ArrayLists and Data

Haverford CS 106 - Introduction to Data Structures

Lab 2 (one week)

1 Maven Setup

You should add the below to your Maven `pom.xml`; it allows us to use a library that makes reading data in from a CSV easier.

```
<dependency>
   <groupId>com.opencsv</groupId>
   <artifactId>opencsv</artifactId>
   <version>4.4</version>
</dependency>
```

2 Finishing Your Data Storage

In the previous lab you created a class to store a single row of the dataset. Now, you should add a class to store the full dataset:

1. Create a class that will hold the full CSV of data, using the class you made to store rows within it. Think carefully about what names, fields, constructors, getters, and setters you should create, and be sure to add comments.

2. Make a field in your dataset class that holds an ArrayList of rows.

Due to the way the OpenCSV library we're using allows us to read in data, it will also be useful to do the below:

1. Create a constructor for your row class that takes a String array as input (if you haven't already).

3 Reading in Data

OpenCSV is a library that allows you to read in data. You can read in data from "compas-scores.csv" using the below code (which you'll need to modify appropriately). You'll also need to add the appropriate imports - Eclipse can help you find them.

```
CSVReaderHeaderAware reader = new CSVReaderHeaderAware(
                                    new FileReader("yourfile.csv"));
ArrayList<String[]> myEntries = new ArrayList<String[]>(reader.readAll());
reader.close();
```

This will give you an ArrayList containing String arrays, where each entry in the String array is once cell from the CSV, and each String array is a row from the CSV.

To read in the data from the CSV:

1. Create a Main class with a main method and use this to read the data from the ArrayList of String arrays into the data structure you developed in the previous lab. Think carefully about how to add to your data structure design to do this.

2. Catch any thrown exceptions due to invalid data entries so that the data continues to be read in, skipping any rows with errors.

Answer the following questions in a block comment to the main function:

1. If any of your validation methods indicate that your precondition assumptions were not met by the data, document what preconditions were violated and in what way. Update your validation methods one error at a time, documenting all issues, until you can read in all the data. If this did not happen, mention that.

2. Describe a person who would generate data that would *not* pass your (updated) precondition assumptions.

4 Analysis

Now that you have the data read into your data structure we can reproduce the analysis ProPublica shows in their chart, "Prediction Fails Differently for Black Defendants."

1. Add or implement any methods that you need to replicate the ProPublica analysis that were missing from your previous lab. This should include method(s) added to your dataset class that perform the full analysis.

2. Have your main function call the relevant methods to calculate and print out the data from the chart. You should be able to check your work by making sure it matches the numbers in the ProPublica chart.

3. Find charges that, based on the `r_charge_desc`, you think should not count as recidivism. Change your analysis using the `two_year_recid` outcome indicator accordingly and rerun the analysis. Do this using an optionally run method so that you can still run the previous version of the analysis. Describe the choices you made and what the resulting analysis shows in the method's Javadoc comments.