Arnav Shirodkar

CMSC 201: Data Structures

Lab 1 Report – Benford’s Law (07/02/2020)

|  |  |  |
| --- | --- | --- |
| **First Digit** | **R7.txt** | **Percentage** |
| 1 | 700000 | 100.00% |
| 2 | 0 | 0.00% |
| 3 | 0 | 0.00% |
| 4 | 0 | 0.00% |
| 5 | 0 | 0.00% |
| 6 | 0 | 0.00% |
| 7 | 0 | 0.00% |
| 8 | 0 | 0.00% |
| 9 | 0 | 0.00% |

**Benford’s Law - Findings**

After first piping the data in the file “heights.txt” through “Benford.java”, I retrieved a new file “startDigits.txt” and piped the values into “Hist.java”. I did the same for the data in “r5.txt” and “r7.txt” – photos of all the results are included on the right.

|  |  |  |
| --- | --- | --- |
| **First Digit** | **R5.txt** | **Percentage** |
| 1 | 116627 | 16.66% |
| 2 | 19525 | 2.79% |
| 3 | 39090 | 5.58% |
| 4 | 58363 | 8.34% |
| 5 | 77738 | 11.11% |
| 6 | 96386 | 13.77% |
| 7 | 116633 | 16.66% |
| 8 | 97928 | 13.99% |
| 9 | 77710 | 11.10% |

|  |  |  |
| --- | --- | --- |
| **First Digit** | **startDigits.txt** | **Percentage** |
| 1 | 24 | 47.06% |
| 2 | 4 | 7.84% |
| 3 | 8 | 15.69% |
| 4 | 6 | 11.76% |
| 5 | 1 | 1.96% |
| 6 | 3 | 5.88% |
| 7 | 3 | 5.88% |
| 8 | 2 | 3.92% |
| 9 | 0 | 0.00% |

Benford’s law states that in sets of naturally occuring numerical data, the first significant figure is likely to be small, with the digit 1 appearing at least 30% of the time and the successive numbers appearing less and less frequently.

This trend seems to generally hold true for the data from the heights.txt, as we can observe a general trend of decreasing occurrence from 1 – 9. However, this does not seem to hold true for the values in r5.txt and those in r7.txt. In the case of r7.txt, we have a unique situation where all the numbers are written in scientific notation and all have 1 as their first digit (r7.txt may not contain a “naturally occuring set of numbers”).

|  |  |  |
| --- | --- | --- |
| **First Digit** | **Primes.txt** | **Percentage** |
| 1 | 25 | 14.88% |
| 2 | 19 | 11.31% |
| 3 | 19 | 11.31% |
| 4 | 20 | 11.90% |
| 5 | 17 | 10.12% |
| 6 | 18 | 10.71% |
| 7 | 18 | 10.71% |
| 8 | 17 | 10.12% |
| 9 | 15 | 8.93% |

I then explored a data set of my own, running the same set of operations on all the prime numbers within the range of 1-1000. Benford’s law seems to generally hold true in this case as we can see the the occurrence of each digit has a generally decreasing trend.

**Stats – Findings**

Stats from r5.txt

N min max mean Standard Deviation

700000 2.000 12.000 7.000 2.417

Stats from r7.txt

(BONUS – The first method provided for calculating Std.Dev does not work with r7.txt and Welford’s method must be used – refer to Statfinder.java for reworked code).

N min max mean Standard Deviation

700000 10000000.000 10000001.000 10000000.500 0.289

Stats from heights.txt

N min max mean Standard Deviation

51 75.000 829.800 265.172 124.084

Benford.java

import edu.princeton.cs.algs4.\*;

public class Benford{

public static void main(String[] args){

while(!StdIn.isEmpty()){

String x = StdIn.readString();

char startDigit = x.charAt(0);

StdOut.println(startDigit);

}

}

}

Stats.java

import edu.princeton.cs.algs4.\*;

public class Stats{

public static void main (String[] args){

double count = 0; //number of input values

double sum = 0.0; // sum of input values

double sum\_sqs = 0.0; //variance\*(n-1)

//Initialise Min and Max as the first values read

double min=StdIn.readDouble(); //Note: Reads the first input, next call of StdIn reads the second input onward!

double max = min;

double first = min;

//Read and compute various stats

while (!StdIn.isEmpty()){

//Average Standard Deviation Stats

double temp = StdIn.readDouble(); //Note: Begins reading from 2nd value!

sum += temp;

sum\_sqs += (temp\*temp);

count++;

//Max and Min

if(temp > max){

max = temp;

}

if(temp < min){

min = temp;

}

}

//Account for 1st value not included

sum = sum + first;

count++;

//calculate stats

double average = sum/count;

double variance\_sqs = (sum\_sqs - ((sum\*sum)/count))/count;

double variance = Math.sqrt(variance\_sqs);

//convert to Scientific Notation or Not

StdOut.printf("%10s %10s %10s %10s %23s","N","min","max","mean","Standard Deviation");

StdOut.println();

StdOut.println("--------------------------------------------------------------------");

System.out.format("%10.0f %10.3f %10.3f %10.3f %10.3f ",count,min,max,average,variance);

StdOut.println();

}

}

Statfinder. Java

import edu.princeton.cs.algs4.\*;

public class Statfinder{

//must make these static to be referenced later

static double count; //number of input values

static double average; // mean value

static double x; //variance \* (n-1)

static final double firstInput = StdIn.readDouble(); //Note: Reads the first input, next call of StdIn reads the second input onward!

public Statfinder(){}

public double mean(){

return average;

}

public double stdDev(){

return Math.sqrt(x/(count-1));

}

public double n(){

return count;

}

public static void main (String[] args){

Statfinder calc = new Statfinder();

double min = firstInput;

double max = firstInput;

count = 1; //account for first number

average = firstInput/count; // starting average (which = to first)

while (!StdIn.isEmpty()){

double input = StdIn.readDouble(); //Note: Begins reading from 2nd value!

count++;

double temp = input - average; //Welford's method

average += temp/count;

x += (count-1)/count\*temp\*temp;

if(input>max){

max=input;

}

if(input<min){

min=input;

}

}

StdOut.printf("%10s %10s %10s %10s %23s","N","min","max","mean","Standard Deviation");

StdOut.println();

StdOut.println("--------------------------------------------------------------------");

System.out.format("%10.0f %10.3f %10.3f %10.3f %10.3f ",calc.n(),min,max,calc.mean(),calc.stdDev());

StdOut.println();

}

}