BIRZEIT UNIVERSITY

Faculty of Engineering and Information Technology

Computer Science Department

COMP338 Project #3 Report

Students name and id :

Iman Salameh 1201786

Ashraf Mtor 1183389

Instructure name : Dr.Mohammed Hellal

Date :28/01/2024

Contents

summary

The Wika Knowledge Analysis Environment, or Weka, is a well-known suite of Java-based machine learning applications created at the University of Wika in New Zealand. It is free to download and is an open source project. Here is a summary of its basic features and functions:

ML Algorithms: Weka offers a wide range of algorithms for machine learning. Many tasks, including association rule extraction, clustering, regression, and classification, can be accomplished using these techniques. The way the algorithms are set up allows users of all experience levels to easily access them.

Data pre-processing: Data pre-processing is often necessary before machine learning algorithms can be implemented. Weka provides tools to perform operations including processing, normalization, transformation, and feature selection.

Data pre-processing: Data pre-processing is often necessary before machine learning algorithms can be implemented. Weka provides tools to help with feature selection, dealing with missing values, transformation, and normalization. This helps in preparing datasets efficiently for analysis.

(GUI): Weka's user-friendly graphic user interface is one of its advantages. One of the core elements of this interface is the Explorer, which provides a straightforward and easy-to-use way to load data, use algorithms, and visualize results. For this reason, even those who do not have any programming skills can use it.

Compatibility with multiple data formats: Weka is able to handle data in CSV, native ARFF (Attribute Relationship File Format), and databases via JDBC.

Visualization Tools: Weka comes with the necessary data visualization tools to decipher analytics results and understand data.

Classes

Controller

```
package ML;

import java.io.File;
import java.io.IOException;

import weka.core.Instances;
import weka.core.converters.ArffSaver;
import weka.core.converters.CSVLoader;

public class Controller {
	public static void main(String[] args) {
		ML();
	}

	public static void initiateConversion() {
		try {
			loadCSVAndProcess();
		} catch (IOException e) {
			e.printStackTrace();
		}
		System.out.println("Conversion Successful");
	}

	private static void loadCSVAndProcess() throws IOException {
		CSVLoader csvLoader = new CSVLoader();
		setCSVSourcePath(csvLoader);
		verifyCSVStructure(csvLoader);
		Instances dataset = csvLoader.getDataSet();
		processDataset(dataset);
	}

	private static void processDataset(Instances dataset) throws
IOException {
		if (dataset != null) {
			for (int i = 0; i < dataset.numInstances(); i++) {
				updateDatasetValues(dataset, i);
			}
			saveAsARFF(dataset);
		} else {
			System.err.println("CSV Data Loading Failed.");
		}
	}
```

```
	private static void updateDatasetValues(Instances dataset, int index)
throws IOException {
		String gender = dataset.instance(index).stringValue(0);
		double genderNumeric = gender.equalsIgnoreCase("Male") ? 1.0 :
0.0;
		dataset.instance(index).setValue(0, genderNumeric);
		double height = dataset.instance(index).value(1);
		dataset.instance(index).setValue(1, height * 2.54);
		double weight = dataset.instance(index).value(2);
		dataset.instance(index).setValue(2, weight * 0.453592);
	}

	private static void saveAsARFF(Instances dataset) throws IOException {
		ArffSaver arffSaver = new ArffSaver();
		arffSaver.setInstances(dataset);
		arffSaver.setFile(new File("C:\\Users\\yazan\\eclipse-
workspace\\weka\\src\\output.arff"));
		arffSaver.writeBatch();
	}

	private static void setCSVSourcePath(CSVLoader csvLoader) throws
IOException {
		csvLoader.setSource(new File("C:\\Users\\yazan\\eclipse-
workspace\\weka\\src\\Height_Weight.csv"));
	}

	private static void verifyCSVStructure(CSVLoader csvLoader) throws
IOException {
		if (csvLoader.getStructure() == null) {
			throw new IOException("Invalid CSV Structure.");
		}
	}

		private static void ML() {
		initiateConversion();
		DataStatistics.calculateAndDisplayStats();
		Module_ML1.performLinearRegression();
		Module_ML2.executeLinearRegression();
		Module_ML3.runLinearRegressionAnalysis();
		Module_ML4.executeLinearRegression();
	}
}
```

DataStatistics

```
package ML;

import weka.core.Instances;
import weka.core.converters.ConverterUtils.DataSource;

import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;

import weka.core.AttributeStats;

public class DataStatistics {

	public static void calculateAndDisplayStats() {
		try {
			DataSource dataSource = initializeDataSource();
			Instances dataset = dataSource.getDataSet();
			AttributeStats heightStatistics =
dataset.attributeStats(1);
			AttributeStats weightStatistics =
dataset.attributeStats(2);

			printTableHeader();

			displayStatistics("Height", heightStatistics, dataset, 1);
			displayStatistics("Weight", weightStatistics, dataset, 2);
		} catch (Exception e) {
			e.printStackTrace();
		}
	}

	public static void printDataToFile(String filePath, String dataToPrint)
{
		try {
			BufferedWriter writer = new BufferedWriter(new
FileWriter(filePath));
			writer.write(dataToPrint);
			writer.close();
			System.out.println("Data has been successfully written to
the file: " + filePath);
		} catch (IOException e) {
			e.printStackTrace();
		}
	}

	private static DataSource initializeDataSource() throws Exception {
```

```
            return new DataSource("C:\\Users\\yazan\\eclipse-
workspace\\weka\\src\\Height_Weight.arff");
      }

      private static void printTableHeader() {
            String outerBorder =
"+=================================================================================
======+";
            String innerBorder = "|----------------+------+-------+------+---
-----------------+--------|";

            System.out.println(outerBorder);
            System.out.printf("| %-14s | %-4s | %-5s | %-4s | %-19s | %-6s
|\n", "Attribute", "Min", "Max", "Mean",
                        "Standard Deviation", "Median");
            System.out.println(innerBorder);
            System.out.println(outerBorder);
      }

      private static void displayStatistics(String attributeName,
AttributeStats stats, Instances data,
                  int attributeIndex) {
            // Table Header (if not already printed in another method)
            String headerBorder = "+-----------+------+-------+------+-------
-------------+--------+";
            String header = "| Attribute | Min  | Max   | Mean | Standard
Deviation | Median |";
            String separator = "|-----------|------|-------|------|----------
----------|--------|";

            System.out.println(headerBorder);
            System.out.println(header);
            System.out.println(separator);

            System.out.printf("| %-9s | %-4.2f | %-5.2f | %-4.2f | %-18.2f |
%-6.2f |\n", attributeName,
                        stats.numericStats.min, stats.numericStats.max,
stats.numericStats.mean, stats.numericStats.stdDev,
                        calculateMedian(data, attributeIndex));

            System.out.println(headerBorder);
      }

      private static double calculateMedian(Instances data, int
attributeIndex) {
            int dataSize = data.numInstances();
            if (dataSize % 2 == 0) {
                  double midValue1 = data.instance(dataSize / 2 -
1).value(attributeIndex);
                  double midValue2 = data.instance(dataSize /
2).value(attributeIndex);
                  return (midValue1 + midValue2) / 2.0;
            } else {
                  return data.instance(dataSize / 2).value(attributeIndex);
            }
      }
}
```

ML1

```
package ML;

import weka.core.Instances;
import weka.core.converters.ConverterUtils.DataSource;
import weka.classifiers.functions.LinearRegression;

import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;

import weka.classifiers.Evaluation;
import weka.filters.Filter;
import weka.filters.unsupervised.instance.Randomize;

public class Module_ML1 {
	public static void performLinearRegression() {
		try {
			Instances data = loadData("C:\\Users\\yazan\\eclipse-
workspace\\weka\\src\\Height_Weight.arff");
			if (data != null) {
				data = prepareData(data);
				Instances[] splitData = splitData(data);
				Instances trainData = splitData[0];
				Instances testData = splitData[1];
				LinearRegression model = buildModel(trainData);
				evaluateModel(model, trainData, testData);
			} else {
				System.err.println("Failed to load data from ARFF.");
			}
		} catch (Exception e) {
			e.printStackTrace();
		}
	}

	private static Instances loadData(String filePath) throws Exception {
		DataSource source = new DataSource(filePath);
		return source.getDataSet();
	}

	private static Instances prepareData(Instances data) throws Exception {
		data.setClassIndex(data.numAttributes() - 1);
		Randomize randomize = new Randomize();
		randomize.setInputFormat(data);
		return Filter.useFilter(data, randomize);
	}

	private static Instances[] splitData(Instances data) {
```

```
            int instancesLimit = 100;
            Instances limitedData = new Instances(data, 0, instancesLimit);
            int trainSize = (int) Math.round(limitedData.numInstances() *
0.7);
            int testSize = limitedData.numInstances() - trainSize;
            Instances trainData = new Instances(limitedData, 0, trainSize);
            Instances testData = new Instances(limitedData, trainSize,
testSize);
            return new Instances[] { trainData, testData };
      }

      private static LinearRegression buildModel(Instances trainData) throws
Exception {
            LinearRegression model = new LinearRegression();
            model.buildClassifier(trainData);
            return model;
      }

      private static void evaluateModel(LinearRegression model, Instances
trainData, Instances testData)
                  throws Exception {
            Evaluation eval = new Evaluation(trainData);
            eval.evaluateModel(model, testData);
            printEvaluationTable(eval);
      }

      public static void printDataToFile(String filePath, String dataToPrint)
{
            try {
                  BufferedWriter writer = new BufferedWriter(new
FileWriter(filePath));
                  writer.write(dataToPrint);
                  writer.close();
                  System.out.println("Data has been successfully written to
the file: " + filePath);
            } catch (IOException e) {
                  e.printStackTrace();
            }
      }

      private static void printEvaluationTable(Evaluation eval) {
            String headerBorder = "+-----------------------------+------------
-----+";
            String dataBorder = "+-----------------------------+--------------
---+";

            System.out.println(headerBorder);
            System.out.printf("| %-25s | %-16s |\n", "Metric", "Value");
            System.out.println(headerBorder);

            System.out.printf("| %-25s | %-16.4f |\n", "Mean Absolute Error",
eval.meanAbsoluteError());
            System.out.printf("| %-25s | %-16.4f |\n", "Root Mean Squared
Error", eval.rootMeanSquaredError());

            System.out.println(dataBorder);
      }
```

```
}
```

ML2

```
package ML;

import weka.core.Instances;
import weka.core.converters.ConverterUtils.DataSource;
import weka.classifiers.functions.LinearRegression;
import weka.classifiers.Evaluation;
import weka.filters.Filter;
import weka.filters.unsupervised.instance.Randomize;

public class Module_ML2 {

	public static void executeLinearRegression() {
		try {
			Instances dataset =
fetchDataSet("C:\\Users\\yazan\\eclipse-
workspace\\weka\\src\\Height_Weight.arff");
			if (dataset != null) {
				Instances processedData =
processDataForAnalysis(dataset);
				Instances[] dividedData = divideData(processedData);
				Instances trainingSet = dividedData[0];
				Instances testingSet = dividedData[1];
				LinearRegression lrModel =
trainLinearRegressionModel(trainingSet);
				performEvaluation(lrModel, trainingSet, testingSet);
			} else {
				System.err.println("Data loading from ARFF file
failed.");
			}
		} catch (Exception e) {
			e.printStackTrace();
		}
	}

	private static Instances fetchDataSet(String path) throws Exception {
		DataSource dataSource = new DataSource(path);
		return dataSource.getDataSet();
	}

	private static Instances processDataForAnalysis(Instances originalData)
throws Exception {
		originalData.setClassIndex(originalData.numAttributes() - 1);
		Randomize randomizer = new Randomize();
		randomizer.setInputFormat(originalData);
		return Filter.useFilter(originalData, randomizer);
	}

	private static Instances[] divideData(Instances preparedData) {
```

```
            int sizeLimit = 500;
            Instances boundedData = new Instances(preparedData, 0,
sizeLimit);
            int trainingSize = (int) Math.round(boundedData.numInstances() *
0.7);
            Instances trainingData = new Instances(boundedData, 0,
trainingSize);
            Instances testingData = new Instances(boundedData, trainingSize,
boundedData.numInstances() - trainingSize);
            return new Instances[] { trainingData, testingData };
      }

      private static LinearRegression trainLinearRegressionModel(Instances
trainData) throws Exception {
            LinearRegression regressionModel = new LinearRegression();
            regressionModel.buildClassifier(trainData);
            return regressionModel;
      }

      private static void performEvaluation(LinearRegression model, Instances
trainData, Instances testData)
                  throws Exception {
            Evaluation evaluation = new Evaluation(trainData);
            evaluation.evaluateModel(model, testData);
            printEvaluationTable(evaluation);
      }

      private static void printEvaluationTable(Evaluation eval) {
            String headerBorder = "+---------------------------+-------------
-----+";
            String dataBorder = "+---------------------------+---------------
---+";

            System.out.println(headerBorder);
            System.out.printf("| %-25s | %-16s |\n", "Metric", "Value");
            System.out.println(headerBorder);

            System.out.printf("| %-25s | %-16.4f |\n", "Mean Absolute Error",
eval.meanAbsoluteError());
            System.out.printf("| %-25s | %-16.4f |\n", "Root Mean Squared
Error", eval.rootMeanSquaredError());

            System.out.println(dataBorder);
      }

}
```

ML3

```
package ML;

import weka.core.Instances;
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;

import weka.core.converters.ConverterUtils.DataSource;
import weka.classifiers.functions.LinearRegression;
import weka.classifiers.Evaluation;
import weka.filters.Filter;
import weka.filters.unsupervised.instance.Randomize;

public class Module ML3 {

	public static void runLinearRegressionAnalysis() {
		try {
			Instances dataset = loadDataset("C:\\Users\\yazan\\eclipse-
workspace\\weka\\src\\Height_Weight.arff");
			if (dataset != null) {
				Instances randomizedData = randomizeData(dataset);
				Instances[] splitDatasets =
splitDataset(randomizedData, 5000, 0.7);
				Instances trainingDataset = splitDatasets[0];
				Instances testingDataset = splitDatasets[1];

				LinearRegression regression =
createAndTrainModel(trainingDataset);
				evaluateRegressionModel(regression, trainingDataset,
testingDataset);
			} else {
				System.err.println("Error loading ARFF data.");
			}
		} catch (Exception e) {
			e.printStackTrace();
		}
	}

	private static Instances loadDataset(String filePath) throws Exception
{
		DataSource dataSource = new DataSource(filePath);
		return dataSource.getDataSet();
	}

	private static Instances randomizeData(Instances data) throws Exception
{
		data.setClassIndex(data.numAttributes() - 1);
```

```
            Randomize randomizeFilter = new Randomize();
            randomizeFilter.setInputFormat(data);
            return Filter.useFilter(data, randomizeFilter);
      }

      private static Instances[] splitDataset(Instances data, int limit,
double trainRatio) {
            Instances boundedData = new Instances(data, 0, limit);
            int trainSize = (int) Math.round(boundedData.numInstances() *
trainRatio);
            Instances trainData = new Instances(boundedData, 0, trainSize);
            Instances testData = new Instances(boundedData, trainSize,
boundedData.numInstances() - trainSize);
            return new Instances[] { trainData, testData };
      }

      private static LinearRegression createAndTrainModel(Instances
trainData) throws Exception {
            LinearRegression model = new LinearRegression();
            model.buildClassifier(trainData);
            return model;
      }

      public static void printDataToFile(String filePath, String dataToPrint)
{
            try {
                  BufferedWriter writer = new BufferedWriter(new
FileWriter(filePath));
                  writer.write(dataToPrint);
                  writer.close();
                  System.out.println("Data has been successfully written to
the file: " + filePath);
            } catch (IOException e) {
                  e.printStackTrace();
            }
      }

      private static void evaluateRegressionModel(LinearRegression model,
Instances trainData, Instances testData)
                  throws Exception {
            Evaluation evaluation = new Evaluation(trainData);
            evaluation.evaluateModel(model, testData);
            printEvaluationTable(evaluation);
      }

      private static void printEvaluationTable(Evaluation eval) {
            String headerBorder = "+---------------------------+------------
-----+";
            String dataBorder = "+---------------------------+--------------
---+";

            System.out.println(headerBorder);
            System.out.printf("| %-25s | %-16s |\n", "Metric", "Value");
            System.out.println(headerBorder);

            System.out.printf("| %-25s | %-16.4f |\n", "Mean Absolute Error",
eval.meanAbsoluteError());
```

```
            System.out.printf("| %-25s | %-16.4f |\n", "Root Mean Squared
Error", eval.rootMeanSquaredError());

            System.out.println(dataBorder);
      }

}
```

ML4

```
package ML;

import weka.core.Instances;
import weka.core.converters.ConverterUtils.DataSource;
import weka.classifiers.functions.LinearRegression;
import weka.classifiers.Evaluation;
import weka.filters.Filter;
import weka.filters.unsupervised.instance.Randomize;
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;

public class Module_ML4 {

      public static void executeLinearRegression() {
            try {
                  Instances dataset =
retrieveDataset("C:\\Users\\yazan\\eclipse-
workspace\\weka\\src\\Height_Weight.arff");
                  if (dataset != null) {
                        Instances shuffledData = shuffleDataset(dataset);
                        Instances[] trainingAndTestingSets =
partitionDataset(shuffledData, 0.7);
                        Instances trainingSet = trainingAndTestingSets[0];
                        Instances testingSet = trainingAndTestingSets[1];

                        LinearRegression trainedModel =
developRegressionModel(trainingSet);
                        analyzeModelPerformance(trainedModel, trainingSet,
testingSet);
                  } else {
                        System.err.println("ARFF  Loading Data ARFF Error.");
                  }
            } catch (Exception e) {
                  e.printStackTrace();
            }
      }

      public static void printDataToFile(String filePath, String dataToPrint)
{
            try {
                  BufferedWriter writer = new BufferedWriter(new
FileWriter(filePath));
                  writer.write(dataToPrint);
                  writer.close();
```

```
                System.out.println("Data has been successfully written to
the file: " + filePath);
            } catch (IOException e) {
                e.printStackTrace();
            }
      }

      private static Instances retrieveDataset(String filePath) throws
Exception {
            DataSource source = new DataSource(filePath);
            return source.getDataSet();
      }

      private static Instances shuffleDataset(Instances originalData) throws
Exception {
            originalData.setClassIndex(originalData.numAttributes() - 1);
            Randomize randomizationFilter = new Randomize();
            randomizationFilter.setInputFormat(originalData);
            return Filter.useFilter(originalData, randomizationFilter);
      }

      private static Instances[] partitionDataset(Instances data, double
trainingRatio) {
            int trainingSize = (int) Math.round(data.numInstances() *
trainingRatio);
            Instances trainDataSet = new Instances(data, 0, trainingSize);
            Instances testDataSet = new Instances(data, trainingSize,
data.numInstances() - trainingSize);
            return new Instances[] { trainDataSet, testDataSet };
      }

      private static LinearRegression developRegressionModel(Instances
trainingData) throws Exception {
            LinearRegression regression = new LinearRegression();
            regression.buildClassifier(trainingData);
            return regression;
      }

      private static void analyzeModelPerformance(LinearRegression model,
Instances trainData, Instances testData)
                  throws Exception {
            Evaluation modelEvaluation = new Evaluation(trainData);
            modelEvaluation.evaluateModel(model, testData);
            printEvaluationTable(modelEvaluation);
      }

      private static void printEvaluationTable(Evaluation eval) {
            String headerBorder = "+---------------------------+-----------
-----+";
            String dataBorder = "+---------------------------+-------------
---+";

            System.out.println(headerBorder);
            System.out.printf("| %-25s | %-16s |\n", "Metric", "Value");
            System.out.println(headerBorder);
```

```
            System.out.printf("| %-25s | %-16.4f |\n", "Mean Absolute Error",
eval.meanAbsoluteError());
            System.out.printf("| %-25s | %-16.4f |\n", "Root Mean Squared
Error", eval.rootMeanSquaredError());

            System.out.println(dataBorder);
      }

}
```

Description Classes

Controller class

It converts data from CSV format to ARFF format

The main functions :

initiateConversion: Load and process a CSV file.

loadCSVAndProcess: Creates a CSVLoader instance, sets the source path of the CSV file, checks the CSV structure, and then loads data into the instances object.

processDataset method: After loading the data, this method iterates through the data set, updating the values using the updateDatasetValues method.

updateDatasetValues: This method converts and manipulates specific data values. It converts gender from string to numeric format, and converts heights and weights to different units.

saveAsARFF method: ArffSaver is used to save the dataset of processed instances into an ARFF file. .

setCSVSourcePath: Sets the path of the CSV file to be loaded by CSVLoader.

CSVStructure: Checks if the CSV file has a valid structure before continuing to load data.

DataStatistics class

calculates and displays statistical information for specific attributes in a data set, calculating the minimum, maximum, mean, standard deviation, and mean values.

The main functions :

calculateAndDisplayStats: Initializes a data source, loads a dataset, and calculates statistics for specific attributes (such as height and weight).

printDataToFile: This method writes a specific string (dataToPrint) to a specific file path (filePath). .

initializeDataSource: This method prepares the data source from an ARFF file.

displayStatistics: This method takes the attribute name, its statistics, the dataset, and the attribute index to display the statistics for that particular attribute.

calculateMedian: This method calculates the average value of a specific attribute in the data set. .

Module_ML1 class

100 subset of the dataset

The main functions :

performLinearRegression: Organizes the linear regression process. It includes loading the data, preparing it, dividing it into training and test sets, building a linear regression model, and then evaluating the model's performance.

loadData: Loads data from a specified ARFF file.

prepareData: Applies a random filter to shuffle the data.

splitData: divides the data set into a 70% training set and a 30% test set for testing.

buildModel: Create a linear regression model using the training data set.

evaluateModel: Evaluate the performance of the model using the test data set.

Module_ML2 class

1000 subset of the dataset

Module_ML3 class

5000 subset of the dataset

Module_ML4 class

entire dataset

Descripe Requerment :

1. Convert Height and Weight Units

method updateDatasetValues in the class Controller:

```
	private static void updateDatasetValues(Instances dataset, int index)
throws IOException {
		String gender = dataset.instance(index).stringValue(0);
		double genderNumeric = gender.equalsIgnoreCase("Male") ? 1.0 :
0.0;
		dataset.instance(index).setValue(0, genderNumeric);
		double height = dataset.instance(index).value(1);
		dataset.instance(index).setValue(1, height * 2.54);
		double weight = dataset.instance(index).value(2);
		dataset.instance(index).setValue(2, weight * 0.453592);
	}
```

Explanation: This method iterates through each instance in the dataset, converts the length and weight values to the required units, and updates the dataset accordingly.

2. Print Main Statistics of Features

method calculateMedian in class DataStatistics

```
	private static double calculateMedian(Instances data, int
attributeIndex) {
		int dataSize = data.numInstances();
		if (dataSize % 2 == 0) {
			double midValue1 = data.instance(dataSize / 2 -
1).value(attributeIndex);
			double midValue2 = data.instance(dataSize /
2).value(attributeIndex);
			return (midValue1 + midValue2) / 2.0;
		} else {
			return data.instance(dataSize / 2).value(attributeIndex);
		}
	}
```

And this method to print :

```
      private static void displayStatistics(String attributeName,
AttributeStats stats, Instances data,
                  int attributeIndex) {
            String headerBorder = "+-----------+------+-------+------+-------
-------------+--------+";
            String header = "| Attribute | Min  | Max   | Mean | Standard
Deviation | Median |";
            String separator = "|-----------|------|-------|------|----------
----------|--------|";

            System.out.println(headerBorder);
            System.out.println(header);
            System.out.println(separator);

            System.out.printf("| %-9s | %-4.2f | %-5.2f | %-4.2f | %-18.2f |
%-6.2f |\n", attributeName,
                        stats.numericStats.min, stats.numericStats.max,
stats.numericStats.mean, stats.numericStats.stdDev,
                        calculateMedian(data, attributeIndex));

            System.out.println(headerBorder);
      }
```

3. Split Data into 70% Training and 30% Test &4,5,6,7 randomizes data

Methoud SplitData :

```
      private static Instances[] splitData(Instances data) {
            int instancesLimit = 100;
            Instances limitedData = new Instances(data, 0, instancesLimit);
            int trainSize = (int) Math.round(limitedData.numInstances() *
0.7);
            int testSize = limitedData.numInstances() - trainSize;
            Instances trainData = new Instances(limitedData, 0, trainSize);
            Instances testData = new Instances(limitedData, trainSize,
testSize);
            return new Instances[] { trainData, testData };
      }
```

Explanation: This method splits the dataset into two parts: 70% for training and 30% for testing, based on the total number of instances.

Module_ML1 : 100

Module_ML2 : 1000

Module_ML3 : 5000

Module_ML4 : entire dataset

And to randomizes the data

```
	private static Instances randomizeData(Instances data) throws Exception
{
		data.setClassIndex(data.numAttributes() - 1);
		Randomize randomizeFilter = new Randomize();
		randomizeFilter.setInputFormat(data);
		return Filter.useFilter(data, randomizeFilter);
	}
```

8-Print the appropriate performanceMatrics

```
+=============================================================================+
| Attribute      | Min   | Max    | Mean | Standard Deviation   | Median |
|----------------+-------+--------+------+----------------------+--------|
+=============================================================================+
+-----------+------+-------+------+--------------------+--------+
| Attribute | Min  | Max   | Mean | Standard Deviation | Median |
|-----------|------|-------|------|--------------------|--------|
| Height    | 137.83 | 200.66 | 168.57 | 9.77              | 164.16 |
+-----------+------+-------+------+--------------------+--------+
+-----------+------+-------+------+--------------------+--------+
| Attribute | Min  | Max   | Mean | Standard Deviation | Median |
|-----------|------|-------|------|--------------------|--------|
| Weight    | 29.35 | 122.47 | 73.23 | 14.56             | 68.26  |
+-----------+------+-------+------+--------------------+--------+
+---------------------------+------------------+
| Metric                    | Value            |
+---------------------------+------------------+
| Mean Absolute Error       | 4.7149           |
| Root Mean Squared Error   | 6.1928           |
+---------------------------+------------------+
+---------------------------+------------------+
| Metric                    | Value            |
+---------------------------+------------------+
| Mean Absolute Error       | 3.7451           |
| Root Mean Squared Error   | 4.6156           |
+---------------------------+------------------+
+---------------------------+------------------+
| Metric                    | Value            |
+---------------------------+------------------+
| Mean Absolute Error       | 3.6935           |
| Root Mean Squared Error   | 4.5651           |
+---------------------------+------------------+
+---------------------------+------------------+
| Metric                    | Value            |
+---------------------------+------------------+
| Mean Absolute Error       | 3.5703           |
| Root Mean Squared Error   | 4.5016           |
```

Conclusion

This project is based on data preprocessing and transformation, statistical analysis of data features, and development and evaluation of linear regression models.