Problem

Consider the following learning set:

1	7	-2	4.5
2	3	-5	2.3
1	6	-2	1.2
2	5	-2	4.5
1	6	-5	2.3

Requirements:

a) Read the learning set from a file saved on your local drive (in.txt) The values on each line will be separated by space

b) Calculate and display the average of each feature (column) using the following formula:

$$\widetilde{x_i} = \frac{\sum_{j=1}^{k} f_j \cdot x_i^j}{\sum_{j=1}^{k} f_j}$$

i = *feature (column number)*

$\widetilde{x_i} = feature\ average$

$x_i^j = the\ j\ distinct\ value\ of\ feature\ i$, that appears f_j times in column i of the learning set

Example:

Consider the following feature:

4
6
6
2
2

The average will be: (1*4 + 2*6 + 2*2)/(1+2+2) = 4

Please note that $\sum_{j=1}^{k} f_j$ will always be equal with the number of elements from the feature

c) Add the following "weight" column to the initial matrix:

1	7	-2	4.5	1
2	3	-5	2.3	3
1	6	-2	1.2	2
2	5	-2	4.5	2
1	6	-5	2.3	1

d) Calculate and display the weighted average of the learning set) using the following formula:

$$\widetilde{x_i} = \frac{\sum_{j=1}^{k}(x_i^j \cdot \sum_{z=1}^{f_j} w_z^j)}{\sum_{i=1}^{n} w_i}$$

$\sum_{z=1}^{f_j} w_z^j)$ - sum of all the weights for each distinct element of the feature
n – total number of feature elements (correspond to the number of shapes from the initial matrix)

Example:
Consider the following feature with the corresponding weights (in red):

4	2
6	1
6	3
2	2
2	4

The weighted average will be: 4*2+6*(1+3)+2*(2+4)/(2+1+3+2+4) = ~3.66

If all the elements form a feature are different (f1 =f2 = …….. fn), we can use the following formula:

$$\widetilde{x_i} = \frac{\sum_{j=1}^{n}(x_i^j \cdot w_i^j)}{\sum_{i=1}^{n} w_i}$$

e) Calculate and display the frequency of occurrence for each distinct value j, using the following formula:

$$v_j = \frac{f_j}{n}$$

f) Calculate and display each feature dispersion using the following formula:

$$D(i) = \frac{1}{n-1}\sum_{k=1}^{n}(x_{ki} - \widetilde{x}_i)^2$$

g) Calculate and display the covariance between two any features using the following formula:

$$cov(i,j) = \frac{1}{n-1}\sum_{k=1}^{n}(x_{ki} - \widetilde{x}_i)(x_{kj} - \widetilde{x}_j)$$

h) Calculate and display the correlation coefficient between two any features using the following formula:

$$r(i,j) = \frac{cov(i,j)}{\sqrt{D(i)\cdot D(j)}}$$

i) Calculate and display the average square deviation for all the features using the following formula:

$$\sigma(i) = \sqrt{D(i)}$$

j) Autoscale the learning set using the following formula:

$$x_{ij} = \frac{x_{ij} - \widetilde{x}_j}{\sum_{k=1}^{n}(x_{jk} - \widetilde{x}_j)^2}$$

Steps & Hints

1. **Created separate methods for each calculation in order to be able to reuse the code**

2. **Use HashMaps to keep counters of elements occurrence**

3. **Use Java Streams to quickly iterate and sum elements of collections**

4. *Create StatisticsUtils class where you will implement the metods for calculating the above required values (the method for calculationg feature average is provided as example)*

```
package ro.usv.rf;

import java.util.HashMap;

public class StatisticsUtils
{

	protected static double calculateFeatureAverage(Double[] feature) {
		Map<Double, Integer> counterMap = getFeatureDistincElementsCounterMap(feature);
		double featureAverage = 0;

		double sum1 = 0;
		double sum2 = 0;

		sum1 = counterMap.keySet().stream()
				.mapToDouble(x -> calculateSum1(x, counterMap.get(x))).sum();
		sum2 = counterMap.values().stream()
				.mapToInt(x -> x).sum();
		featureAverage = sum1 / sum2;
		System.out.println("The feature average is: " +  featureAverage);
		return featureAverage;
}

	protected static Map<Double, Integer> getFeatureDistincElementsCounterMap(Double feature[])
	{
		Map<Double, Integer> counterMap = new HashMap<Double, Integer>();
		for (int j = 0; j < feature.length; j++) {
			if (counterMap.containsKey(feature[j])) {
				int count = counterMap.get(feature[j]);
				counterMap.put((feature[j]), ++count);
			} else {
				counterMap.put((feature[j]), 1);
			}
		}
		return counterMap;
	}

	private static Double calculateSum1(double value, int count)
	{
		return count * value;
	}

	protected static double calculateFeatureWeightedAverage(Double[] feature, Double[] weights) {
		double featureWeightedAverage = 0.0;
		// your code here
		return featureWeightedAverage;
	}

	protected static double calculateFrequencyOfOccurence(Map<Double, Integer> counterMap, double
featureElement) {
		double frequencyOfOccurence = 0.0;
		// your code here
		return frequencyOfOccurence;
	}

	protected static double calculateFeatureDispersion(Double[] feature, double
featureWeightedAverage) {
		double featureDispersion = 0.0;
		// your code here
		return featureDispersion;
	}
```

```
        protected static double calculateCovariance (Double[] feature1, Double[] feature2,
                        double feature1WeightedAverage,double feature2WeightedAverage) {
                double covariance = 0.0;
                // your code here
                return covariance;
        }

        protected static double calculateCorrelationCoefficient  (double covariance, double
feature1Dispersion,
                        double feature2Dispersion ) {
                double correlationCoefficient = 0.0;
                // your code here
                return correlationCoefficient;
        }

        protected static double calculateAverageSquareDeviation (double featureDispersion ) {
                double averageSquareDeviation = 0.0;
                // your code here
                return averageSquareDeviation;
        }
}
```

Note1) in the above example, in `calculateFeatureAverage()`, counterMap keeps the a mapping between x_i^j and fj values from the formula. Basically, it will tell us how many times each distinct element appears

Note2) The above example can be improved in terms of performance. Can you find what can be improved ?

MainClass

```
package ro.usv.rf;

import java.util.ArrayList;
import java.util.List;
import java.util.Map;

public class MainClass {


        public static void main(String[] args) {
                double[][] learningSet = FileUtils.readLearningSetFromFile("e:\\in.txt");

                int numberOfForms = learningSet.length;
                int numberOfFeatures = learningSet[0].length;

                //calculate average for each feature
                for (int featureIndex = 0; featureIndex < numberOfFeatures; featureIndex++)
                {
                        Double[] feature = new Double[numberOfForms];
                        for (int formIndex = 0; formIndex < numberOfForms; formIndex++)
                        {
                                feature[formIndex] = learningSet[formIndex][featureIndex];
                        }
                        double featureAverage = StatisticsUtils.calculateFeatureAverage(feature);
                        System.out.println("Feature average is : " + featureAverage);
                }
```

```
                //get weights from matrix - only after you add the new line !!!
                // since the last column represent the weights, the number of features is now
learningSet[0].length - 1
                numberOfFeatures = learningSet[0].length - 1;


                Double[] weights = new Double[numberOfForms];
                for (int formIndex = 0; formIndex < numberOfForms; formIndex++)
                {
                        // weights are always located on the last  column
                        int weightColumnIndex = learningSet[formIndex].length - 1;
                        weights[formIndex] = learningSet[formIndex][weightColumnIndex];
                }


                // calculate Feature Weighted Average and feature dispersion
                List<Double[]> featureList = new ArrayList<Double[]>();
                double[] featureWeightedAverages = new double[numberOfFeatures];
                double[] featureDispersions = new double[numberOfFeatures];
                double[] averageSquareDeviations = new double[numberOfFeatures];

                for (int featureIndex = 0; featureIndex < numberOfFeatures; featureIndex++)
                {

                        Double[] feature = new Double[numberOfForms];
                        for (int formIndex = 0; formIndex < numberOfForms; formIndex++)
                        {
                                feature[formIndex] = learningSet[formIndex][featureIndex];
                        }
                        featureList.add(feature);

                        featureWeightedAverages[featureIndex]=
StatisticsUtils.calculateFeatureWeightedAverage(feature,weights);
                        System.out.println("Feature weighted average is : " +
featureWeightedAverages[featureIndex]);

                        featureDispersions[featureIndex] =
StatisticsUtils.calculateFeatureDispersion(feature, featureWeightedAverages[featureIndex]);
                        System.out.println("Feature dispersion is : " +
featureDispersions[featureIndex]);

                        averageSquareDeviations[featureIndex] =
StatisticsUtils.calculateAverageSquareDeviation(featureDispersions[featureIndex]);
                        System.out.println("average Square Deviations is : " +
averageSquareDeviations[featureIndex]);
                }

                //we select a feature and an element to calculate frequency of occurence

                int featureIndex = 0;
                int elementIndex = 2;
                Map<Double, Integer> featureDistincElementsCounterMap =

        StatisticsUtils.getFeatureDistincElementsCounterMap(featureList.get(featureIndex));
                double frequencyOfOccurence =
StatisticsUtils.calculateFrequencyOfOccurence(featureDistincElementsCounterMap,
learningSet[elementIndex][featureIndex]);
                System.out.println("frequency Of Occurence is : " + frequencyOfOccurence);


                //we select two features to calculate covariance and corelation
                int feature1Index = 0;
                int feature2Index = 1;

                double covariance = StatisticsUtils.calculateCovariance(featureList.get(feature1Index),
featureList.get(feature2Index),
                                featureWeightedAverages[feature1Index],  featureWeightedAverages[1]);
                System.out.println("covariance is : " + covariance);
```

```
                double corelation = StatisticsUtils.calculateCorrelationCoefficient(covariance,
featureDispersions[feature1Index], featureDispersions[feature2Index]);
                System.out.println("corelation is : " + corelation);


                FileUtils.writeLearningSetToFile("e:\\scaledSet.csv", autoscaleLearningSet(learningSet));



        }

        private static double[][] autoscaleLearningSet (double[][] learningSet)
        {
                double[][] autoscaledLearningSet = new double[learningSet.length][];
                //.. your code here
                return autoscaledLearningSet;
        }


}
```

Discussion points:

Java maps

Lambda expressions

Writing Java methods
