John Raven

CSCI-3327-001 Probability and Applied Stats

In my Statistics.java file I have a total of 13 methods all doing various things. The first method that has been created is the convert method, what this will do is take in an Array and copy the elements in the array to an ArrayList. This was created to assist some of my later methods with taking Arrays as inputs opposed to ArrayLists. The next method is mean, this will take in an Array, add all the elements together, and divide them by the Array length; in other words, it will find the mean of the Array. Median is the next method, this will read through a sorted Array and find the middle element, if the Array is even, it will add the two middle numbers and then divide them by two. Next is the mode method, this will read through the array and keep count of any duplicate elements. If there is more than one duplicate element it will see which has a higher count and then switch the mode to the new higher count mode. The following method is my stdDevi (Standard Deviation) method, this will read in an Array and perform the standard deviation formula on the Array. Inside of this method I also call the mean method that was mentioned earlier to avoid needing to compute the average manually. The variance method which follows the stdDevi method does everything that the stdDevi method does, except without the last step of finding the square root of the variance, instead it returns the variance. The next method I created was a factorial method, all this does is create the factorial of a number so I can call it later in methods that require factorial to be used. Permutation is the next method in this file, it returns the permutation of two numbers by using the factorial method when necessary. The combination method returns the combination of two numbers also using the factorial method when necessary. The next method in the list is the union method, the union method uses the convert method to convert two Arrays into two ArrayLists, this is done so the list can expand and shrink easily when necessary. The method then returns the union of two lists, as it is reading through the lists it also checks for any duplicates that may be in the list. Next is the intersection method, like the union method, this uses the convert method to convert two Arrays into two ArrayLists. Once converted the method will return the intersection of two lists by checking if the second list contains the element that the first list is currently at. After confirming that the element is in the second list, it checks if the element was already added to the intersection list, if not it adds the element. The next method is the compliment method, this method takes in an Array, converts it to an ArrayList, then checks against a preset list to see if the preset list contains anything the current list does not. If the preset list does contain elements that are not in the list it will add them to the list. My final method is my result method, this method takes in two Arrays and two doubles, it will then apply these parameters to the following methods: mean, median, mode, stdDevi, variance, permutation, combination, union, intersection, and compliment. The Statistics\_Tester.java file creates a Statistics object, two Arrays and two doubles, then will pass them into the result method that is called from the Statistics.java file.

The CSV\_Writer.java file contains two methods, the first being CSV\_Writer and the second being writeData. The CSV\_Writer method simply creates the csv file called “FavoriteNumber.csv” and will output “Error occurred: CSV\_Writer” if the file cannot be created. The writeData method will write two columns into the “FavoriteNumbers.csv”, the first column will contain ID numbers, the second column will contain randomly generated “favorite numbers”. The CSV\_Writer\_Tester.java file simply creates a CSV\_Writer object, then calls upon the writeData method. Inside of the csv file you will find the ID and Favorite Numbers column, this csv file needed to be saved as an xlsx file to properly save the histogram once created.

The next file that I will touch on is the Monte\_Carlo.java file, this program simulates the Monte Hall Gameshow simulation. The Monte\_Carlo.java file contains three methods, dontChange, doChange, and run. The dontChange method takes in the parameter plays and will run as many times as plays specifies. During the run it will generate two random numbers, playerDoor and winDoor, these numbers will be 0-2. Using these numbers, it will see if the winDoor is equal to playerDoor, if it is it will add to the count and then later use the count to find the probability of winning without changing the door. The doChange method also takes in the same parameter plays and generates two random numbers for playerDoor and winDoor. The method then creates a variable called wrongDoor and sets it equal to whatever number winDoor is. This method then reads through a while loop, if wrongDoor is equal to winDoor or playerDoor it will randomly generate wrongDoor’s integer again until it is no longer equal to winDoor or playerDoor. Another variable is created called changeDoor, this will subtract the sum of playerDoor and wongDoor from three to then pass to an if statement. This if statement sees if changeDoor is equal to winDoor, if it is it will add one to the count then return the probability of winning if you change door every time.