Spring Semester Probability and Statistic Project

Ahmed Muaz Atik

Explain dataset and explain the column which you choose and why you choose.

I choosed Salary_Data dataset that includes Age, Gender, Education Level, Job Title, Years of Experience and Salary headings.

I choosed Years of Experience column because i wanted to calculate mathematical operations for this column.

Find mean of column data.

```
public static double calculateMean(List<Double> dataset) {
    double sum = 0.0;
    for (double value : dataset) {
        sum += value;
    }
    return sum / dataset.size();
}
```

Find median of column data.

```java
public static double calculateMedian(ArrayList<Double> dataset) {
    int length = dataset.size();
    int middleIndex = length / 2;

    // Custom sorting algorithm (e.g., bubble sort)
    for (int i = 0; i < length - 1; i++) {
        for (int j = 0; j < length - i - 1; j++) {
            if (dataset.get( index: j) > dataset.get(j + 1)) {
                // Swap the elements
                double temp = dataset.get( index: j);
                dataset.set( index: j,  element: dataset.get(j + 1));
                dataset.set(j + 1,  element: temp);
            }
        }
    }

    if (length % 2 == 0) {
        // Even number of values, average the two middle values
        double middleValue1 = dataset.get(middleIndex - 1);
        double middleValue2 = dataset.get( index: middleIndex);
        return (middleValue1 + middleValue2) / 2.0;
    } else {
        // Odd number of values, return the middle value
        return dataset.get( index: middleIndex);
    }
}
```

Find the variance, standard deviation and standard error.

```java
public static double calculateVariance(ArrayList<Double> dataset) {
    double mean = calculateMean(dataset);
    double sumOfSquaredDifferences = 0.0;

    for (double value : dataset) {
        double difference = value - mean;
        double squaredDifference = difference * difference;
        sumOfSquaredDifferences += squaredDifference;
    }

    double variance = sumOfSquaredDifferences / dataset.size();
    return variance;
}
```

```java
public static double calculateStandardError(ArrayList<Double> dataset) {
    double standardDeviation = calculateStandardDeviation(dataset);
    int sampleSize = dataset.size();

    // Calculate the square root of the sample size
    double sqrtSampleSize = customSqrt( value: sampleSize);

    // Calculate the standard error using division
    double standardError = standardDeviation / sqrtSampleSize;

    return standardError;
}
```

```java
public static double calculateStandardDeviation(ArrayList<Double> dataset) {
    double variance = calculateVariance(dataset);

    // Calculate the square root of the variance
    double standardDeviation = customSqrt( value: variance);

    return standardDeviation;
}
```

Decide the shape of distribution.

```
// Decide the shape of the distribution
if (isNormal) {
    System.out.println( x: "Normally Distributed");
} else {
    System.out.println( x: "Not Normally Distributed");
}
System.out.println("Skewness: " + skewness);

System.out.println("Kurtosis: " + kurtosis);
```

Find outliers if there is.

```java
public static List<Double> findOutliers(ArrayList<Double> dataset) {
    List<Double> outliers = new ArrayList<>();
    double mean = calculateMean(dataset);
    double standardDeviation = calculateStandardDeviation(dataset);
    double threshold = 2.0; // Threshold for outliers (can be adjusted)
    double deviationThreshold = threshold * standardDeviation;

    for (double value : dataset) {
        double deviation = customAbs(value - mean);
        if (deviation > deviationThreshold && !outliers.contains( o:value)) {
            outliers.add( e:value);
        }
    }

    return outliers;
}
```

Graph the column data using histogram and make comment about data.

```
System.out.println( x: "Histogram: ");

// Define histogram parameters
int numBins = 10;
double minValue = getMinValue(data);
double maxValue = getMaxValue(data);

// Calculate bin width
double binWidth = (maxValue - minValue) / numBins;

// Initialize bin counts
int[] binCounts = new int[numBins];

// Populate bin counts
for (double value : data) {
    int binIndex = (int) ((value - minValue) / binWidth);
    if (binIndex >= 0 && binIndex < numBins) {
        binCounts[binIndex]++;
    }
}

// Display histogram
for (int i = 0; i < numBins; i++) {
    double binStart = minValue + i * binWidth;
    double binEnd = binStart + binWidth;
    System.out.printf( format: "%.2f - %.2f: %d\n",  args: binStart,  args: binEnd, binCounts[i]);
}
```

Draw boxplot and make comment.

```
// Create a JFrame to hold the boxplot
JFrame frame = new JFrame( title: "Box Plot");
frame.setDefaultCloseOperation( operation: JFrame.EXIT_ON_CLOSE);
frame.setSize( width: 400,  height: 400);

// Create a JPanel to draw the boxplot
JPanel panel = new JPanel() {
    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);

        // Calculate boxplot dimensions
        int x = 50; // X-coordinate of the boxplot
        int y = 50; // Y-coordinate of the boxplot
        int width = 300; // Width of the boxplot
        int height = 300; // Height of the boxplot

        // Calculate quartiles and interquartile range
        double q1 = getQuartile(data,  quartile: 1);
        double q3 = getQuartile(data,  quartile: 3);
        double iqr = q3 - q1;

        // Calculate whisker positions
        double lowerWhisker = q1 - 1.5 * iqr;
        double upperWhisker = q3 + 1.5 * iqr;

        // Draw the box
        g.setColor( c: Color.BLACK);
        g.drawRect(x, y + height / 4, width, height / 2);
        g.drawLine( x1: x, y + height / 2, x + width, y + height / 2);

        // Draw the whiskers
        g.drawLine( x1: x, y + height / 2,  x2: x, y + height / 4);
        g.drawLine(x + width, y + height / 2, x + width, y + height / 4);
```

```
        // Draw the median line
        double median = getQuartile(data,  quartile: 2);
        int medianX = x;
        int medianY = (int) (y + height / 2 - ((median - lowerWhisker) / (upperWhisker - lowerWhisker)) * height / 2);
        g.drawLine( x1: medianX,  y1: medianY, medianX + width,  y2: medianY);

        // Draw the outliers
        for (double value : data) {
            if (value < lowerWhisker || value > upperWhisker) {
                int outlierX = (int) (x + width / 2);
                int outlierY = (int) (y + height / 2 - ((value - lowerWhisker) / (upperWhisker - lowerWhisker)) * height / 2);
                g.setColor( c: Color.RED);
                g.drawOval(outlierX - 3, outlierY - 3,  width: 6,  height: 6);
            }
        }
    }
};

// Set the layout manager to null for custom positioning
panel.setLayout( mgr: null);

// Set the bounds of the panel
panel.setBounds( x: 0,  y: 0,  width: frame.getWidth(),  height: frame.getHeight());

// Add the panel to the frame
frame.add( comp: panel);

// Make the frame visible
frame.setVisible( b: true);
}
```

Take specific number of sample and construct %95 confidence interval for the mean and variance.

```
double criticalValue = 1.96;

int sample = 5;

double meanMarginOfError = criticalValue * Math.sqrt(variance / sample);

System.out.println("%95 Confidence Interval: " + meanMarginOfError);
```

How large a sample for your data should be collected to estimate the population mean with a margin at most 0.1 units with confidence 90%.

```java
public static int calculateSampleSize(List<Double> data, double marginOfError, double confidenceLevel) {
    double zScore = getZScore(confidenceLevel);
    double estimatedPopulationStdDev = calculatePopulationStdDev(data);
    double sampleSize = customPow((zScore * estimatedPopulationStdDev) / marginOfError, exponent: 2);
    return (int) customCeil( value: sampleSize);
}

public static double getZScore(double confidenceLevel) {
    double[] confidenceLevels = {0.8, 0.85, 0.9, 0.95, 0.975, 0.99, 0.995};
    double[] zScores = {1.2816, 1.4401, 1.645, 1.9599, 2.2414, 2.5758, 2.807};

    // Find the closest confidence level in the lookup table
    int closestIndex = 0;
    double minDifference = customAbs(confidenceLevels[0] - confidenceLevel);
    for (int i = 1; i < confidenceLevels.length; i++) {
        double difference = customAbs(confidenceLevels[i] - confidenceLevel);
        if (difference < minDifference) {
            minDifference = difference;
            closestIndex = i;
        }
    }

    return zScores[closestIndex];
}
```

Output:

```
Mean: 8.094225604297225
Median: 7.0
Variance: 36.701991425081616
Standart Deviation: 6.058216874626333
Standart Error: 0.07400185437207733
Outliers: [21.0, 22.0, 23.0, 24.0, 25.0, 26.0, 27.0, 28.0, 29.0, 30.0, 31.0, 32.0, 33.0, 34.0]
%95 Confidence Interval: 5.31026120371858
Sample Size: 9934
Not Normally Distributed
Skewness: 0.9811926299746659
Kurtosis: 0.7677874112099521
Histogram:
0.00 - 3.40: 1885
3.40 - 6.80: 1361
6.80 - 10.20: 1351
10.20 - 13.60: 823
13.60 - 17.00: 624
17.00 - 20.40: 414
20.40 - 23.80: 126
23.80 - 27.20: 61
27.20 - 30.60: 34
30.60 - 34.00: 21
```