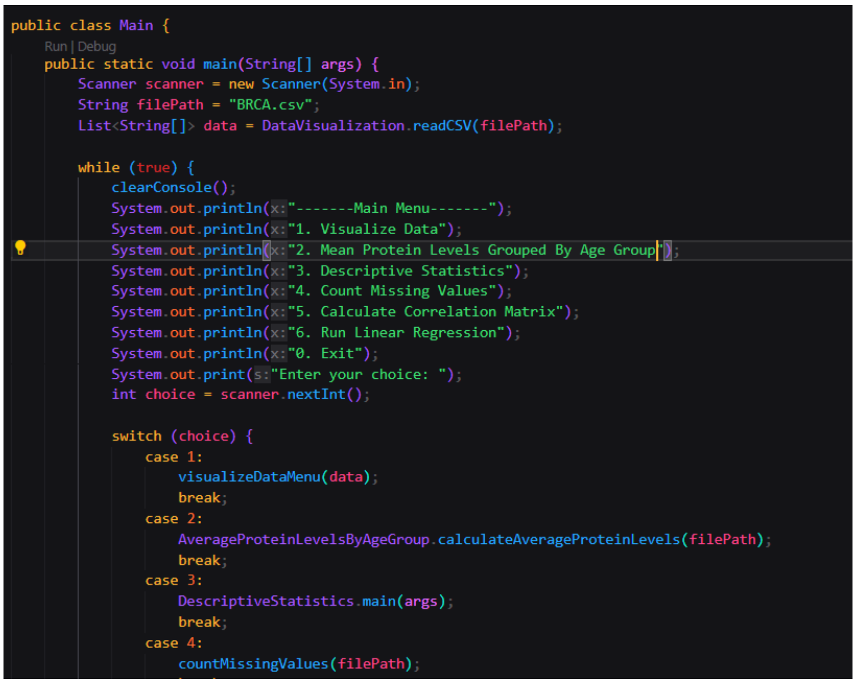
k) **Surgery_Type:** Type of surgery performed on the patient, including options such as mastectomy (removal of the whole breast), lumpectomy (removal of the tumor and some surrounding tissue), and other surgical interventions.

l) **Patient_Status:** The current health status of the patient, categorized as alive, deceased, or disease-free. This outcome measure is essential for survival analysis and treatment effectiveness studies.

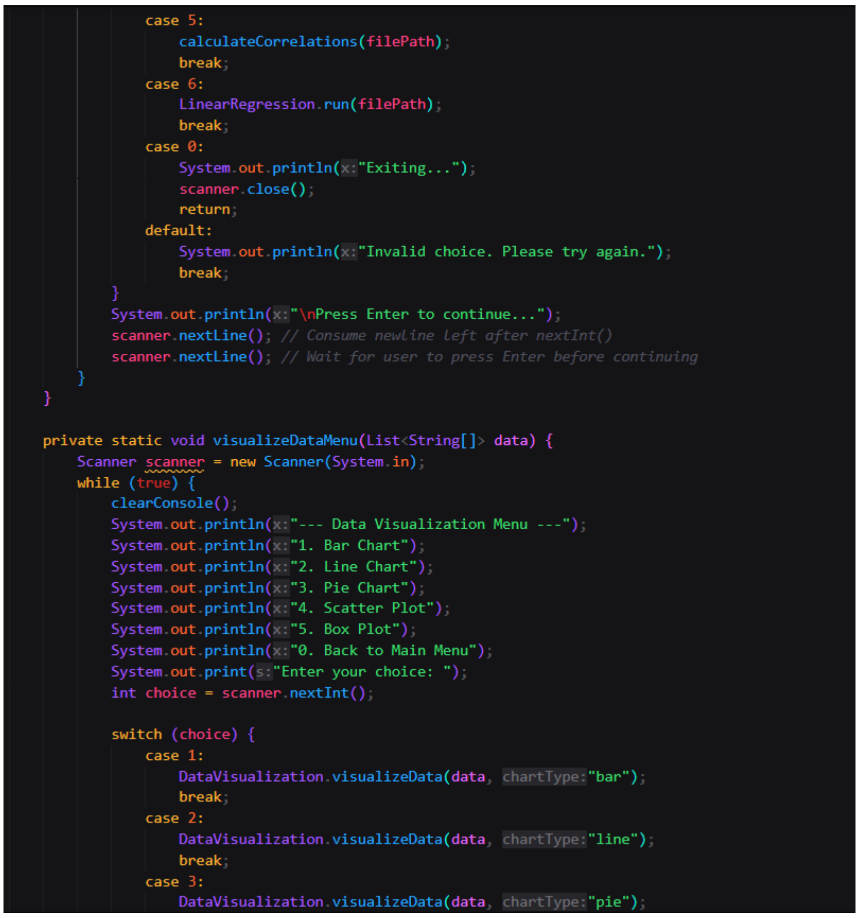
Code Implementation

The code implementation involves developing a predictive model in Main.java using Java. The program utilizes various packages, including java.util for data structures, java.io for file handling, and custom machine learning algorithms for linear regression. This setup ensures efficient data processing and model training. The Main.java file handles data loading, preprocessing, model training, and evaluation. Key functions include reading the dataset, normalizing features, and splitting data into training and testing sets. The custom linear regression algorithm is implemented to predict Protein4 levels accurately. The implementation is modular, making it easy to update and maintain.

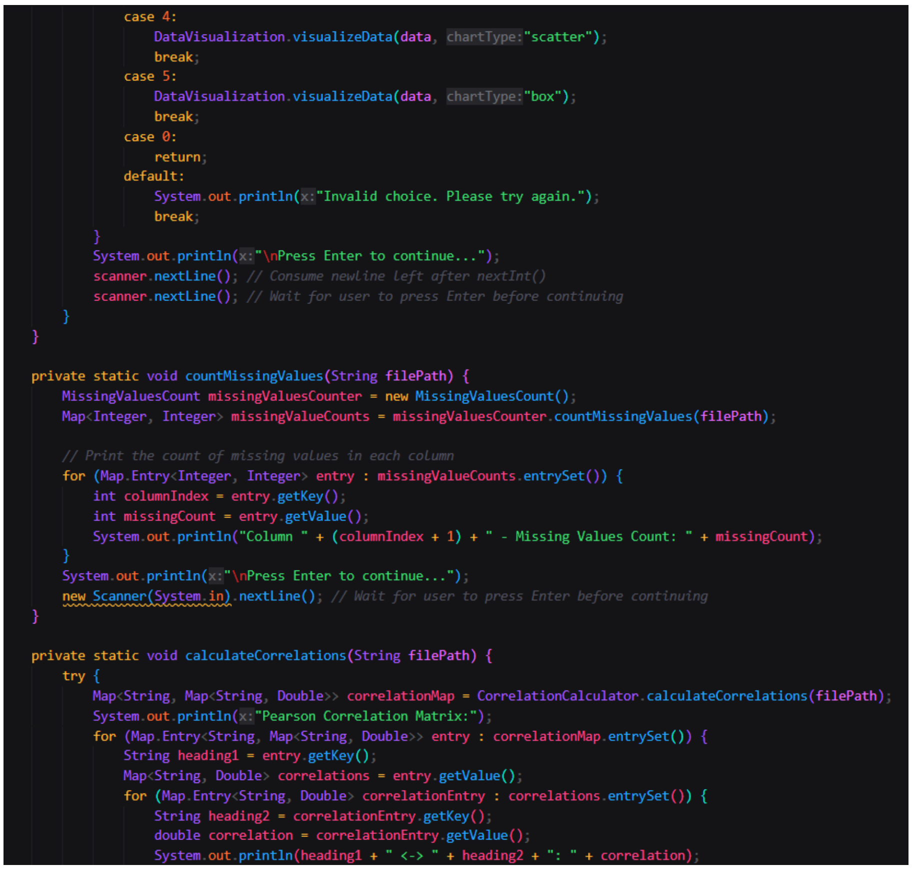
I. Main



```
public class Main {
    Run | Debug
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String filePath = "BRCA.csv";
        List<String[]> data = DataVisualization.readCSV(filePath);

        while (true) {
            clearConsole();
            System.out.println(x:"-------Main Menu-------");
            System.out.println(x:"1. Visualize Data");
            System.out.println(x:"2. Mean Protein Levels Grouped By Age Group");
            System.out.println(x:"3. Descriptive Statistics");
            System.out.println(x:"4. Count Missing Values");
            System.out.println(x:"5. Calculate Correlation Matrix");
            System.out.println(x:"6. Run Linear Regression");
            System.out.println(x:"0. Exit");
            System.out.print(s:"Enter your choice: ");
            int choice = scanner.nextInt();

            switch (choice) {
                case 1:
                    visualizeDataMenu(data);
                    break;
                case 2:
                    AverageProteinLevelsByAgeGroup.calculateAverageProteinLevels(filePath);
                    break;
                case 3:
                    DescriptiveStatistics.main(args);
                    break;
                case 4:
                    countMissingValues(filePath);
```

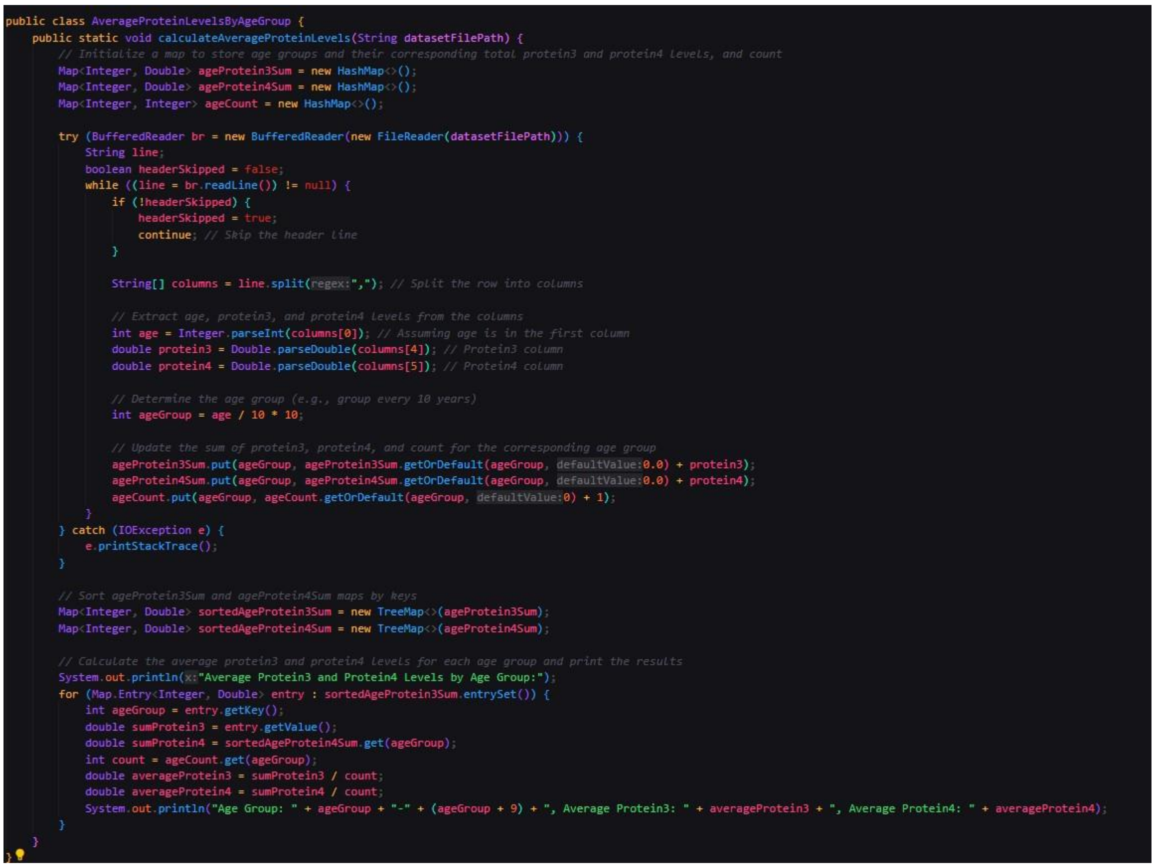


```
                case 5:
                    calculateCorrelations(filePath);
                    break;
                case 6:
                    LinearRegression.run(filePath);
                    break;
                case 0:
                    System.out.println(x:"Exiting...");
                    scanner.close();
                    return;
                default:
                    System.out.println(x:"Invalid choice. Please try again.");
                    break;
            }
            System.out.println(x:"\nPress Enter to continue...");
            scanner.nextLine(); // Consume newline left after nextInt()
            scanner.nextLine(); // Wait for user to press Enter before continuing
        }
    }

    private static void visualizeDataMenu(List<String[]> data) {
        Scanner scanner = new Scanner(System.in);
        while (true) {
            clearConsole();
            System.out.println(x:"--- Data Visualization Menu ---");
            System.out.println(x:"1. Bar Chart");
            System.out.println(x:"2. Line Chart");
            System.out.println(x:"3. Pie Chart");
            System.out.println(x:"4. Scatter Plot");
            System.out.println(x:"5. Box Plot");
            System.out.println(x:"0. Back to Main Menu");
            System.out.print(s:"Enter your choice: ");
            int choice = scanner.nextInt();

            switch (choice) {
                case 1:
                    DataVisualization.visualizeData(data, chartType:"bar");
                    break;
                case 2:
                    DataVisualization.visualizeData(data, chartType:"line");
                    break;
                case 3:
                    DataVisualization.visualizeData(data, chartType:"pie");
```



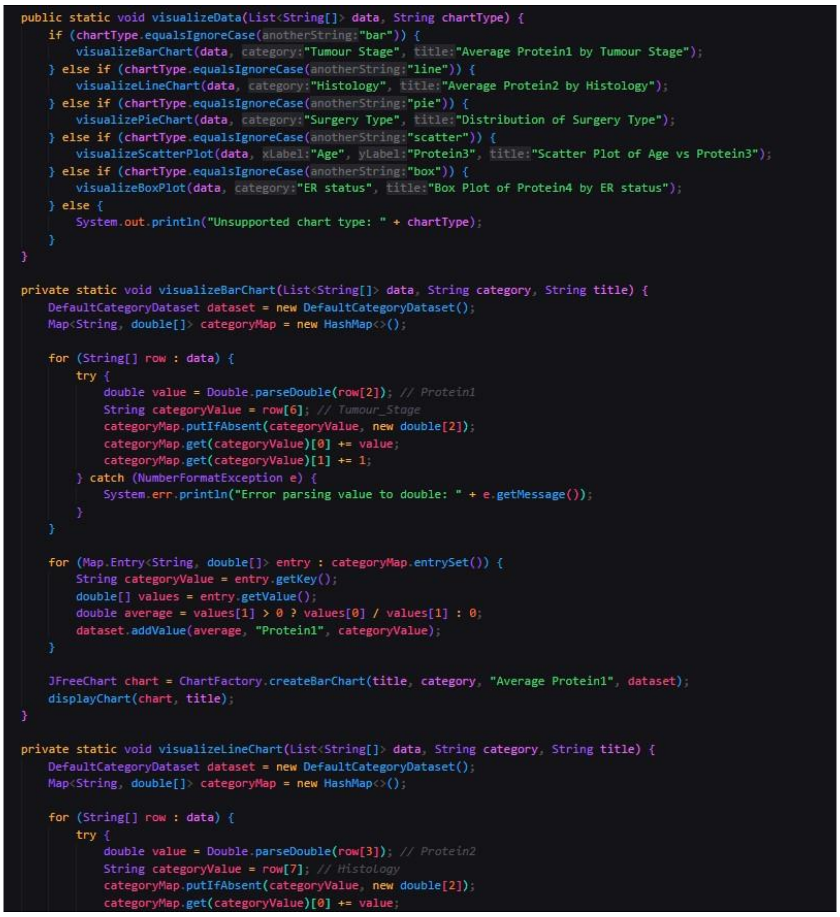
```java
                case 4:
                    DataVisualization.visualizeData(data, chartType:"scatter");
                    break;
                case 5:
                    DataVisualization.visualizeData(data, chartType:"box");
                    break;
                case 0:
                    return;
                default:
                    System.out.println(x:"Invalid choice. Please try again.");
                    break;
            }
            System.out.println(x:"\nPress Enter to continue...");
            scanner.nextLine(); // Consume newline left after nextInt()
            scanner.nextLine(); // Wait for user to press Enter before continuing
        }
    }

    private static void countMissingValues(String filePath) {
        MissingValuesCount missingValuesCounter = new MissingValuesCount();
        Map<Integer, Integer> missingValueCounts = missingValuesCounter.countMissingValues(filePath);

        // Print the count of missing values in each column
        for (Map.Entry<Integer, Integer> entry : missingValueCounts.entrySet()) {
            int columnIndex = entry.getKey();
            int missingCount = entry.getValue();
            System.out.println("Column " + (columnIndex + 1) + " - Missing Values Count: " + missingCount);
        }
        System.out.println(x:"\nPress Enter to continue...");
        new Scanner(System.in).nextLine(); // Wait for user to press Enter before continuing
    }

    private static void calculateCorrelations(String filePath) {
        try {
            Map<String, Map<String, Double>> correlationMap = CorrelationCalculator.calculateCorrelations(filePath);
            System.out.println(x:"Pearson Correlation Matrix:");
            for (Map.Entry<String, Map<String, Double>> entry : correlationMap.entrySet()) {
                String heading1 = entry.getKey();
                Map<String, Double> correlations = entry.getValue();
                for (Map.Entry<String, Double> correlationEntry : correlations.entrySet()) {
                    String heading2 = correlationEntry.getKey();
                    double correlation = correlationEntry.getValue();
                    System.out.println(heading1 + " <-> " + heading2 + ": " + correlation);
```

II. Average Protein Levels Grouped by Age

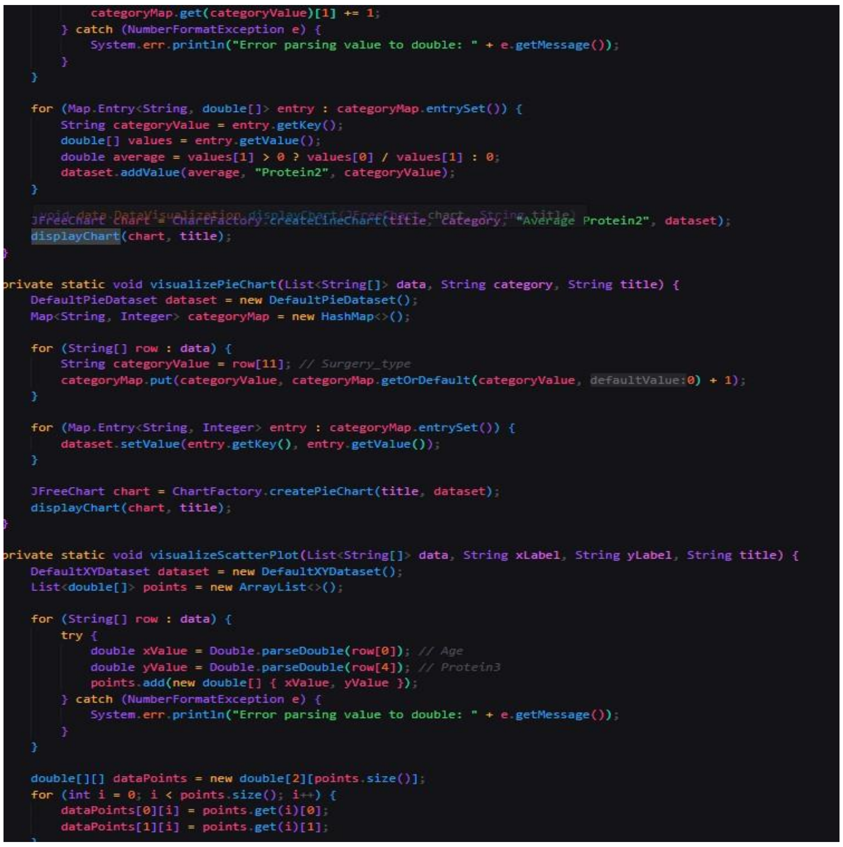


```java
public class AverageProteinLevelsByAgeGroup {
    public static void calculateAverageProteinLevels(String datasetFilePath) {
        // Initialize a map to store age groups and their corresponding total protein3 and protein4 levels, and count
        Map<Integer, Double> ageProtein3Sum = new HashMap<>();
        Map<Integer, Double> ageProtein4Sum = new HashMap<>();
        Map<Integer, Integer> ageCount = new HashMap<>();

        try (BufferedReader br = new BufferedReader(new FileReader(datasetFilePath))) {
            String line;
            boolean headerSkipped = false;
            while ((line = br.readLine()) != null) {
                if (!headerSkipped) {
                    headerSkipped = true;
                    continue; // Skip the header line
                }

                String[] columns = line.split(regex:","); // Split the row into columns

                // Extract age, protein3, and protein4 levels from the columns
                int age = Integer.parseInt(columns[0]); // Assuming age is in the first column
                double protein3 = Double.parseDouble(columns[4]); // Protein3 column
                double protein4 = Double.parseDouble(columns[5]); // Protein4 column

                // Determine the age group (e.g., group every 10 years)
                int ageGroup = age / 10 * 10;

                // Update the sum of protein3, protein4, and count for the corresponding age group
                ageProtein3Sum.put(ageGroup, ageProtein3Sum.getOrDefault(ageGroup, defaultValue:0.0) + protein3);
                ageProtein4Sum.put(ageGroup, ageProtein4Sum.getOrDefault(ageGroup, defaultValue:0.0) + protein4);
                ageCount.put(ageGroup, ageCount.getOrDefault(ageGroup, defaultValue:0) + 1);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }

        // Sort ageProtein3Sum and ageProtein4Sum maps by keys
        Map<Integer, Double> sortedAgeProtein3Sum = new TreeMap<>(ageProtein3Sum);
        Map<Integer, Double> sortedAgeProtein4Sum = new TreeMap<>(ageProtein4Sum);

        // Calculate the average protein3 and protein4 levels for each age group and print the results
        System.out.println(x:"Average Protein3 and Protein4 Levels by Age Group:");
        for (Map.Entry<Integer, Double> entry : sortedAgeProtein3Sum.entrySet()) {
            int ageGroup = entry.getKey();
            double sumProtein3 = entry.getValue();
            double sumProtein4 = sortedAgeProtein4Sum.get(ageGroup);
            int count = ageCount.get(ageGroup);
            double averageProtein3 = sumProtein3 / count;
            double averageProtein4 = sumProtein4 / count;
            System.out.println("Age Group: " + ageGroup + "-" + (ageGroup + 9) + ", Average Protein3: " + averageProtein3 + ", Average Protein4: " + averageProtein4);
        }
    }
}
```

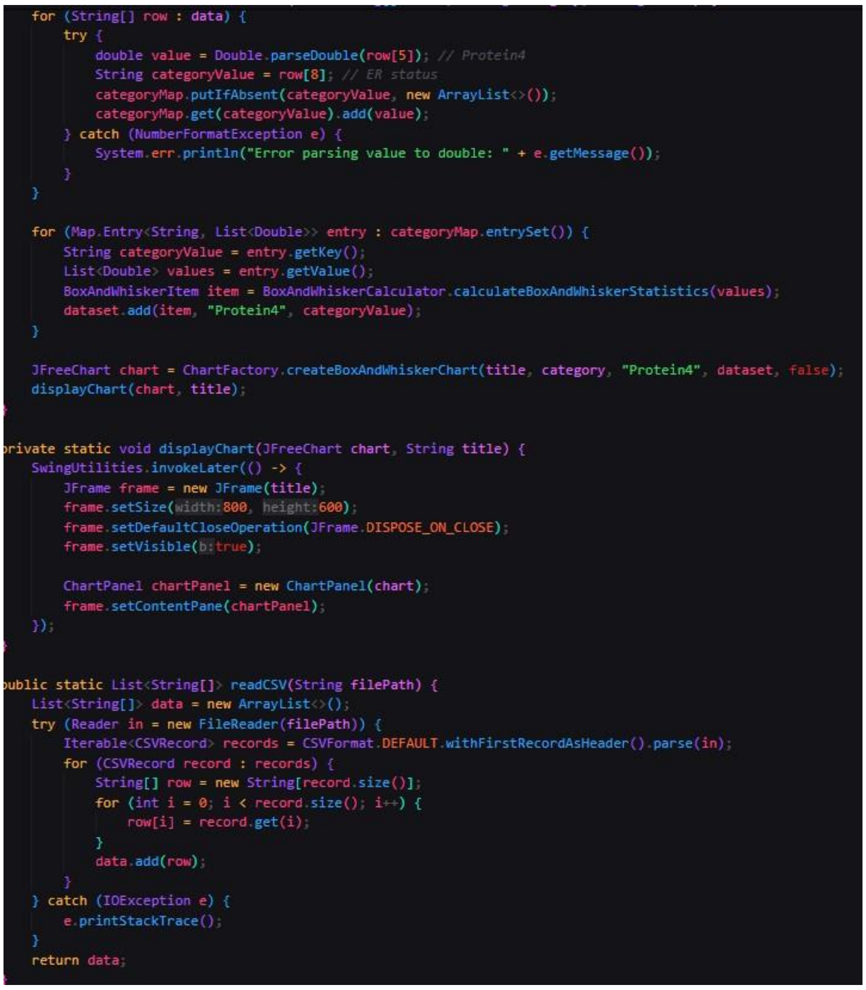
III. Data Visualization



```java
public static void visualizeData(List<String[]> data, String chartType) {
    if (chartType.equalsIgnoreCase(anotherString:"bar")) {
        visualizeBarChart(data, category:"Tumour Stage", title:"Average Protein1 by Tumour Stage");
    } else if (chartType.equalsIgnoreCase(anotherString:"line")) {
        visualizeLineChart(data, category:"Histology", title:"Average Protein2 by Histology");
    } else if (chartType.equalsIgnoreCase(anotherString:"pie")) {
        visualizePieChart(data, category:"Surgery Type", title:"Distribution of Surgery Type");
    } else if (chartType.equalsIgnoreCase(anotherString:"scatter")) {
        visualizeScatterPlot(data, xLabel:"Age", yLabel:"Protein3", title:"Scatter Plot of Age vs Protein3");
    } else if (chartType.equalsIgnoreCase(anotherString:"box")) {
        visualizeBoxPlot(data, category:"ER status", title:"Box Plot of Protein4 by ER status");
    } else {
        System.out.println("Unsupported chart type: " + chartType);
    }
}

private static void visualizeBarChart(List<String[]> data, String category, String title) {
    DefaultCategoryDataset dataset = new DefaultCategoryDataset();
    Map<String, double[]> categoryMap = new HashMap<>();

    for (String[] row : data) {
        try {
            double value = Double.parseDouble(row[2]); // Protein1
            String categoryValue = row[6]; // Tumour_Stage
            categoryMap.putIfAbsent(categoryValue, new double[2]);
            categoryMap.get(categoryValue)[0] += value;
            categoryMap.get(categoryValue)[1] += 1;
        } catch (NumberFormatException e) {
            System.err.println("Error parsing value to double: " + e.getMessage());
        }
    }

    for (Map.Entry<String, double[]> entry : categoryMap.entrySet()) {
        String categoryValue = entry.getKey();
        double[] values = entry.getValue();
        double average = values[1] > 0 ? values[0] / values[1] : 0;
        dataset.addValue(average, "Protein1", categoryValue);
    }

    JFreeChart chart = ChartFactory.createBarChart(title, category, "Average Protein1", dataset);
    displayChart(chart, title);
}

private static void visualizeLineChart(List<String[]> data, String category, String title) {
    DefaultCategoryDataset dataset = new DefaultCategoryDataset();
    Map<String, double[]> categoryMap = new HashMap<>();

    for (String[] row : data) {
        try {
            double value = Double.parseDouble(row[3]); // Protein2
            String categoryValue = row[7]; // Histology
            categoryMap.putIfAbsent(categoryValue, new double[2]);
            categoryMap.get(categoryValue)[0] += value;
```

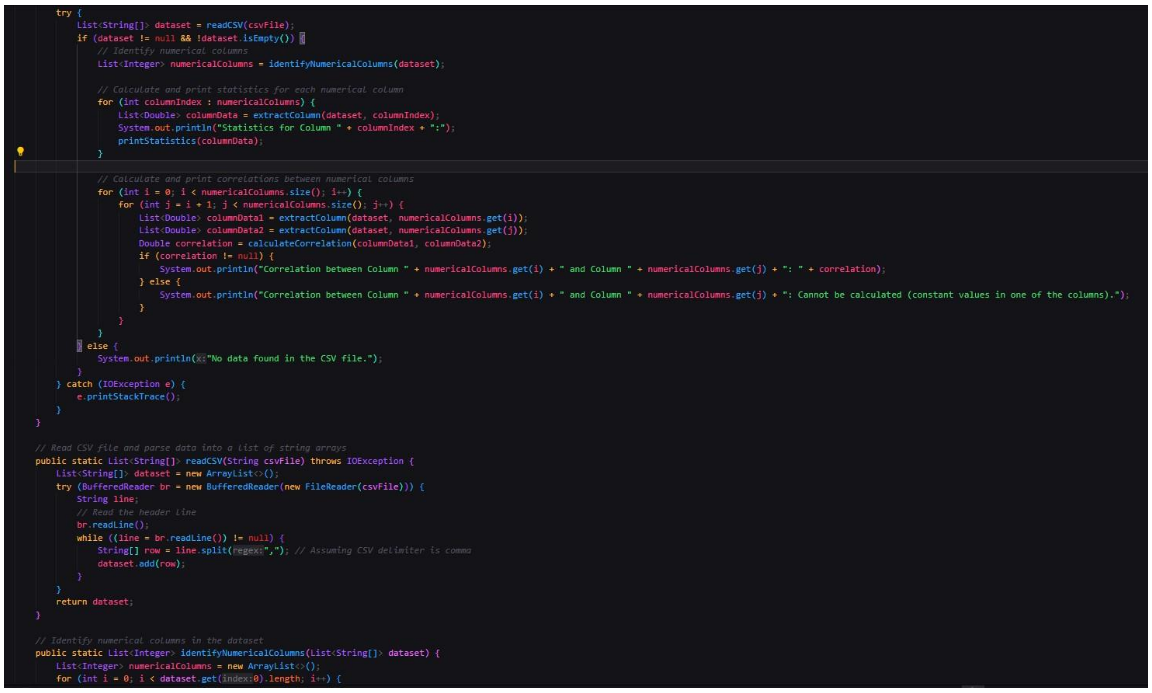


```java
            categoryMap.get(categoryValue)[1] += 1;
        } catch (NumberFormatException e) {
            System.err.println("Error parsing value to double: " + e.getMessage());
        }
    }

    for (Map.Entry<String, double[]> entry : categoryMap.entrySet()) {
        String categoryValue = entry.getKey();
        double[] values = entry.getValue();
        double average = values[1] > 0 ? values[0] / values[1] : 0;
        dataset.addValue(average, "Protein2", categoryValue);
    }

    void data.DataVisualization.displayChart(JFreeChart chart, String title)
    JFreeChart chart = ChartFactory.createLineChart(title, category, "Average Protein2", dataset);
    displayChart(chart, title);
}

rivate static void visualizePieChart(List<String[]> data, String category, String title) {
    DefaultPieDataset dataset = new DefaultPieDataset();
    Map<String, Integer> categoryMap = new HashMap<>();

    for (String[] row : data) {
        String categoryValue = row[11]; // Surgery_type
        categoryMap.put(categoryValue, categoryMap.getOrDefault(categoryValue, defaultValue:0) + 1);
    }

    for (Map.Entry<String, Integer> entry : categoryMap.entrySet()) {
        dataset.setValue(entry.getKey(), entry.getValue());
    }

    JFreeChart chart = ChartFactory.createPieChart(title, dataset);
    displayChart(chart, title);
}

rivate static void visualizeScatterPlot(List<String[]> data, String xLabel, String yLabel, String title) {
    DefaultXYDataset dataset = new DefaultXYDataset();
    List<double[]> points = new ArrayList<>();

    for (String[] row : data) {
        try {
            double xValue = Double.parseDouble(row[0]); // Age
            double yValue = Double.parseDouble(row[4]); // Protein3
            points.add(new double[] { xValue, yValue });
        } catch (NumberFormatException e) {
            System.err.println("Error parsing value to double: " + e.getMessage());
        }
    }

    double[][] dataPoints = new double[2][points.size()];
    for (int i = 0; i < points.size(); i++) {
        dataPoints[0][i] = points.get(i)[0];
        dataPoints[1][i] = points.get(i)[1];
```

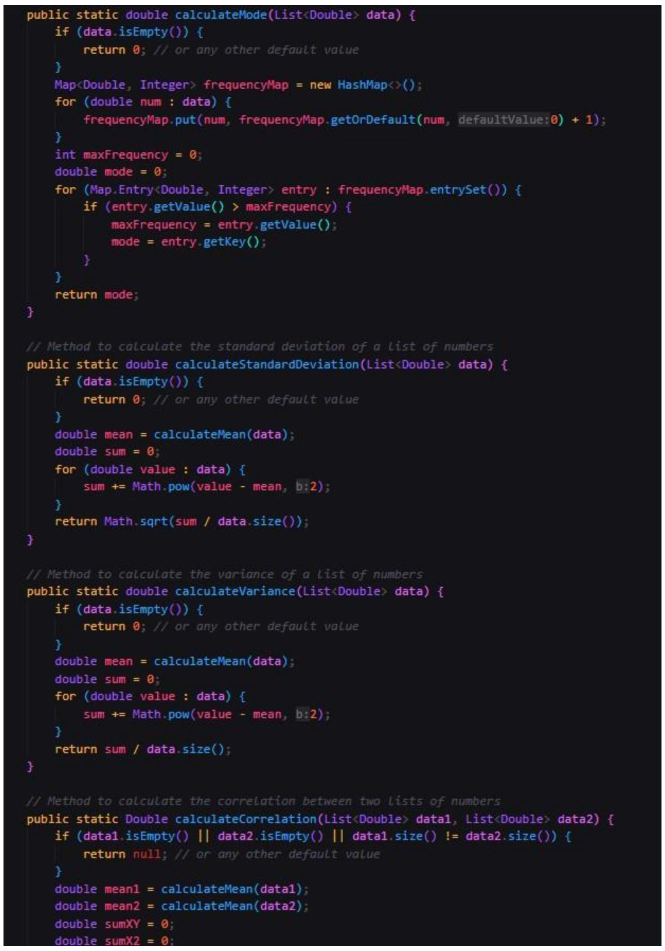


```
    for (String[] row : data) {
        try {
            double value = Double.parseDouble(row[5]); // Protein4
            String categoryValue = row[8]; // ER status
            categoryMap.putIfAbsent(categoryValue, new ArrayList<>());
            categoryMap.get(categoryValue).add(value);
        } catch (NumberFormatException e) {
            System.err.println("Error parsing value to double: " + e.getMessage());
        }
    }

    for (Map.Entry<String, List<Double>> entry : categoryMap.entrySet()) {
        String categoryValue = entry.getKey();
        List<Double> values = entry.getValue();
        BoxAndWhiskerItem item = BoxAndWhiskerCalculator.calculateBoxAndWhiskerStatistics(values);
        dataset.add(item, "Protein4", categoryValue);
    }

    JFreeChart chart = ChartFactory.createBoxAndWhiskerChart(title, category, "Protein4", dataset, false);
    displayChart(chart, title);
}

rivate static void displayChart(JFreeChart chart, String title) {
    SwingUtilities.invokeLater(() -> {
        JFrame frame = new JFrame(title);
        frame.setSize(width:800, height:600);
        frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        frame.setVisible(b:true);

        ChartPanel chartPanel = new ChartPanel(chart);
        frame.setContentPane(chartPanel);
    });
}

ublic static List<String[]> readCSV(String filePath) {
    List<String[]> data = new ArrayList<>();
    try (Reader in = new FileReader(filePath)) {
        Iterable<CSVRecord> records = CSVFormat.DEFAULT.withFirstRecordAsHeader().parse(in);
        for (CSVRecord record : records) {
            String[] row = new String[record.size()];
            for (int i = 0; i < record.size(); i++) {
                row[i] = record.get(i);
            }
            data.add(row);
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
    return data;
}
```

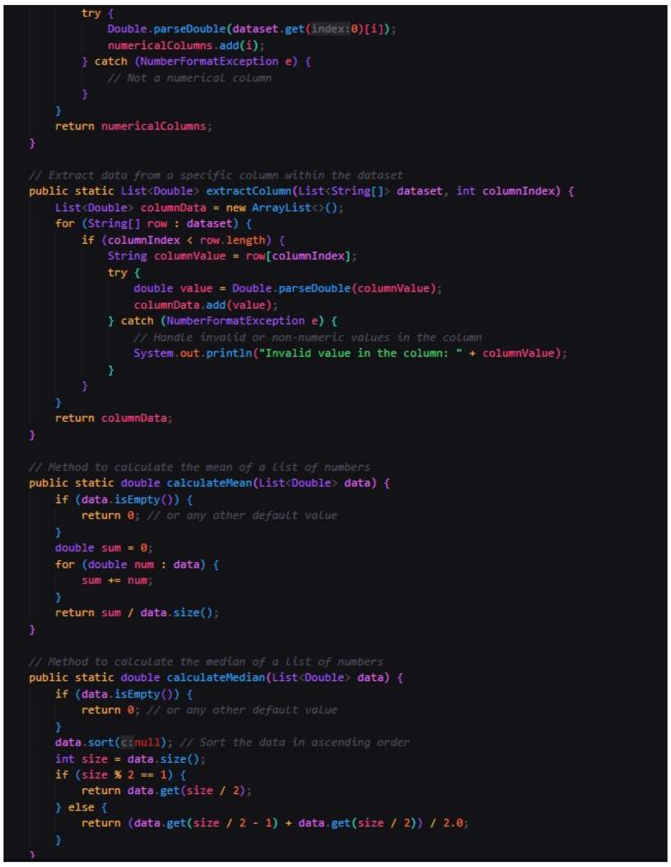
IV. Descriptive Statistics



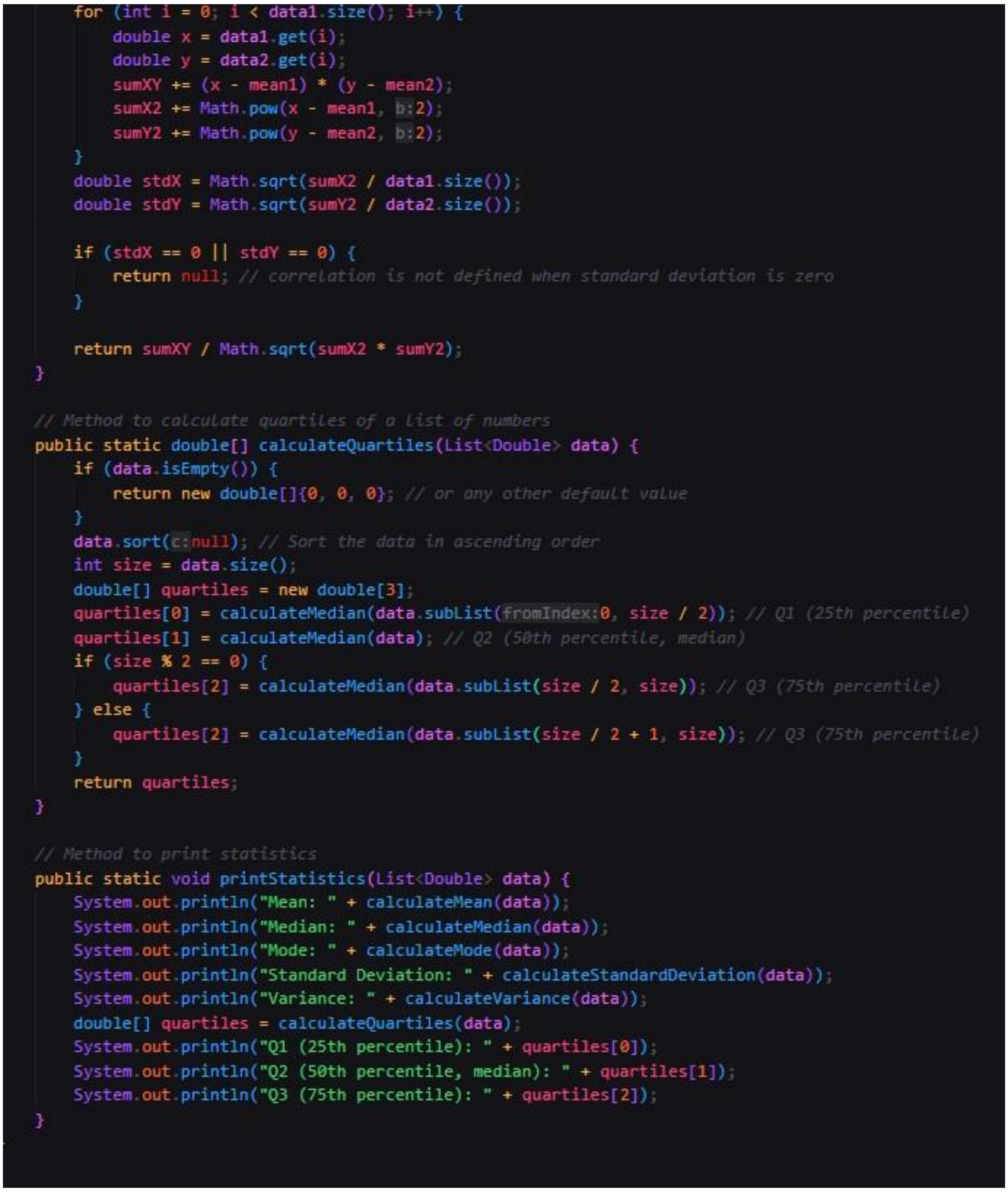
```
        try {
            List<String[]> dataset = readCSV(csvFile);
            if (dataset != null && !dataset.isEmpty()) {
                // Identify numerical columns
                List<Integer> numericalColumns = identifyNumericalColumns(dataset);

                // Calculate and print statistics for each numerical column
                for (int columnIndex : numericalColumns) {
                    List<Double> columnData = extractColumn(dataset, columnIndex);
                    System.out.println("Statistics for Column " + columnIndex + ":");
                    printStatistics(columnData);
                }

                // Calculate and print correlations between numerical columns
                for (int i = 0; i < numericalColumns.size(); i++) {
                    for (int j = i + 1; j < numericalColumns.size(); j++) {
                        List<Double> columnData1 = extractColumn(dataset, numericalColumns.get(i));
                        List<Double> columnData2 = extractColumn(dataset, numericalColumns.get(j));
                        Double correlation = calculateCorrelation(columnData1, columnData2);
                        if (correlation != null) {
                            System.out.println("Correlation between Column " + numericalColumns.get(i) + " and Column " + numericalColumns.get(j) + ": " + correlation);
                        } else {
                            System.out.println("Correlation between Column " + numericalColumns.get(i) + " and Column " + numericalColumns.get(j) + ": Cannot be calculated (constant values in one of the columns).");
                        }
                    }
                }
            } else {
                System.out.println(x:"No data found in the CSV file.");
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    // Read CSV file and parse data into a list of string arrays
    public static List<String[]> readCSV(String csvFile) throws IOException {
        List<String[]> dataset = new ArrayList<>();
        try (BufferedReader br = new BufferedReader(new FileReader(csvFile))) {
            String line;
            // Read the header line
            br.readLine();
            while ((line = br.readLine()) != null) {
                String[] row = line.split(regex:","); // Assuming CSV delimiter is comma
                dataset.add(row);
            }
        }
        return dataset;
    }

    // Identify numerical columns in the dataset
    public static List<Integer> identifyNumericalColumns(List<String[]> dataset) {
        List<Integer> numericalColumns = new ArrayList<>();
        for (int i = 0; i < dataset.get(index:0).length; i++) {
```



```java
public static double calculateMode(List<Double> data) {
    if (data.isEmpty()) {
        return 0; // or any other default value
    }
    Map<Double, Integer> frequencyMap = new HashMap<>();
    for (double num : data) {
        frequencyMap.put(num, frequencyMap.getOrDefault(num, defaultValue:0) + 1);
    }
    int maxFrequency = 0;
    double mode = 0;
    for (Map.Entry<Double, Integer> entry : frequencyMap.entrySet()) {
        if (entry.getValue() > maxFrequency) {
            maxFrequency = entry.getValue();
            mode = entry.getKey();
        }
    }
    return mode;
}

// Method to calculate the standard deviation of a list of numbers
public static double calculateStandardDeviation(List<Double> data) {
    if (data.isEmpty()) {
        return 0; // or any other default value
    }
    double mean = calculateMean(data);
    double sum = 0;
    for (double value : data) {
        sum += Math.pow(value - mean, b:2);
    }
    return Math.sqrt(sum / data.size());
}

// Method to calculate the variance of a list of numbers
public static double calculateVariance(List<Double> data) {
    if (data.isEmpty()) {
        return 0; // or any other default value
    }
    double mean = calculateMean(data);
    double sum = 0;
    for (double value : data) {
        sum += Math.pow(value - mean, b:2);
    }
    return sum / data.size();
}

// Method to calculate the correlation between two lists of numbers
public static Double calculateCorrelation(List<Double> data1, List<Double> data2) {
    if (data1.isEmpty() || data2.isEmpty() || data1.size() != data2.size()) {
        return null; // or any other default value
    }
    double mean1 = calculateMean(data1);
    double mean2 = calculateMean(data2);
    double sumXY = 0;
    double sumX2 = 0;
```

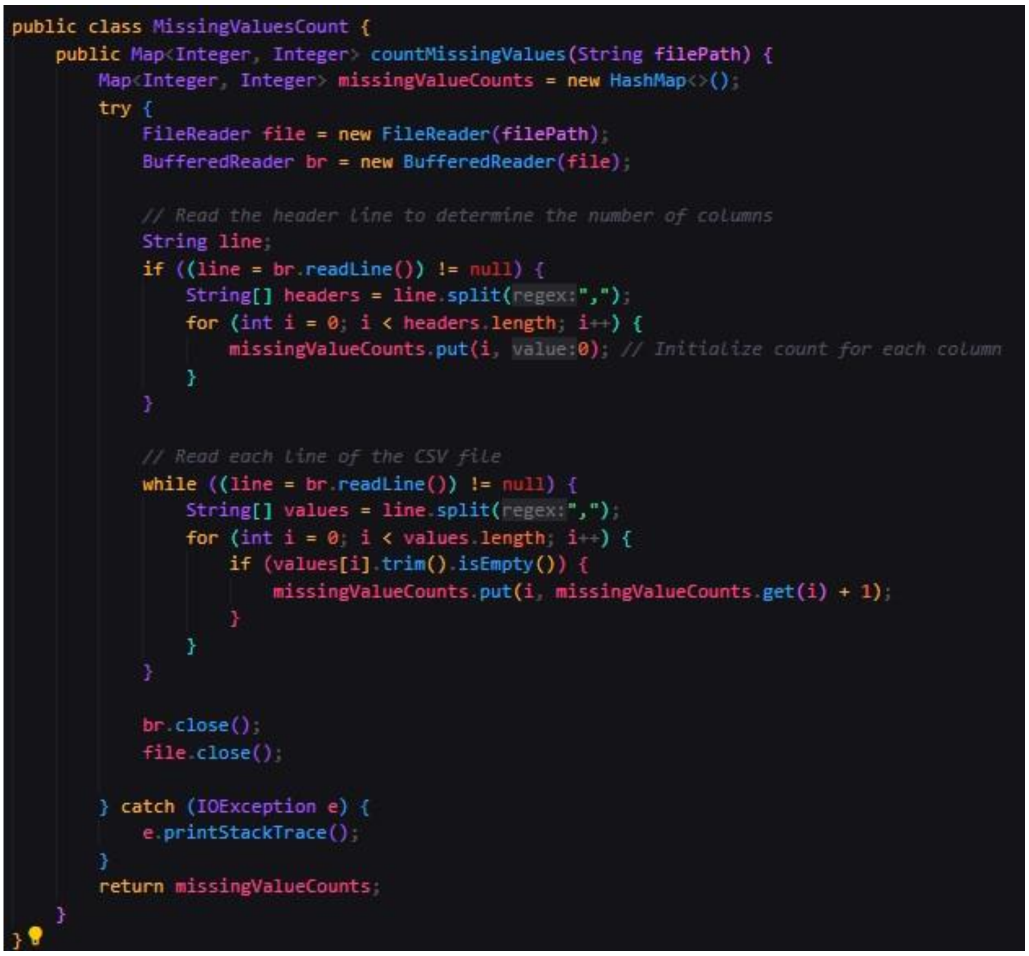


```java
            try {
                Double.parseDouble(dataset.get(index:0)[i]);
                numericalColumns.add(i);
            } catch (NumberFormatException e) {
                // Not a numerical column
            }
        }
        return numericalColumns;
    }

    // Extract data from a specific column within the dataset
    public static List<Double> extractColumn(List<String[]> dataset, int columnIndex) {
        List<Double> columnData = new ArrayList<>();
        for (String[] row : dataset) {
            if (columnIndex < row.length) {
                String columnValue = row[columnIndex];
                try {
                    double value = Double.parseDouble(columnValue);
                    columnData.add(value);
                } catch (NumberFormatException e) {
                    // Handle invalid or non-numeric values in the column
                    System.out.println("Invalid value in the column: " + columnValue);
                }
            }
        }
        return columnData;
    }

    // Method to calculate the mean of a list of numbers
    public static double calculateMean(List<Double> data) {
        if (data.isEmpty()) {
            return 0; // or any other default value
        }
        double sum = 0;
        for (double num : data) {
            sum += num;
        }
        return sum / data.size();
    }

    // Method to calculate the median of a list of numbers
    public static double calculateMedian(List<Double> data) {
        if (data.isEmpty()) {
            return 0; // or any other default value
        }
        data.sort(c:null); // Sort the data in ascending order
        int size = data.size();
        if (size % 2 == 1) {
            return data.get(size / 2);
        } else {
            return (data.get(size / 2 - 1) + data.get(size / 2)) / 2.0;
        }
    }
```



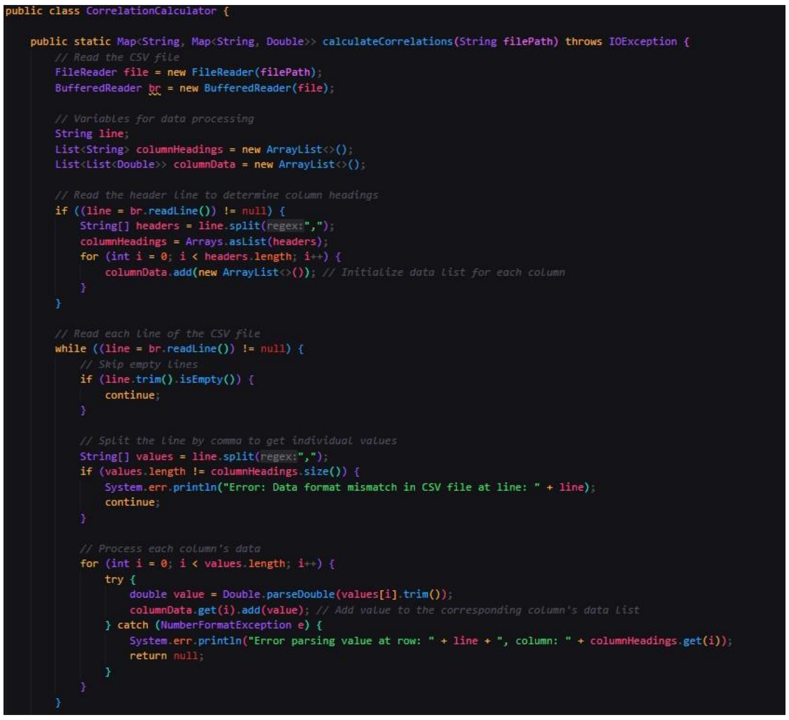
```java
        for (int i = 0; i < data1.size(); i++) {
            double x = data1.get(i);
            double y = data2.get(i);
            sumXY += (x - mean1) * (y - mean2);
            sumX2 += Math.pow(x - mean1, b:2);
            sumY2 += Math.pow(y - mean2, b:2);
        }
        double stdX = Math.sqrt(sumX2 / data1.size());
        double stdY = Math.sqrt(sumY2 / data2.size());

        if (stdX == 0 || stdY == 0) {
            return null; // correlation is not defined when standard deviation is zero
        }

        return sumXY / Math.sqrt(sumX2 * sumY2);
    }

    // Method to calculate quartiles of a list of numbers
    public static double[] calculateQuartiles(List<Double> data) {
        if (data.isEmpty()) {
            return new double[]{0, 0, 0}; // or any other default value
        }
        data.sort(c:null); // Sort the data in ascending order
        int size = data.size();
        double[] quartiles = new double[3];
        quartiles[0] = calculateMedian(data.subList(fromIndex:0, size / 2)); // Q1 (25th percentile)
        quartiles[1] = calculateMedian(data); // Q2 (50th percentile, median)
        if (size % 2 == 0) {
            quartiles[2] = calculateMedian(data.subList(size / 2, size)); // Q3 (75th percentile)
        } else {
            quartiles[2] = calculateMedian(data.subList(size / 2 + 1, size)); // Q3 (75th percentile)
        }
        return quartiles;
    }

    // Method to print statistics
    public static void printStatistics(List<Double> data) {
        System.out.println("Mean: " + calculateMean(data));
        System.out.println("Median: " + calculateMedian(data));
        System.out.println("Mode: " + calculateMode(data));
        System.out.println("Standard Deviation: " + calculateStandardDeviation(data));
        System.out.println("Variance: " + calculateVariance(data));
        double[] quartiles = calculateQuartiles(data);
        System.out.println("Q1 (25th percentile): " + quartiles[0]);
        System.out.println("Q2 (50th percentile, median): " + quartiles[1]);
        System.out.println("Q3 (75th percentile): " + quartiles[2]);
    }
```

V. Missing Values Count

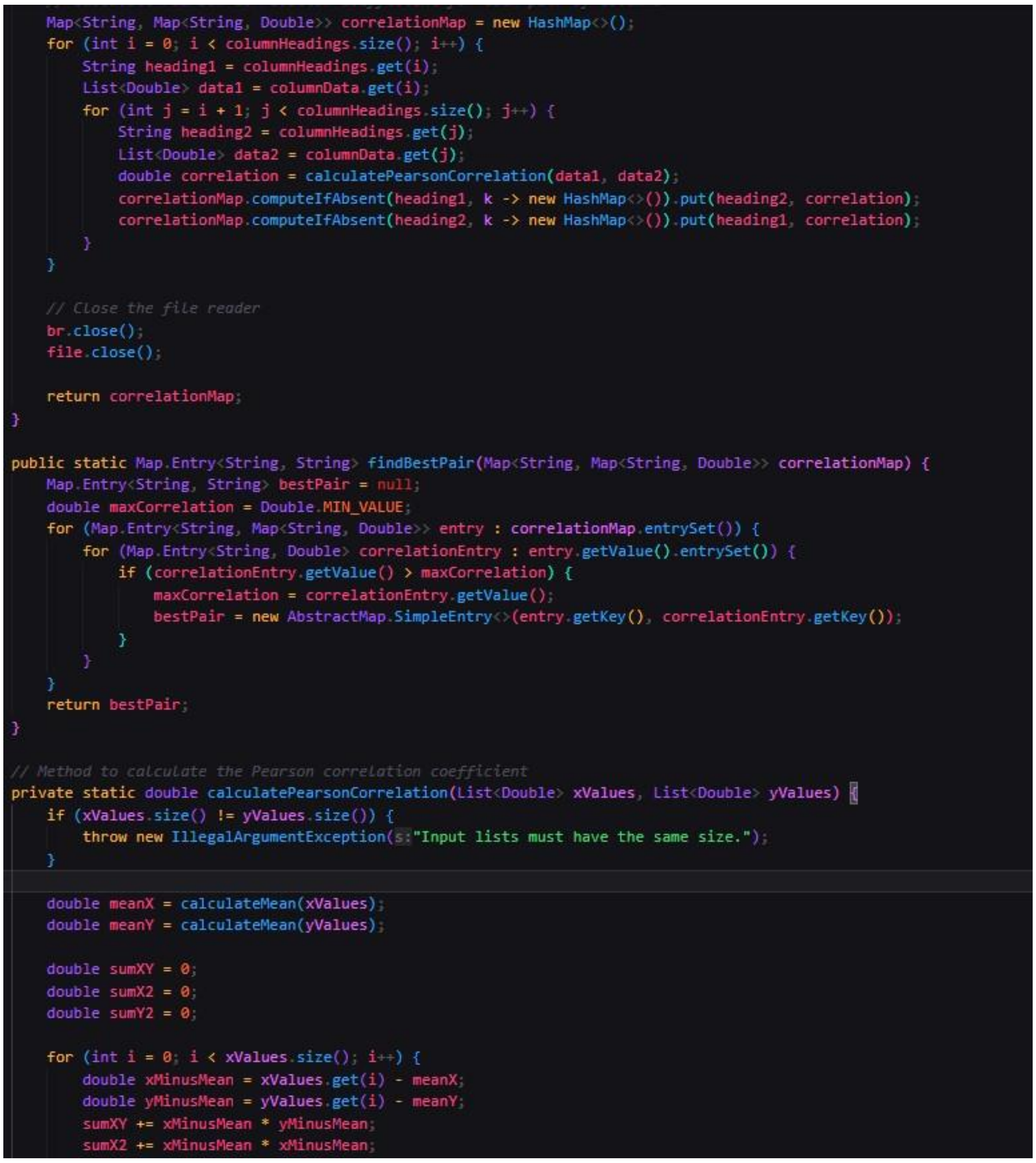


```java
public class MissingValuesCount {
    public Map<Integer, Integer> countMissingValues(String filePath) {
        Map<Integer, Integer> missingValueCounts = new HashMap<>();
        try {
            FileReader file = new FileReader(filePath);
            BufferedReader br = new BufferedReader(file);

            // Read the header line to determine the number of columns
            String line;
            if ((line = br.readLine()) != null) {
                String[] headers = line.split(regex:",");
                for (int i = 0; i < headers.length; i++) {
                    missingValueCounts.put(i, value:0); // Initialize count for each column
                }
            }

            // Read each line of the CSV file
            while ((line = br.readLine()) != null) {
                String[] values = line.split(regex:",");
                for (int i = 0; i < values.length; i++) {
                    if (values[i].trim().isEmpty()) {
                        missingValueCounts.put(i, missingValueCounts.get(i) + 1);
                    }
                }
            }

            br.close();
            file.close();

        } catch (IOException e) {
            e.printStackTrace();
        }
        return missingValueCounts;
    }
}
```

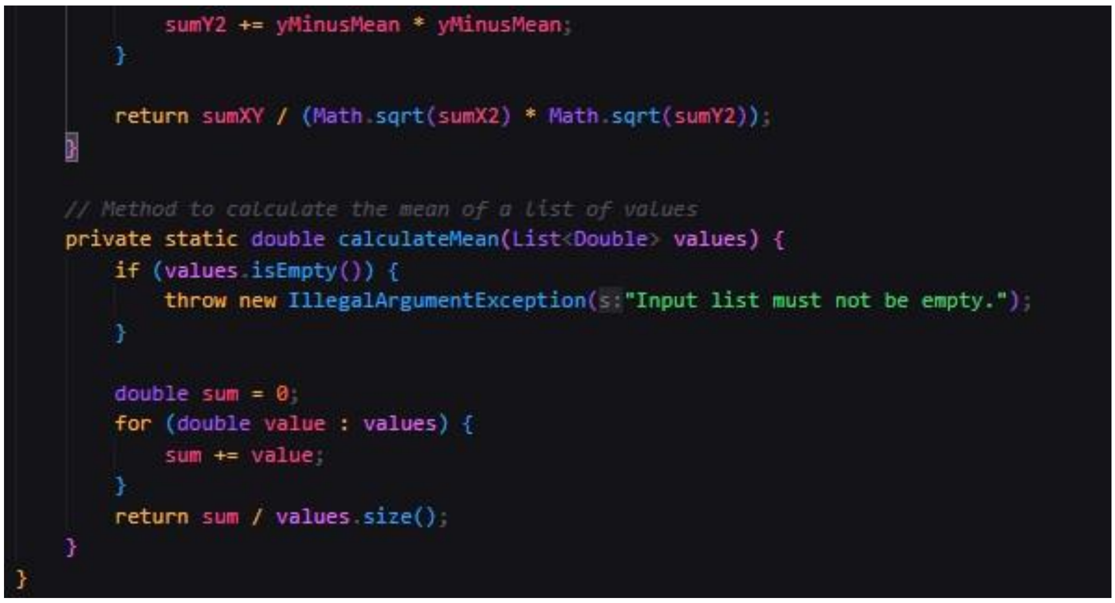
VI. Correlation Calculator



```java
public class CorrelationCalculator {

    public static Map<String, Map<String, Double>> calculateCorrelations(String filePath) throws IOException {
        // Read the CSV file
        FileReader file = new FileReader(filePath);
        BufferedReader br = new BufferedReader(file);

        // Variables for data processing
        String line;
        List<String> columnHeadings = new ArrayList<>();
        List<List<Double>> columnData = new ArrayList<>();

        // Read the header line to determine column headings
        if ((line = br.readLine()) != null) {
            String[] headers = line.split(regex:",");
            columnHeadings = Arrays.asList(headers);
            for (int i = 0; i < headers.length; i++) {
                columnData.add(new ArrayList<>()); // Initialize data list for each column
            }
        }

        // Read each line of the CSV file
        while ((line = br.readLine()) != null) {
            // Skip empty lines
            if (line.trim().isEmpty()) {
                continue;
            }

            // Split the line by comma to get individual values
            String[] values = line.split(regex:",");
            if (values.length != columnHeadings.size()) {
                System.err.println("Error: Data format mismatch in CSV file at line: " + line);
                continue;
            }

            // Process each column's data
            for (int i = 0; i < values.length; i++) {
                try {
                    double value = Double.parseDouble(values[i].trim());
                    columnData.get(i).add(value); // Add value to the corresponding column's data list
                } catch (NumberFormatException e) {
                    System.err.println("Error parsing value at row: " + line + ", column: " + columnHeadings.get(i));
                    return null;
                }
            }
        }
```

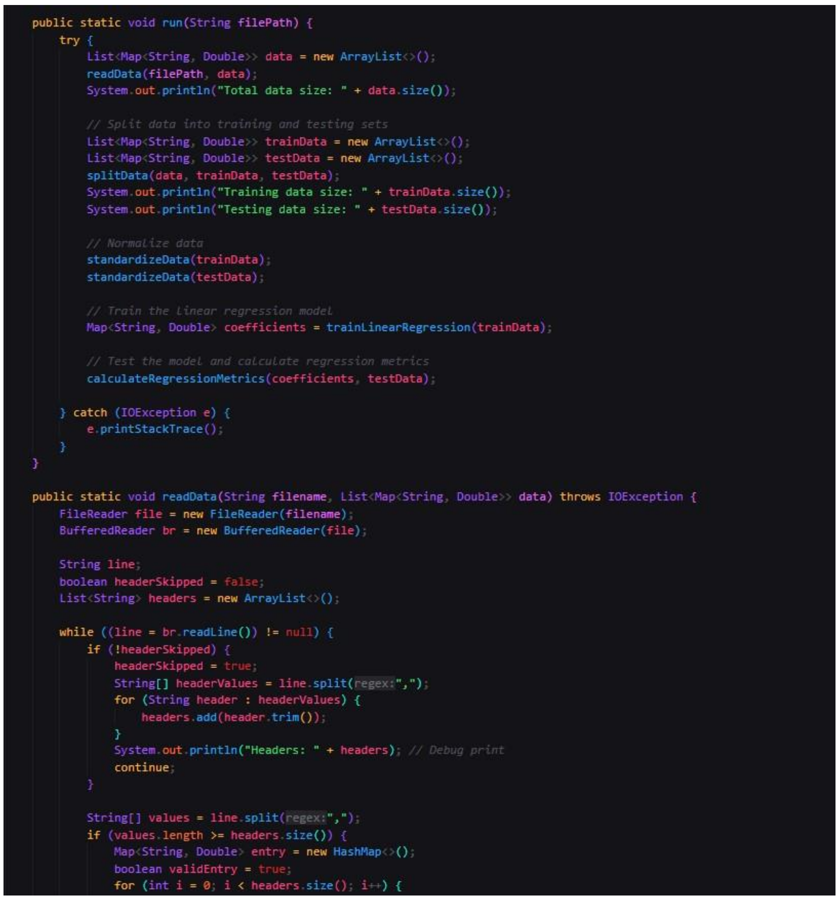


```java
    Map<String, Map<String, Double>> correlationMap = new HashMap<>();
    for (int i = 0; i < columnHeadings.size(); i++) {
        String heading1 = columnHeadings.get(i);
        List<Double> data1 = columnData.get(i);
        for (int j = i + 1; j < columnHeadings.size(); j++) {
            String heading2 = columnHeadings.get(j);
            List<Double> data2 = columnData.get(j);
            double correlation = calculatePearsonCorrelation(data1, data2);
            correlationMap.computeIfAbsent(heading1, k -> new HashMap<>()).put(heading2, correlation);
            correlationMap.computeIfAbsent(heading2, k -> new HashMap<>()).put(heading1, correlation);
        }
    }

    // Close the file reader
    br.close();
    file.close();

    return correlationMap;
}

public static Map.Entry<String, String> findBestPair(Map<String, Map<String, Double>> correlationMap) {
    Map.Entry<String, String> bestPair = null;
    double maxCorrelation = Double.MIN_VALUE;
    for (Map.Entry<String, Map<String, Double>> entry : correlationMap.entrySet()) {
        for (Map.Entry<String, Double> correlationEntry : entry.getValue().entrySet()) {
            if (correlationEntry.getValue() > maxCorrelation) {
                maxCorrelation = correlationEntry.getValue();
                bestPair = new AbstractMap.SimpleEntry<>(entry.getKey(), correlationEntry.getKey());
            }
        }
    }
    return bestPair;
}

// Method to calculate the Pearson correlation coefficient
private static double calculatePearsonCorrelation(List<Double> xValues, List<Double> yValues) {
    if (xValues.size() != yValues.size()) {
        throw new IllegalArgumentException(s:"Input lists must have the same size.");
    }

    double meanX = calculateMean(xValues);
    double meanY = calculateMean(yValues);

    double sumXY = 0;
    double sumX2 = 0;
    double sumY2 = 0;

    for (int i = 0; i < xValues.size(); i++) {
        double xMinusMean = xValues.get(i) - meanX;
        double yMinusMean = yValues.get(i) - meanY;
        sumXY += xMinusMean * yMinusMean;
        sumX2 += xMinusMean * xMinusMean;
```

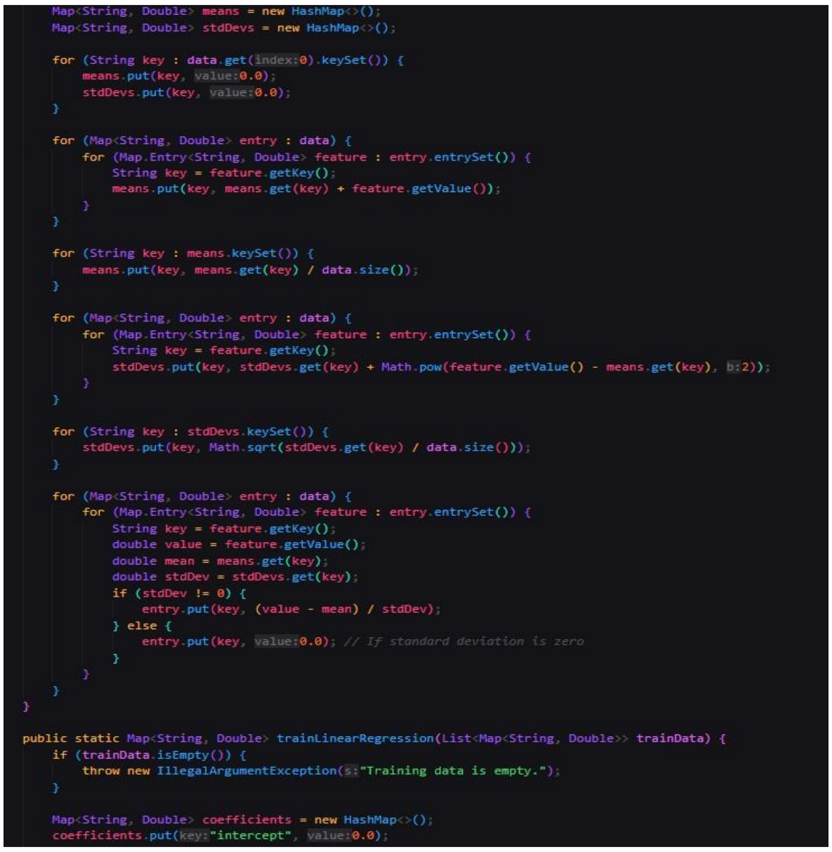


```java
            sumY2 += yMinusMean * yMinusMean;
        }

        return sumXY / (Math.sqrt(sumX2) * Math.sqrt(sumY2));
    }

    // Method to calculate the mean of a list of values
    private static double calculateMean(List<Double> values) {
        if (values.isEmpty()) {
            throw new IllegalArgumentException(s:"Input list must not be empty.");
        }

        double sum = 0;
        for (double value : values) {
            sum += value;
        }
        return sum / values.size();
    }
}
```

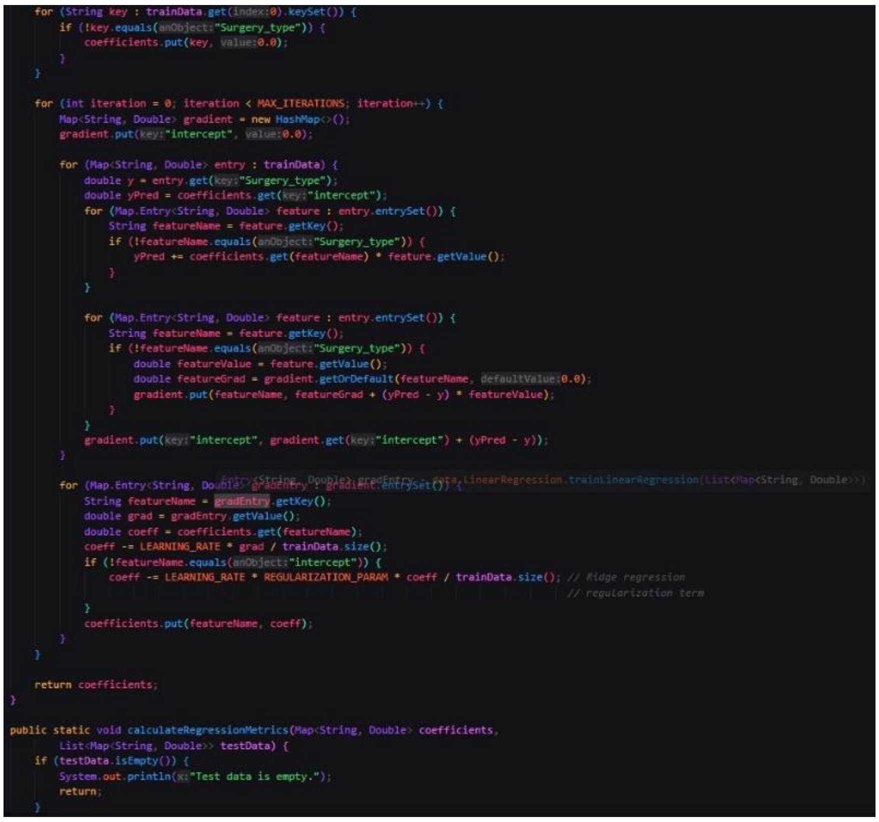
VII. Linear Regression Calculator



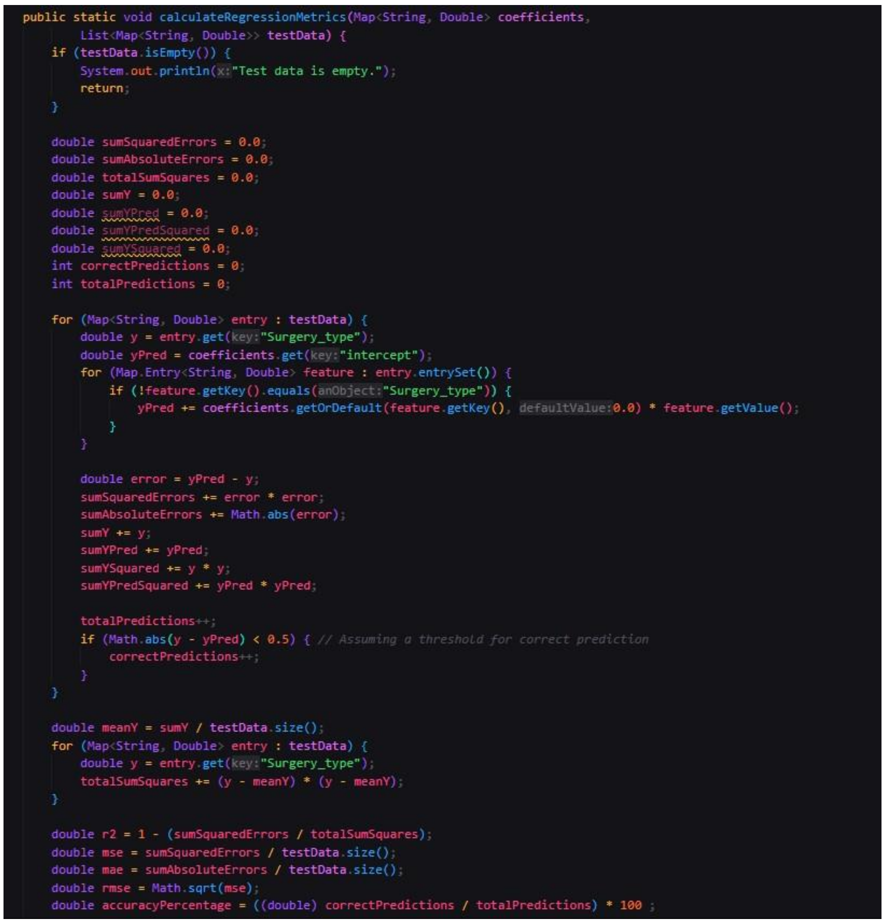
```java
public static void run(String filePath) {
    try {
        List<Map<String, Double>> data = new ArrayList<>();
        readData(filePath, data);
        System.out.println("Total data size: " + data.size());

        // Split data into training and testing sets
        List<Map<String, Double>> trainData = new ArrayList<>();
        List<Map<String, Double>> testData = new ArrayList<>();
        splitData(data, trainData, testData);
        System.out.println("Training data size: " + trainData.size());
        System.out.println("Testing data size: " + testData.size());

        // Normalize data
        standardizeData(trainData);
        standardizeData(testData);

        // Train the linear regression model
        Map<String, Double> coefficients = trainLinearRegression(trainData);

        // Test the model and calculate regression metrics
        calculateRegressionMetrics(coefficients, testData);

    } catch (IOException e) {
        e.printStackTrace();
    }
}

public static void readData(String filename, List<Map<String, Double>> data) throws IOException {
    FileReader file = new FileReader(filename);
    BufferedReader br = new BufferedReader(file);

    String line;
    boolean headerSkipped = false;
    List<String> headers = new ArrayList<>();

    while ((line = br.readLine()) != null) {
        if (!headerSkipped) {
            headerSkipped = true;
            String[] headerValues = line.split(regex:",");
            for (String header : headerValues) {
                headers.add(header.trim());
            }
            System.out.println("Headers: " + headers); // Debug print
            continue;
        }

        String[] values = line.split(regex:",");
        if (values.length >= headers.size()) {
            Map<String, Double> entry = new HashMap<>();
            boolean validEntry = true;
            for (int i = 0; i < headers.size(); i++) {
```



```java
    Map<String, Double> means = new HashMap<>();
    Map<String, Double> stdDevs = new HashMap<>();

    for (String key : data.get(index:0).keySet()) {
        means.put(key, value:0.0);
        stdDevs.put(key, value:0.0);
    }

    for (Map<String, Double> entry : data) {
        for (Map.Entry<String, Double> feature : entry.entrySet()) {
            String key = feature.getKey();
            means.put(key, means.get(key) + feature.getValue());
        }
    }

    for (String key : means.keySet()) {
        means.put(key, means.get(key) / data.size());
    }

    for (Map<String, Double> entry : data) {
        for (Map.Entry<String, Double> feature : entry.entrySet()) {
            String key = feature.getKey();
            stdDevs.put(key, stdDevs.get(key) + Math.pow(feature.getValue() - means.get(key), b:2));
        }
    }

    for (String key : stdDevs.keySet()) {
        stdDevs.put(key, Math.sqrt(stdDevs.get(key) / data.size()));
    }

    for (Map<String, Double> entry : data) {
        for (Map.Entry<String, Double> feature : entry.entrySet()) {
            String key = feature.getKey();
            double value = feature.getValue();
            double mean = means.get(key);
            double stdDev = stdDevs.get(key);
            if (stdDev != 0) {
                entry.put(key, (value - mean) / stdDev);
            } else {
                entry.put(key, value:0.0); // If standard deviation is zero
            }
        }
    }
}

public static Map<String, Double> trainLinearRegression(List<Map<String, Double>> trainData) {
    if (trainData.isEmpty()) {
        throw new IllegalArgumentException(s:"Training data is empty.");
    }

    Map<String, Double> coefficients = new HashMap<>();
    coefficients.put(key:"intercept", value:0.0);
```



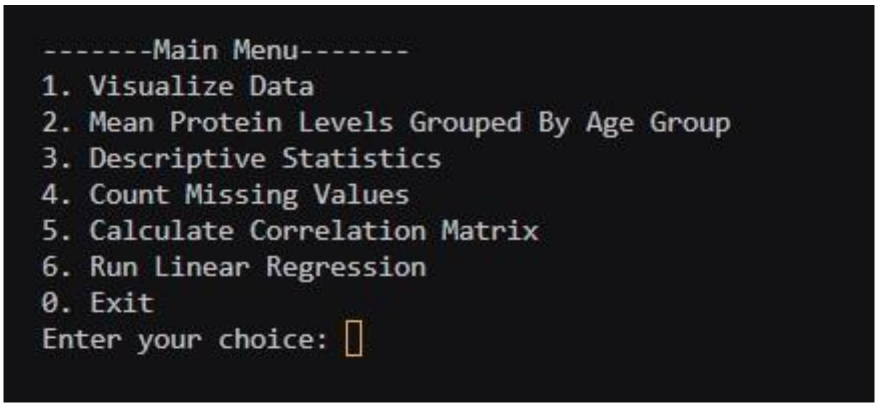
```java
    for (String key : trainData.get(index:0).keySet()) {
        if (!key.equals(anObject:"Surgery_type")) {
            coefficients.put(key, value:0.0);
        }
    }

    for (int iteration = 0; iteration < MAX_ITERATIONS; iteration++) {
        Map<String, Double> gradient = new HashMap<>();
        gradient.put(key:"intercept", value:0.0);

        for (Map<String, Double> entry : trainData) {
            double y = entry.get(key:"Surgery_type");
            double yPred = coefficients.get(key:"intercept");
            for (Map.Entry<String, Double> feature : entry.entrySet()) {
                String featureName = feature.getKey();
                if (!featureName.equals(anObject:"Surgery_type")) {
                    yPred += coefficients.get(featureName) * feature.getValue();
                }
            }

            for (Map.Entry<String, Double> feature : entry.entrySet()) {
                String featureName = feature.getKey();
                if (!featureName.equals(anObject:"Surgery_type")) {
                    double featureValue = feature.getValue();
                    double featureGrad = gradient.getOrDefault(featureName, defaultValue:0.0);
                    gradient.put(featureName, featureGrad + (yPred - y) * featureValue);
                }
            }
            gradient.put(key:"intercept", gradient.get(key:"intercept") + (yPred - y));
        }

        for (Map.Entry<String, Double> gradEntry : gradient.entrySet()) {   Entry<String, Double> gradEntry - data.LinearRegression.trainLinearRegression(List<Map<String, Double>>)
            String featureName = gradEntry.getKey();
            double grad = gradEntry.getValue();
            double coeff = coefficients.get(featureName);
            coeff -= LEARNING_RATE * grad / trainData.size();
            if (!featureName.equals(anObject:"intercept")) {
                coeff -= LEARNING_RATE * REGULARIZATION_PARAM * coeff / trainData.size(); // Ridge regression
                                                                                          // regularization term
            }
            coefficients.put(featureName, coeff);
        }
    }

    return coefficients;
}

public static void calculateRegressionMetrics(Map<String, Double> coefficients,
        List<Map<String, Double>> testData) {
    if (testData.isEmpty()) {
        System.out.println(x:"Test data is empty.");
        return;
    }
```



```java
public static void calculateRegressionMetrics(Map<String, Double> coefficients,
        List<Map<String, Double>> testData) {
    if (testData.isEmpty()) {
        System.out.println(x:"Test data is empty.");
        return;
    }

    double sumSquaredErrors = 0.0;
    double sumAbsoluteErrors = 0.0;
    double totalSumSquares = 0.0;
    double sumY = 0.0;
    double sumYPred = 0.0;
    double sumYPredSquared = 0.0;
    double sumYSquared = 0.0;
    int correctPredictions = 0;
    int totalPredictions = 0;

    for (Map<String, Double> entry : testData) {
        double y = entry.get(key:"Surgery_type");
        double yPred = coefficients.get(key:"intercept");
        for (Map.Entry<String, Double> feature : entry.entrySet()) {
            if (!feature.getKey().equals(anObject:"Surgery_type")) {
                yPred += coefficients.getOrDefault(feature.getKey(), defaultValue:0.0) * feature.getValue();
            }
        }

        double error = yPred - y;
        sumSquaredErrors += error * error;
        sumAbsoluteErrors += Math.abs(error);
        sumY += y;
        sumYPred += yPred;
        sumYSquared += y * y;
        sumYPredSquared += yPred * yPred;

        totalPredictions++;
        if (Math.abs(y - yPred) < 0.5) { // Assuming a threshold for correct prediction
            correctPredictions++;
        }
    }

    double meanY = sumY / testData.size();
    for (Map<String, Double> entry : testData) {
        double y = entry.get(key:"Surgery_type");
        totalSumSquares += (y - meanY) * (y - meanY);
    }

    double r2 = 1 - (sumSquaredErrors / totalSumSquares);
    double mse = sumSquaredErrors / testData.size();
    double mae = sumAbsoluteErrors / testData.size();
    double rmse = Math.sqrt(mse);
    double accuracyPercentage = ((double) correctPredictions / totalPredictions) * 100 ;
```

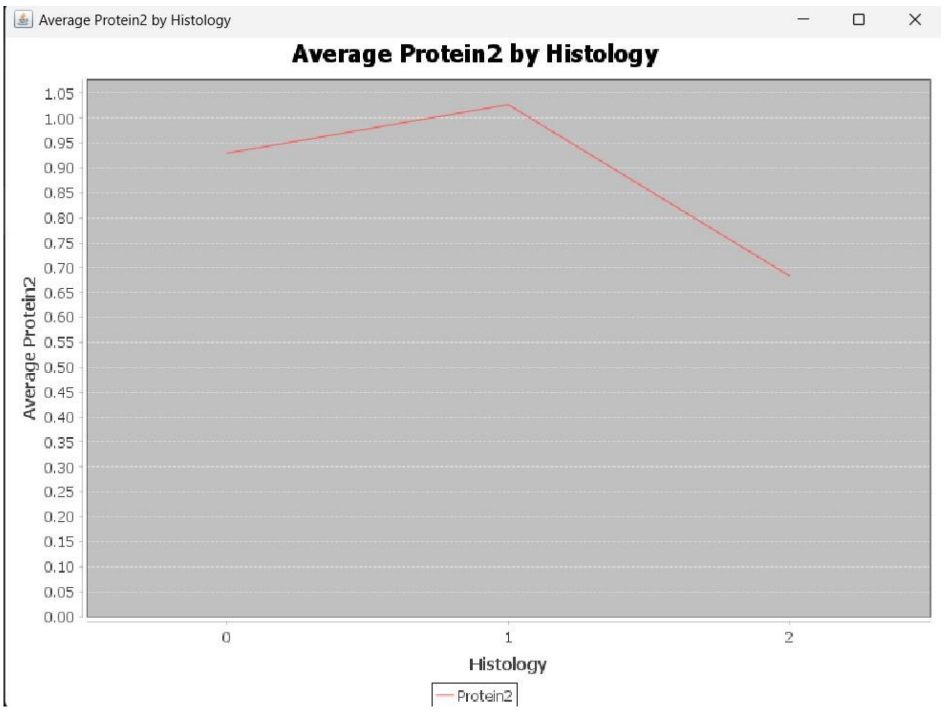
Output

I. Menu Driven

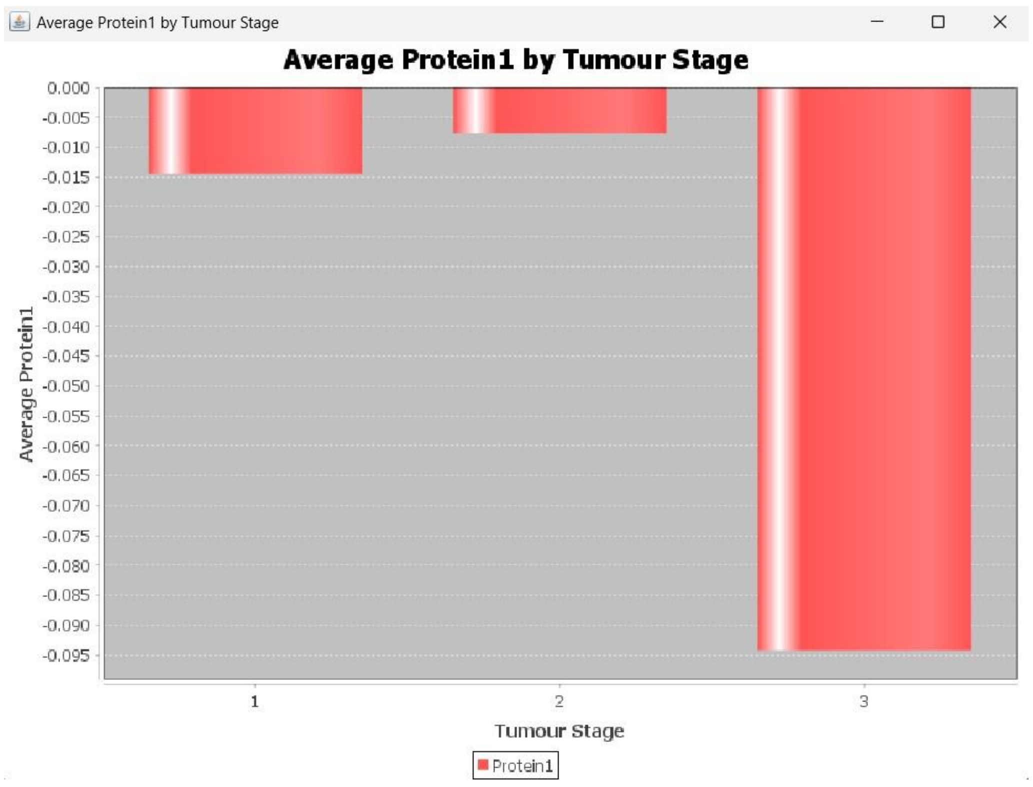


```
-------Main Menu-------
1. Visualize Data
2. Mean Protein Levels Grouped By Age Group
3. Descriptive Statistics
4. Count Missing Values
5. Calculate Correlation Matrix
6. Run Linear Regression
0. Exit
Enter your choice: 
```

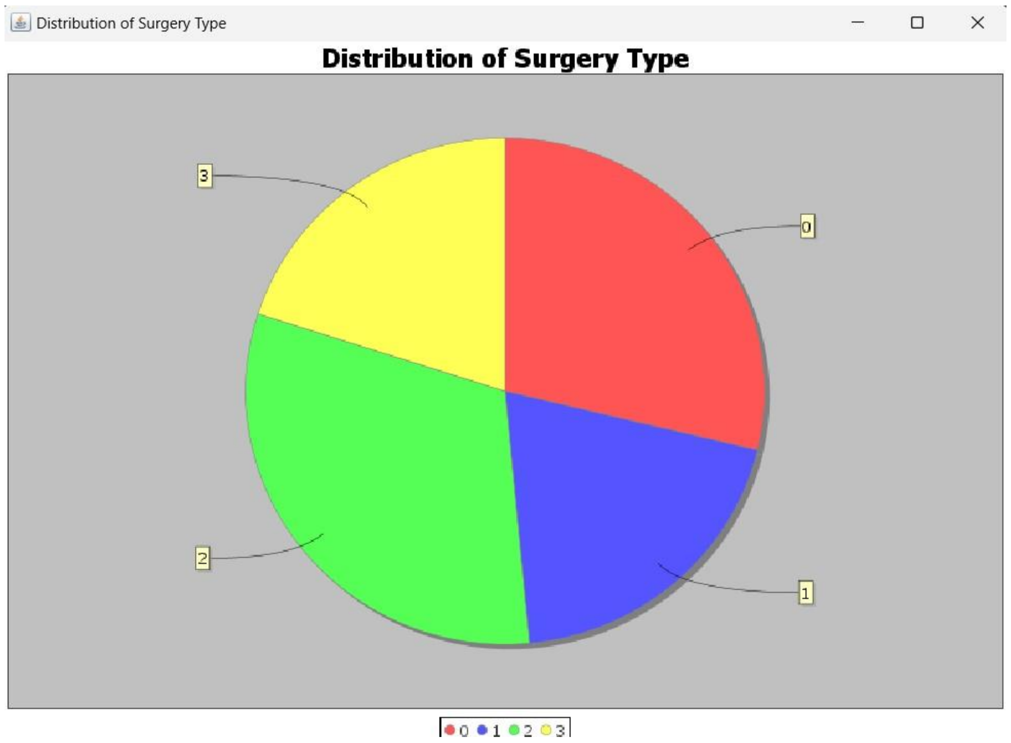
II. Data Visualization



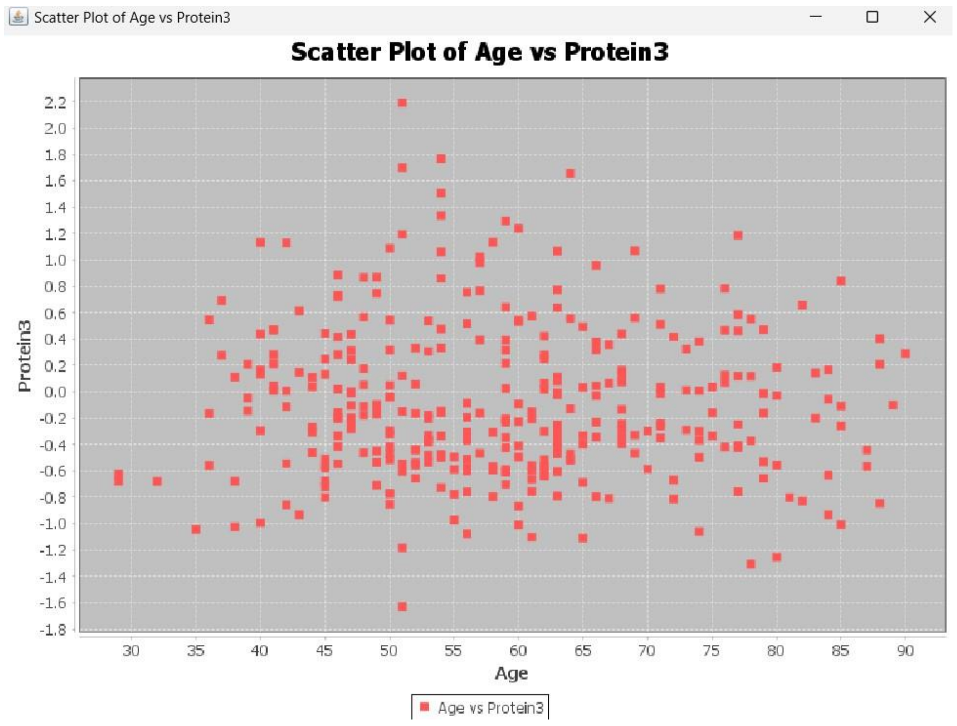
The graph shows rising levels of protein 2 across various tissue types (histologic features). This could be due to increased overall protein 2 production, changes in the tissues themselves, or biases in the measurement method. More context is needed to determine the cause.



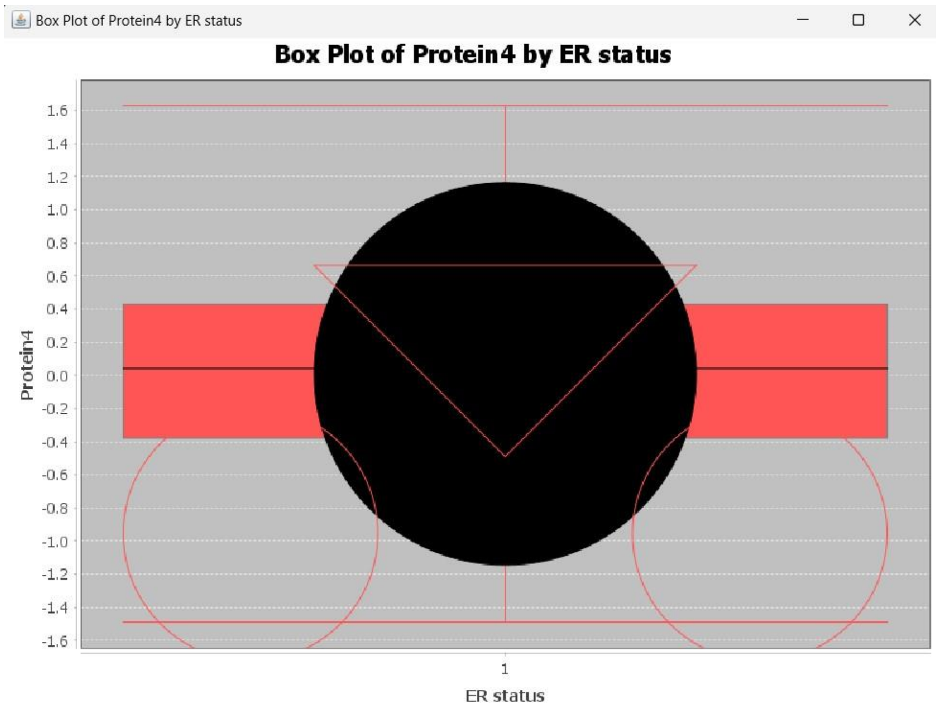
The graph shows the average level of protein 1 relative to tumour stage. However, the values on the y-axis are negative. It is difficult to interpret the meaning of negative protein levels from this graph alone. It could be that there is an error in the data, or that the way protein levels are measured in this experiment results in negative values.



This graph shows the distribution of Surgery type based on the counts of Surgeries happened in the given dataset.



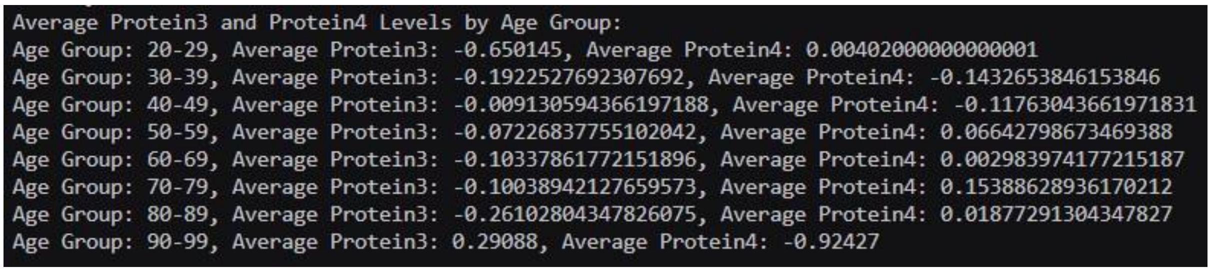
We can see that there is a positive correlation between age and protein3. This means that as age increases, protein3 levels also tend to increase. Protein3 is a nutrient that is essential for the body, and it helps to build and repair tissues. The scatter plot also shows that there is some variation in protein3 levels for each age group, so not everyone will follow this trend.



The box plot shows lower protein4 levels in the ER positive group compared to the ER negative group. There's also more variation in protein4 levels for the ER negative group.

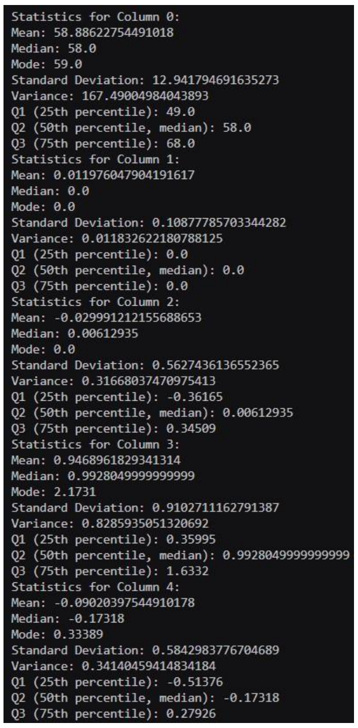
ode

III. Average Protein Levels Categorized by Age

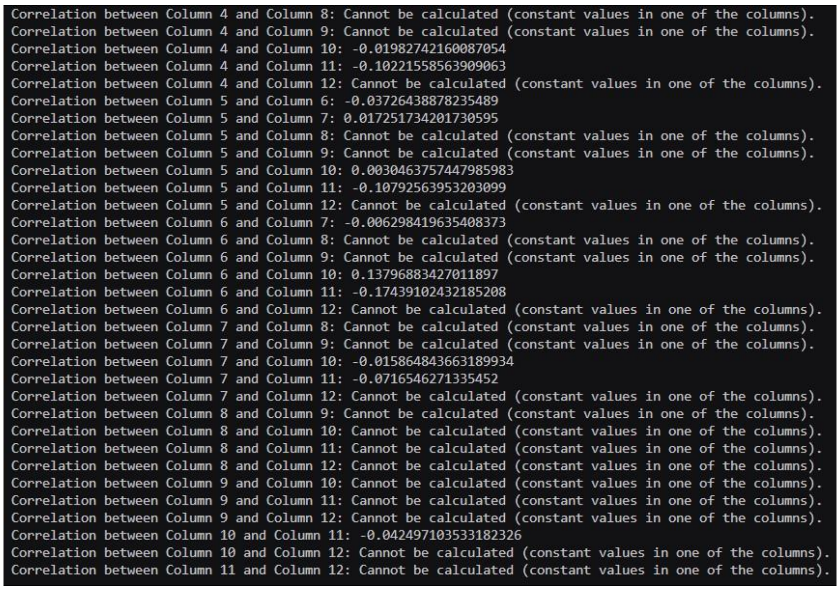


```
Average Protein3 and Protein4 Levels by Age Group:
Age Group: 20-29, Average Protein3: -0.650145, Average Protein4: 0.00402000000000001
Age Group: 30-39, Average Protein3: -0.1922527692307692, Average Protein4: -0.1432653846153846
Age Group: 40-49, Average Protein3: -0.009130594366197188, Average Protein4: -0.11763043661971831
Age Group: 50-59, Average Protein3: -0.07226837755102042, Average Protein4: 0.06642798673469388
Age Group: 60-69, Average Protein3: -0.10337861772151896, Average Protein4: 0.002983974177215187
Age Group: 70-79, Average Protein3: -0.10038942127659573, Average Protein4: 0.15388628936170212
Age Group: 80-89, Average Protein3: -0.26102804347826075, Average Protein4: 0.01877291304347827
Age Group: 90-99, Average Protein3: 0.29088, Average Protein4: -0.92427
```

IV. Descriptive Statistics

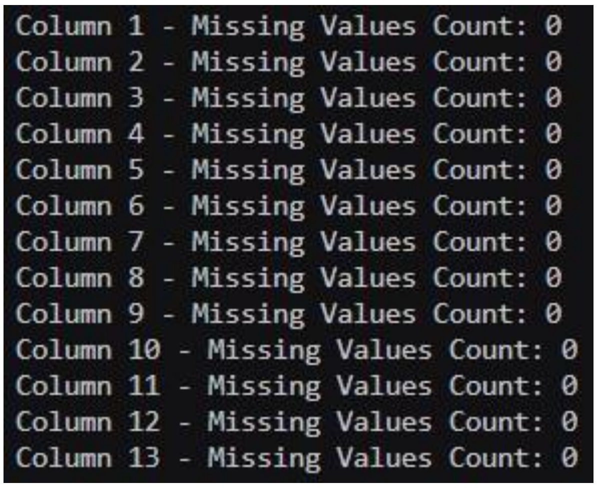


```
Statistics for Column 0:
Mean: 58.88622754491018
Median: 58.0
Mode: 59.0
Standard Deviation: 12.941794691635273
Variance: 167.49004984043893
Q1 (25th percentile): 49.0
Q2 (50th percentile, median): 58.0
Q3 (75th percentile): 68.0
Statistics for Column 1:
Mean: 0.011976047904191617
Median: 0.0
Mode: 0.0
Standard Deviation: 0.10877785703344282
Variance: 0.011832622180788125
Q1 (25th percentile): 0.0
Q2 (50th percentile, median): 0.0
Q3 (75th percentile): 0.0
Statistics for Column 2:
Mean: -0.029991212155688653
Median: 0.00612935
Mode: 0.0
Standard Deviation: 0.5627436136552365
Variance: 0.31668037470975413
Q1 (25th percentile): -0.36165
Q2 (50th percentile, median): 0.00612935
Q3 (75th percentile): 0.34509
Statistics for Column 3:
Mean: 0.9468961829341314
Median: 0.9928049999999999
Mode: 2.1731
Standard Deviation: 0.9102711162791387
Variance: 0.8285935051320692
Q1 (25th percentile): 0.35995
Q2 (50th percentile, median): 0.9928049999999999
Q3 (75th percentile): 1.6332
Statistics for Column 4:
Mean: -0.09020397544910178
Median: -0.17318
Mode: 0.33389
Standard Deviation: 0.5842983776704689
Variance: 0.34140459414834184
Q1 (25th percentile): -0.51376
Q2 (50th percentile, median): -0.17318
Q3 (75th percentile): 0.27926
```



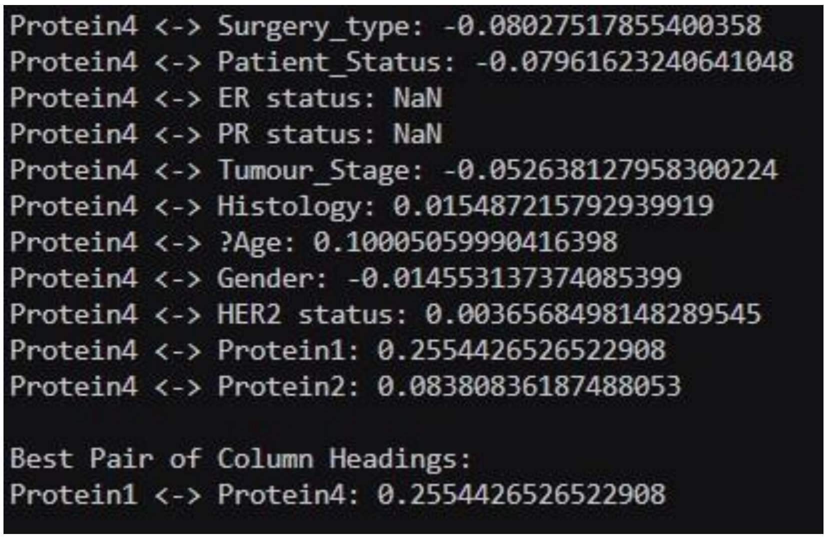
```
Correlation between Column 4 and Column 8: Cannot be calculated (constant values in one of the columns).
Correlation between Column 4 and Column 9: Cannot be calculated (constant values in one of the columns).
Correlation between Column 4 and Column 10: -0.01982742160087054
Correlation between Column 4 and Column 11: -0.10221558563909063
Correlation between Column 4 and Column 12: Cannot be calculated (constant values in one of the columns).
Correlation between Column 5 and Column 6: -0.03726438878235489
Correlation between Column 5 and Column 7: 0.017251734201730595
Correlation between Column 5 and Column 8: Cannot be calculated (constant values in one of the columns).
Correlation between Column 5 and Column 9: Cannot be calculated (constant values in one of the columns).
Correlation between Column 5 and Column 10: 0.0030463757447985983
Correlation between Column 5 and Column 11: -0.10792563953203099
Correlation between Column 5 and Column 12: Cannot be calculated (constant values in one of the columns).
Correlation between Column 6 and Column 7: -0.006298419635408373
Correlation between Column 6 and Column 8: Cannot be calculated (constant values in one of the columns).
Correlation between Column 6 and Column 9: Cannot be calculated (constant values in one of the columns).
Correlation between Column 6 and Column 10: 0.13796883427011897
Correlation between Column 6 and Column 11: -0.17439102432185208
Correlation between Column 6 and Column 12: Cannot be calculated (constant values in one of the columns).
Correlation between Column 7 and Column 8: Cannot be calculated (constant values in one of the columns).
Correlation between Column 7 and Column 9: Cannot be calculated (constant values in one of the columns).
Correlation between Column 7 and Column 10: -0.015864843663189934
Correlation between Column 7 and Column 11: -0.0716546271335452
Correlation between Column 7 and Column 12: Cannot be calculated (constant values in one of the columns).
Correlation between Column 8 and Column 9: Cannot be calculated (constant values in one of the columns).
Correlation between Column 8 and Column 10: Cannot be calculated (constant values in one of the columns).
Correlation between Column 8 and Column 11: Cannot be calculated (constant values in one of the columns).
Correlation between Column 8 and Column 12: Cannot be calculated (constant values in one of the columns).
Correlation between Column 9 and Column 10: Cannot be calculated (constant values in one of the columns).
Correlation between Column 9 and Column 11: Cannot be calculated (constant values in one of the columns).
Correlation between Column 9 and Column 12: Cannot be calculated (constant values in one of the columns).
Correlation between Column 10 and Column 11: -0.042497103533182326
Correlation between Column 10 and Column 12: Cannot be calculated (constant values in one of the columns).
Correlation between Column 11 and Column 12: Cannot be calculated (constant values in one of the columns).
```

V. Count Missing Values



```
Column 1 - Missing Values Count: 0
Column 2 - Missing Values Count: 0
Column 3 - Missing Values Count: 0
Column 4 - Missing Values Count: 0
Column 5 - Missing Values Count: 0
Column 6 - Missing Values Count: 0
Column 7 - Missing Values Count: 0
Column 8 - Missing Values Count: 0
Column 9 - Missing Values Count: 0
Column 10 - Missing Values Count: 0
Column 11 - Missing Values Count: 0
Column 12 - Missing Values Count: 0
Column 13 - Missing Values Count: 0
```

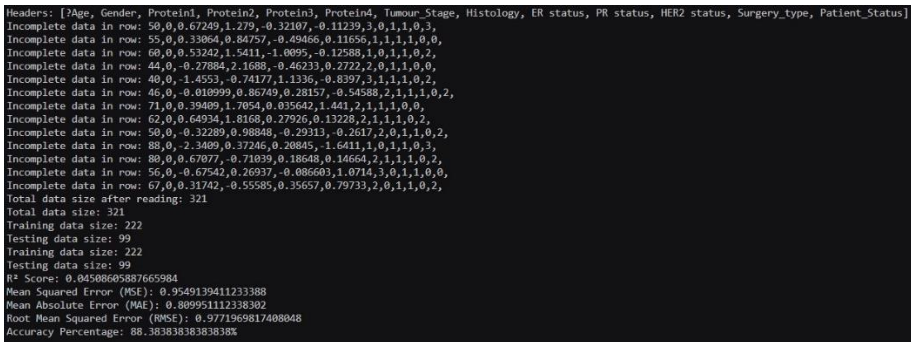
VI. Correlation Matrix



```
Protein4 <-> Surgery_type: -0.08027517855400358
Protein4 <-> Patient_Status: -0.07961623240641048
Protein4 <-> ER status: NaN
Protein4 <-> PR status: NaN
Protein4 <-> Tumour_Stage: -0.052638127958300224
Protein4 <-> Histology: 0.015487215792939919
Protein4 <-> ?Age: 0.10005059990416398
Protein4 <-> Gender: -0.014553137374085399
Protein4 <-> HER2 status: 0.0036568498148289545
Protein4 <-> Protein1: 0.2554426526522908
Protein4 <-> Protein2: 0.08380836187488053

Best Pair of Column Headings:
Protein1 <-> Protein4: 0.2554426526522908
```

The output displays the correlation coefficients between Protein4 levels and various features. Protein1 shows the highest positive correlation with Protein4 (0.255), indicating a moderate relationship. Tumour_Stage and Age have weaker correlations, while Surgery_type and Patient_Status show slight negative correlations. ER status and PR status have NaN values, indicating missing data. The strongest correlation is between Protein1 and Protein4.

VII. Linear Regression Model



```
Headers: [?Age, Gender, Protein1, Protein2, Protein3, Protein4, Tumour_Stage, Histology, ER status, PR status, HER2 status, Surgery_type, Patient_Status]
Incomplete data in row: 50,0,0.67249,1.279,-0.32107,-0.11239,3,0,1,1,0,3,
Incomplete data in row: 55,0,0.33064,0.84757,-0.49466,0.11656,1,1,1,1,0,0,
Incomplete data in row: 60,0,0.53242,1.5411,-1.0095,-0.12588,1,0,1,1,0,2,
Incomplete data in row: 44,0,-0.27884,2.1688,-0.46233,0.2722,2,0,1,1,0,0,
Incomplete data in row: 40,0,-1.4553,-0.74177,1.1336,-0.8397,3,1,1,1,0,2,
Incomplete data in row: 46,0,-0.010999,0.86749,0.28157,-0.54588,2,1,1,1,0,2,
Incomplete data in row: 71,0,0.39409,1.7054,0.035642,1.441,2,1,1,1,0,0,
Incomplete data in row: 62,0,0.64934,1.8168,0.27926,0.13228,2,1,1,1,0,2,
Incomplete data in row: 50,0,-0.32289,0.98848,-0.29313,-0.2617,2,0,1,1,0,2,
Incomplete data in row: 88,0,-2.3409,0.37246,0.20845,-1.6411,1,0,1,1,0,3,
Incomplete data in row: 80,0,0.67077,-0.71039,0.18648,0.14664,2,1,1,1,0,2,
Incomplete data in row: 56,0,-0.67542,0.26937,-0.086603,1.0714,3,0,1,1,0,0,
Incomplete data in row: 67,0,0.31742,-0.55585,0.35657,0.79733,2,0,1,1,0,2,
Total data size after reading: 321
Total data size: 321
Training data size: 222
Testing data size: 99
Training data size: 222
Testing data size: 99
R² Score: 0.04508605887665984
Mean Squared Error (MSE): 0.9549139411233388
Mean Absolute Error (MAE): 0.809951112338302
Root Mean Squared Error (RMSE): 0.9771969817408048
Accuracy Percentage: 88.38383838383838%
```

The dataset initially had 321 entries, but some rows contained incomplete data. After preprocessing, the total dataset size remained 321, with 222 samples used for training and 99 for testing. The model achieved an R^2 score of 0.045, indicating low explanatory power. The mean squared error (MSE) is 0.954, the mean absolute error (MAE) is 0.899, and the root mean squared error (RMSE) is 0.978. Despite the low R^2 score, the accuracy percentage stands at 88.38%, suggesting the model may still perform reasonably well in practical terms, though further tuning and evaluation are necessary.

Conclusion

In this project, we aimed to predict Protein4 expression levels in breast cancer patients using a linear regression model. The dataset included a comprehensive range of features such as Protein1, Protein2, Protein3, Age, Gender, Tumor_Stage, Histology, ER status, PR status, HER2 status, Surgery_type, and Patient_Status. Data cleaning involved handling missing values, normalizing features, and encoding categorical variables, resulting in 321 complete entries. Data visualization techniques, including histograms and scatter plots, were employed to identify patterns and outliers, enhancing our understanding of the dataset.A correlation matrix was generated to uncover the relationships between Protein4 and other features, revealing Protein1 as the most significant predictor with a correlation of 0.255. Other features showed weaker correlations, and ER and PR statuses had missing data. The linear regression model was then built and evaluated, showing moderate accuracy with an 88.38% accuracy rate but a low R^2 score of 0.045. Key error metrics included a mean squared error (MSE) of 0.954, a mean absolute error (MAE) of 0.899, and a root mean squared error (RMSE) of 0.978.Overall, the project demonstrated the potential of using linear regression to predict Protein4 levels in breast cancer patients. While the model showed reasonable practical accuracy, the low R^2 score indicates room for improvement. Future work could focus on incorporating more advanced machine learning techniques, addressing missing data issues, and exploring additional features to enhance model performance. The insights gained from this study can aid in better understanding the factors influencing Protein4 expression and contribute to more personalized and effective breast cancer treatment strategies.