Code Documentation for StatsLibrary Project

By Rachel Hussmann

Overview

The purpose of the StatsLibrary project was to create classes and methods that calculate statistical operations that we learned in class. This project has twelve classes, each that manage a separate subset of statistical operations.

How It Works

BinomialDistribution

The BinomialDistribution class is responsible for containing the methods that calculate probabilities based on a binomial distribution. This class has nine methods:

- binomialDistribution
    - Parameters: int numberOfTrials – The number of trials to be run, double probOfSuccess – The probability of a successful trial, int totalNumSuccess – The total number of successes
    - Functionality – Calculates the probability of an event based off of a binomial distribution formula
    - Returns: double – The probability of the event occurring
- binomialDistribution
    - Parameters: BigInteger numberOfTrials – The number of trials to be run, BigDecimal probOfSuccess – The probability of success, BigInteger totalNumSuccess – The total number of successes
    - Functionality – Calculates the probability of an event based off of a binomial distribution formula
    - Returns: BigDecimal – The probability of the event occurring
- expectedValue
    - Parameters: int numberOfTrials – The total number of trials, double probOfSuccess – The probability of a successful trial
    - Functionality – Calculates the mean of a binomial distribution
    - Returns: double – The expected value (mean) of the binomial distribution
- expectedValue
    - Parameters: BigInteger numberOfTrials– The total number of trials, BigDecimal probOfSuccess – The probability of a successful trial
    - Functionality – Calculates the mean of a binomial distribution

    - Returns: BigDecimal – The expected value (mean) of the binomial distribution
- variance
    - Parameters: int numberOfTrials – The total number of trials, double probOfSuccess – The probability of success for a trial
    - Functionality – Calculates the variance of a binomial distribution
    - Returns: double – The variance of the binomial distribution
- variance
    - Parameters: BigInteger numberOfTrials – The total number of trials, BigDecimal probOfSuccess – The probability of success for a trial
    - Functionality – Calculates the variance of a binomial distribution
    - Returns: BigDecimal – The variance of the binomial distribution
- standardDeviation
    - Parameter: double variance – The variance of the binomial distribution
    - Functionality – Calculates the standard deviation of the binomial distribution
    - Returns: double – The standard deviation of the binomial distribution
- standardDeviation
    - Parameter: BigDecimal variance – The variance of the binomial distribution
    - Functionality – Calculates the standard deviation of the binomial distribution
    - Returns: BigDecimal – The standard deviation of
- testerOutput
    - Parameters: None
    - Functionality – Prints out examples of all methods
    - Returns: Nothing, but prints out statements

Combination

The Combination class is responsible for containing the methods that calculate the number of combinations of objects. This class has three methods:

- getCombination
    - Parameters: int n – The size of the set to choose from, int r – The number of choices to be made from the set
    - Functionality – Uses the combination formula and integers to calculate the number of combinations that are possible
    - Returns: int – The total number of combinations that can be made from n and r
- getCombination
    - Parameters: BigInteger n – The size of the set to choose from, BigInteger r – The number of choices to be made from the set

    - Functionality – Uses the combination formula and BigInteger objects to calculate the number of combinations that are possible
    - Returns: BigInteger – The total number of combinations that can be made from n and r
- testerOutput
    - Parameters: None
    - Functionality – Prints out examples of all methods
    - Returns: Nothing, but prints out statements

Factorial

The Factorial class is responsible for containing the methods that calculate factorials. This class has two methods:

- factorial
    - Parameters: int n – The number to find the factorial of
    - Functionality – Calculates the number equal to 1 * 2 * 3 * . . . * n-1 * n (notes usually as n!)
    - Returns: The number equal to the factorial of n as an integer
- factorial
    - Parameters: BigInteger – The number to find the factorial of
    - Functionality – Calculates the number equal to 1 * 2 * 3 * . . . * n-1 * n (notes usually as n!)
    - Returns: The number equal to the factorial of n as a BigInteger object

GeometricDistribution

The GeometricDistribution class is responsible for containing the methods that calculate probabilities based on a geometric distribution. This class has nine methods:

- geometricDistribution
    - Parameters: double probOfFirstSuccess – The probability of a successful trial, int numberOfFirstSuccess – The trial where the first success happens
    - Functionality – Calculates the probability of an event based off of a geometric distribution formula
    - Returns: double – The probability of the event
- geometricDistribution
    - Parameters: BigDecimal probOfFirstSuccess – The probability of a successful trial, BigInteger numberOfFirstSuccess – The trials where the first success happens

    - Functionality – Calculates the probability of an event based off of a geometric distribution formula
    - Returns: BigDecimal – The probability of the event
- expectedValue
    - Parameter: double probOfSuccess – The probability of a successful trial
    - Functionality – Calculates the expected value (mean) of a geometric distribution
    - Returns: double – The expected value (mean) of the geometric distribution
- expectedValue
    - Parameters: BigDecimal probOfSuccess – The probability of a successful trial
    - Functionality – Calculates the expected value (mean) of a geometric distribution
    - Returns: BigDecimal – The expected value (mean) of the geometric distribution
- variance
    - Parameter: double probOfSuccess – The probability of a successful trial
    - Functionality – Calculates the variance of a geometric distribution
    - Returns: double – The variance of the geometric distribution
- variance
    - Parameters: BigDecimal probOfSuccess – The probability of a successful trial
    - Functionality – Calculates the variance of a geometric distribution
    - Returns: BigDecimal – The variance of the geometric distribution
- standardDeviation
    - Parameter: double variance – The variance of the geometric distribution
    - Functionality – Calculates the standard deviation of a geometric distribution
    - Returns: double – The standard deviation of the geometric distribution
- standardDeviation
    - Parameter: BigDecimal variance – The variance of the geometric distribution
    - Functionality – Calculates the standard deviation of a geometric distribution
    - Returns: BigDecimal – The standard deviation of the geometric distribution
- testerOutput
    - Parameters: None
    - Functionality – Prints out examples of all methods
    - Returns: Nothing, but prints out statements

HypergeometricDistribution

The HypergeometricDistribution class is responsible for containing the methods that calculate probabilities based on a hypergeometric distribution. This class has nine methods:

- hypergeometricDistribution
    - Parameters: int N – The total number of items in the population, int n – The number of successful events in the population, int y – The number of successful events in the sample, int r – The number of items in the sample
    - Functionality – Calculates the probability of an event occurring based on a hypergeometric distribution formula
    - Returns: double – The probability of the event
- hypergeometricDistribution
    - Parameters: BigInteger N – The total number of items in the population, BigInteger n – The number of successful events in population, BigInteger y – The number of successful events in the sample, BigInteger r – The number of items in the sample
    - Functionality – Calculates the probability of an event occurring based on a hypergeometric distribution formula
    - Returns: BigDecimal – The probability of the event
- expectedValue
    - Parameters: int n – The number of successful events in the population, int r – The number of items in the sample, int N – The total number of items in the population
    - Functionality – Calculates the expected value (mean) of a hypergeometric distribution
    - Returns: double - The expected value (mean) of the hypergeometric distribution
- expectedValue
    - Parameters: BigInteger n – The number of successful events in the population, BigInteger r – The number of items in the sample, BigInteger N, The total number of items in the population
    - Functionality – Calculates the expected value (mean) of a hypergeometric distribution
    - Returns: BigDecimal – The expected value (mean) of the hypergeometric distribution
- variance
    - Parameters: int n – The number of successful events in the population, int r – The number of items in the sample, int N – The total number of items in the population
    - Functionality – Calculates the variance of a hypergeometric distribution

    - Returns: double – The variance of a hypergeometric distribution
- variance
    - Parameters: BigInteger n – The number of successful events in the population, BigInteger r – The number of items in the sample, BigInteger N – The total number of items in the population
    - Functionality – Calculate the expected value (mean) of the hypergeometric distribution
    - Returns: BigDecimal – The variance of the distribution
- standardDeviation
    - Parameter: double variance – The variance of the hypergeometric distribution
    - Functionality – Calculates the standard deviation of the hypergeometric distribution
    - Returns: double – The standard deviation of the hypergeometric distribution
- standardDeviation
    - Parameter: BigDecimal variance – The variance of the hypergeometric distribution
    - Functionality – Calculates the standard deviation of the hypergeometric distribution
    - Returns: BigDecimal – The standard deviation of the hypergeometric distribution
- testerOutput
    - Parameters: None
    - Functionality – Prints out examples of all methods
    - Returns: Nothing, but prints statements

NegativeBinomialDistribution

The NegativeBinomialDisrtibution class is responsible for containing the methods that calculate probabilities based on a negative binomial distribution. This class has nine methods:

- negativeBinomialDistribution
    - Parameters: int trialWithSuccess – The trial number that had the successful result, int numberOfSuccesses – The total number of successful trial, double probOfSuccess – The probability of a successful trial
    - Functionality – Calculates the probability of an event based on a negative binomial distribution
    - Returns: double – The probability of the event
- negativeBinomialDistribution
    - Parameters: BigInteger trialWithSuccess – The trial number that had the successful result, BigInteger numberOfSucesses – The total number of

successful trials, BigDecimal probOfSuccess – The probability of a successful trial
    - Functionality – Calculate the probability of an event based on a negative binomial distribution
    - Returns: BigDecimal – The probability of the event
- expectedValue
    - Parameters: int numberOfSuccesses – The total number of successes, double probOfSuccess – The probability of a successful trial
    - Functionality – Calculates the expected value (mean) of a negative binomial distribution
    - Returns: double – The expected value (mean) of the negative binomial distribution
- expectedValue
    - Parameters: BigInteger numberOfSuccesses – The total number of successes, BigDecimal probOfSuccess – The probability of a successful trial
    - Functionality – Calculates the expected value (mean) of a negative binomial distribution
    - Returns: BigDecimal – The expected value (mean) of the negative binomial distribution
- variance
    - Parameters: int numberOfSuccesses – The total number of successes, double probOfSuccess – The probability of a successful trial
    - Functionality – Calculates the variance of a negative binomial distribution
    - Returns: double – The variance of the negative binomial distribution
- variance
    - Parameters: BigInteger numberOfSuccesses – The total number of successes, BigDecimal probOfSuccess – The probability of a successful trial
    - Functionality – Calculates the variance of a negative binomial distribution
    - Returns: BigDecimal – The variance of the negative binomial distribution
- standardDeviation
    - Parameter: double variance – The variance of the negative binomial distribution
    - Functionality – Calculates the standard deviation of a negative binomial distribution
    - Returns: double – The standard deviation of the negative binomial distribution
- standardDeviation
    - Parameter: BigDecimal variance – The variance of the negative binomial distribution
    - Functionality – Calculates the standard deviation of a negative binomial distribution

    - Returns: BigDecimal – The standard deviation of the negative binomial distribution
- testerOutput
    - Parameters: None
    - Functionality – Prints out examples from all of the methods
    - Returns: Nothing, but prints out statements

Permutation

The Permutation class is responsible for containing the methods that calculate the number of permutation of objects. This class has three methods:

- getPermutation
    - Parameters: int n – The size of the set to choose from, int r – The number of choices to be made from the set
    - Functionality – Uses the permutation formula and integers to calculate the number of permutations that are possible
    - Returns: int – The total number of permutations that can be made from n and r
- getPermutation
    - Parameters: BigInteger n – The size of the set to choose from, BigInteger r – The number of choices to be made from the set
    - Functionality – Uses the permutation formula and BigInteger objects to calculate the number of permutations that are possible
    - Returns: BigInteger – The total number of permutations that can be made from n and r
- testerOutput
    - Parameters: None
    - Functionality – Prints out examples from each method
    - Returns: Nothing, but prints out statements

ProbabilityCalculations

The ProbabilityCalculations class is responsible for containing methods that calculate the probability of an event occurring using various theorems and rules. This class has thirteen methods:

- conditionalProbability
    - Parameters: double probAandB – The probability of A and B occurring at the same time, double probB – The probability of B
    - Functionality – Calculates the probability of A given B

  - Returns: double – The probability of A given B
- independence
  - Parameters: double probA – The probability of event A, double probB – The probability of event B, double probAandB – The probability of events A and B occurring at the same time
  - Functionality – Checks to see if the two events are independent
  - Returns: Boolean – True if the two events are independent, False if the two events are dependent
- independenceUsingAGivenB
  - Parameters: double probAgivenB – The probability of event A given B, double probA – The probability of event A
  - Functionality – Checks to see if the two events are independent using the probability of A given B and the probability of A
  - Returns: boolean – True if the two events are independent, false if the events are dependent
- independenceUsingBGivenA
  - Parameters: double probBgivenA – The probability of B given A, double probB – The probability of B
  - Functionality – Checks to see if the two events are independent using the probability of B given A and the probability of B
  - Returns: Boolean – True if the two events are independent, False if the two events are dependent
- mnRule
  - Parameter: ArrayList<Integer> listOfNumbers – The list of numbers to be multiplied
  - Functionality – Finds the total number of simple events given a set of numbers using the mn rule
  - Returns: int – The total number of events
- multinomialCoefficient
  - Parameters: int n – The number of objects, ArrayList<Integer> listOfSets – The list of groups
  - Functionality – Finds the number of possible groups of objects into multiple different groups
  - Returns: int – The number of groups that can be made
- multinomialCoefficient
  - Parameters: BigInteger n – The number of objects, ArrayList<BigInteger> listOfSets – The list of groups
  - Functionality – Finds the number of possible groups of objects into multiple different groups
  - Returns: BigInteger – The number of groups that can be made
- multiplicativeProbability

    - Parameters: double probA – The probability of event A, double probB – The probability of event B, double probAgivenB – The probability of event A given B
    - Functionality – Finds the probability of A and B, depending of if they are independent or not
    - Returns: double – The probability of A and B
- additiveProbability
    - Parameters: double probA – The probability of event A, double probB – The probability of event B, double probAandB – The probability of A and B
    - Functionality – Finds the probability of A or B, depending on if they are independent or not
    - Returns: double – The probability of A or B
- findingAUsingAInverse
    - Parameter: double probAInverse – The probability of A inverse
    - Functionality – Finds the probability of A using the probability of A inverse
    - Returns: double – The probability of A
- lawOfTotalProbability
    - Parameters: ArrayList<Double> AgivenBi – An ArrayList holding all of the probabilities of A given Bi, where i is another event, ArrayList<Double> Bi – An ArrayList holding all of the probabilities of Bi
    - Functionality – Calculates the probability of A happening across a range of different other events
    - Returns: double – The probability of A
- bayesRule
    - Parameters: ArrayList<Double> AgivenBi – An ArrayList holding all of the probabilities of A given Bi, where i is another event, ArrayList<Double> Bi – An ArrayList holding all the probabilities of Bi, double AgivenBj – The probability of A happening given a specific event, Bj, happening, double Bj – The probability of a specific event, Bj, happening
    - Functionality – Calculates the probability of A happening when we only know information about events B1, B2, . . ., Bi
    - Returns: double – The probability of a specific event, Bj, given A happens
- testerOutput
    - Parameters: None
    - Functionality – Prints out examples from each method
    - Returns: Nothing, but prints out statements

RandomVariableCalculations

The RandomVariableCalculations class is responsible for containing methods that calculate statistical measurements for random variables. This class has five methods:

- expectedValue
  - Parameters: ArrayList<Integer> valuesOfY – The values of the random variable Y, ArrayList<Double> probsOfY – The probabilities associated with each value of Y
  - Functionality – Calculates the expected value (mean) of a random variable Y
  - Returns: double – The expected value (mean) of Y
- expectedValueOfGOfY
  - Parameters: ArrayList<Integer> gOfY – The values of the random variable Y after transformation, ArrayList<Double> probsOfY – The probabilities associated with each value of Y
  - Functionality – Calculates the expected value (mean) of a transformed random variable Y
  - Returns: double – The expected value of g of Y
- variance
  - Parameters: ArrayList<Integer> valuesOfY – The values of the random variable Y, ArrayList<Double> probsOfY – The probabilities associated with each value of Y
  - Functionality – Calculates the variance of the random variable Y
  - Returns: double – The variance of the random variable Y
- standardDeviation
  - Parameters: double variance – The variance of the random variable Y
  - Functionality – Calculates the standard deviation of the random variable Y
  - Returns: double – The standard deviation of the random variable Y
- testerOutput
  - Parameters: None
  - Functionality – Prints out examples of each method
  - Returns: Nothing, but prints out statements

SetOperations

The SetOperations class is responsible for containing the methods that perform operations on a set. This class has four methods:

- union
  - Parameters: ArrayList<String> array1 – list of strings to be joined with array2, ArrayList<String> array2 – list of strings to be joined with array1

    - Functionality – Returns distinct strings that are shared between the two inputted ArrayLists
    - Returns: ArrayList<String> - The distinct strings of both ArrayList
- intersect
    - Parameters: ArrayList<String> array1 – list of strings to compare to array2, ArrayList<String> array2 – list of strings to compare to array1
    - Functionality – Returns the strings that are only in both ArrayLists
    - Returns: ArrayList<String> - The ArrayList of strings that are only in both ArrayList
- complement
    - Parameters: ArrayList<String> subset – The list that contains a part of the full dataset, ArrayList<String> sample – The list that contains a sample of the subset dataset
    - Functionality – Finds the strings that are not in the sample (the complement)
    - Returns: ArrayList<String> - The list of strings that are not in the given sample
- testerOutput
    - Parameters: None
    - Functionality – Prints out examples of each method
    - Returns: Nothing, but prints out statements

StatsLibrary

The StatsLibrary class is responsible for containing the methods that perform basic statistical operations. The class has eighteen methods:

- getMeanDouble
    - Parameter: double[] values – Dataset to have the mean calculated for
    - Functionality – Calculates the mean of the dataset
    - Returns: double – The mean of the dataset
- getMeanInt
    - Parameter: int[] values – Dataset to have the mean calculated for
    - Functionality – Calculates the mean of the dataset
    - Returns: double – The mean of the dataset
- getMeanDouble
    - Parameter: ArrayList<Double> values – Dataset to have the mean calculated for
    - Functionality – Calculates the mean of the dataset
    - Returns: double – The mean of the dataset
- getMeanInt
    - Parameter: ArrayList<Integer> values – Dataset to have the mean calculated for

    - Functionality – Calculates the mean of the dataset
    - Returns: double – The mean of the dataset
- getMedianDouble
    - Parameter: double[] values – Dataset to have the median calculated for
    - Functionality – Calculates the median of the dataset
    - Returns: double – The median of the dataset
- getMedianInt
    - Parameter: int[] values – Dataset to have the median calculated for
    - Functionality – Calculates the median of the dataset
    - Returns: double – The median of the dataset
- getMedianDouble
    - Parameter: ArrayList<Double> values – Dataset to have the median calculated for
    - Functionality – Calculates the median of the dataset
    - Returns: double – The median of the dataset
- getMedianInt
    - Parameter: ArrayList<Integer> values – Dataset to have the median calculated for
    - Functionality – Calculates the median of the dataset
    - Returns: double – The median of the dataset
- getModeDouble
    - Parameter: double[] values – Dataset to have the mode(s) calculated for
    - Functionality – Calculates the mode(s) of the dataset
    - Returns: ArrayList<Double> - The mode(s) of the dataset
- getModeInt
    - Parameter: int[] values – Dataset to have the mode(s) calculated for
    - Functionality – Calculates the mode(s) of the dataset
    - Returns: ArrayList<Integer> - The mode(s) of the dataset
- getModeDouble
    - Parameter: ArrayList<Double> values – Dataset to have the mode(s) calculated for
    - Functionality – Calculates the mode(s) of the dataset
    - Returns: ArrayList<Double> - The mode(s) of the dataset
- getModeInt
    - Parameter: ArrayList<Integer> values – Dataset to have the mode(s) calculated for
    - Functionality – Calculates the mode(s) of the dataset
    - Returns: ArrayList<Integer> - The mode(s) of the dataset
- getVarianceDouble
    - Parameter: double[] values – Dataset to have the variance calculated for
    - Functionality – Calculate the variance of the dataset

        - Returns: double – The variance of the dataset
- getVarianceInt
    - Parameter: int[] values – Dataset to have the variance calculated for
    - Functionality – Calculate the variance of the dataset
    - Returns: double – The variance of the dataset
- getVarianceDouble
    - Parameter: ArrayList<Double> values – Dataset to have the variance calculated for
    - Functionality – Calculate the variance of the dataset
    - Returns: double – The variance of the dataset
- getVarianceInt
    - Parameter: ArrayList<Integer> values – Dataset to have the variance calculated for
    - Functionality – Calculate the variance of the dataset
    - Returns: double – The variance of the dataset
- getStandardDeviation
    - Parameter: double variance – The variance of the dataset
    - Functionality – Calculates the standard deviation of the dataset
    - Returns: double – The standard deviation of the dataset
- testerOutput
    - Parameters: None
    - Functionality – Prints out examples from each method
    - Returns: Nothing, but prints out statements

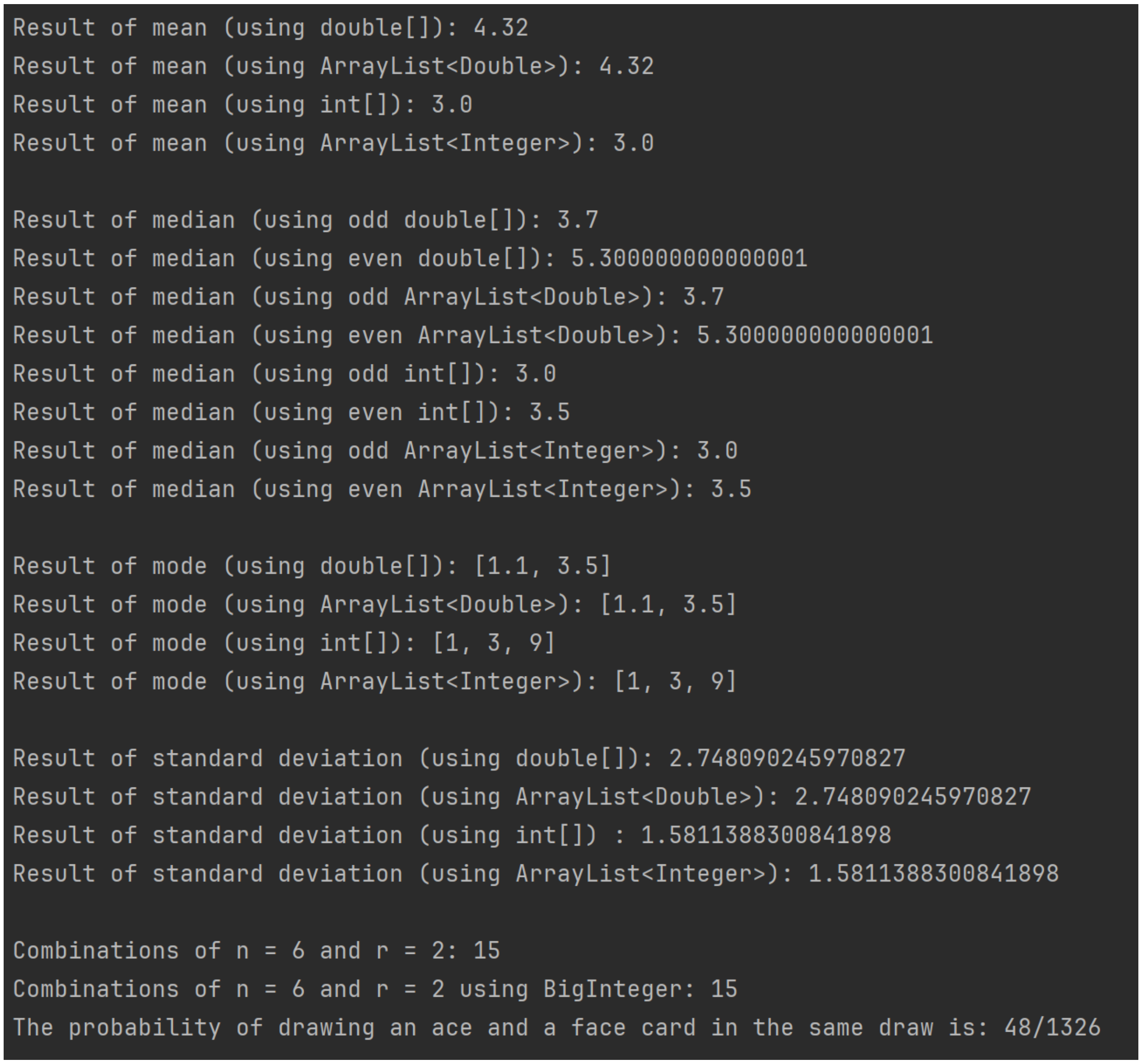
StatsTester

The StatsTester class is responsible for testing the classes contained within the StatsLibrary project. This class holds the main method and runs the testers associated with each class.

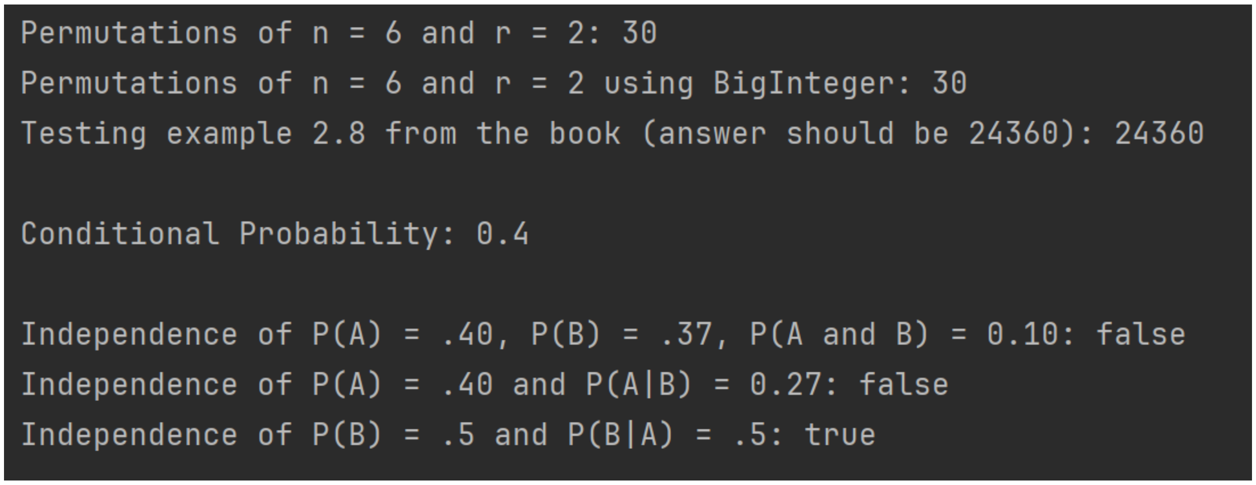
Output

The output of this project is examples of each method performing its statistical calculation.

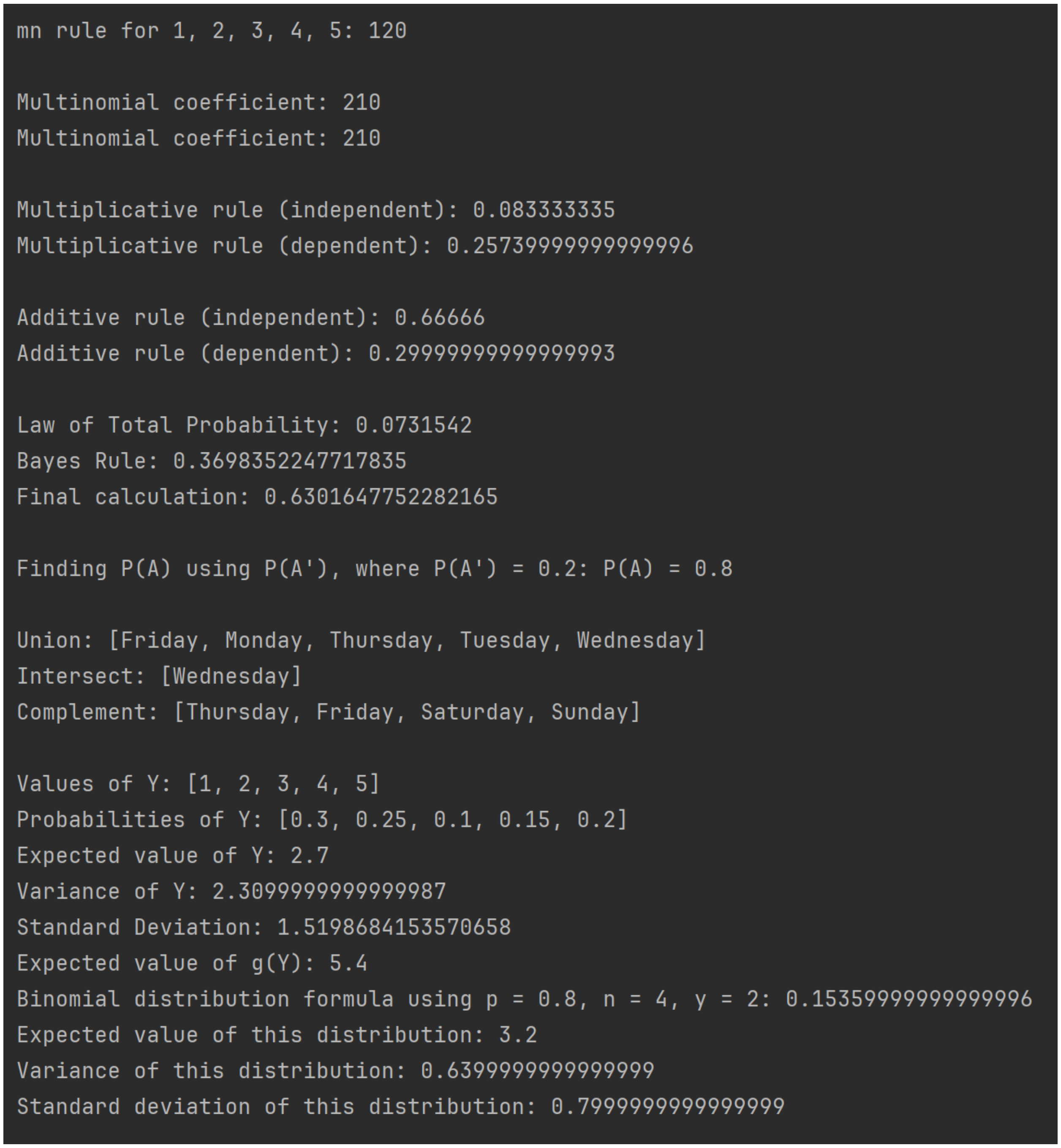
Screenshots



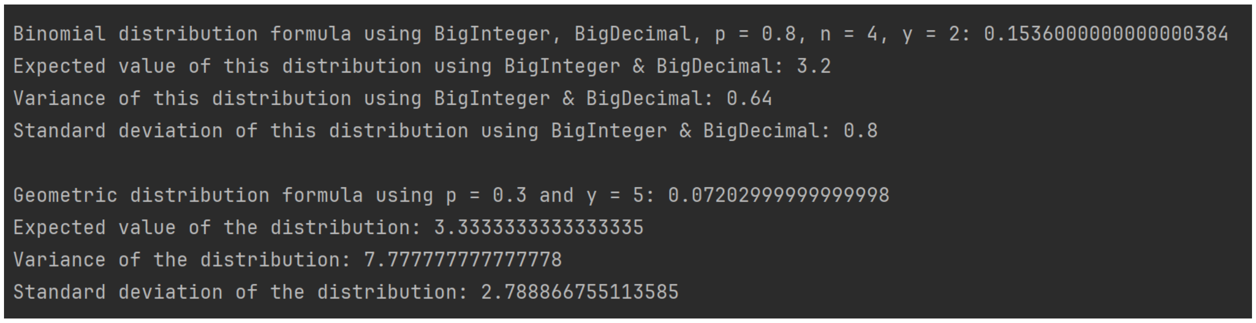
```
Result of mean (using double[]): 4.32
Result of mean (using ArrayList<Double>): 4.32
Result of mean (using int[]): 3.0
Result of mean (using ArrayList<Integer>): 3.0

Result of median (using odd double[]): 3.7
Result of median (using even double[]): 5.300000000000001
Result of median (using odd ArrayList<Double>): 3.7
Result of median (using even ArrayList<Double>): 5.300000000000001
Result of median (using odd int[]): 3.0
Result of median (using even int[]): 3.5
Result of median (using odd ArrayList<Integer>): 3.0
Result of median (using even ArrayList<Integer>): 3.5

Result of mode (using double[]): [1.1, 3.5]
Result of mode (using ArrayList<Double>): [1.1, 3.5]
Result of mode (using int[]): [1, 3, 9]
Result of mode (using ArrayList<Integer>): [1, 3, 9]

Result of standard deviation (using double[]): 2.748090245970827
Result of standard deviation (using ArrayList<Double>): 2.748090245970827
Result of standard deviation (using int[]) : 1.5811388300841898
Result of standard deviation (using ArrayList<Integer>): 1.5811388300841898

Combinations of n = 6 and r = 2: 15
Combinations of n = 6 and r = 2 using BigInteger: 15
The probability of drawing an ace and a face card in the same draw is: 48/1326
```



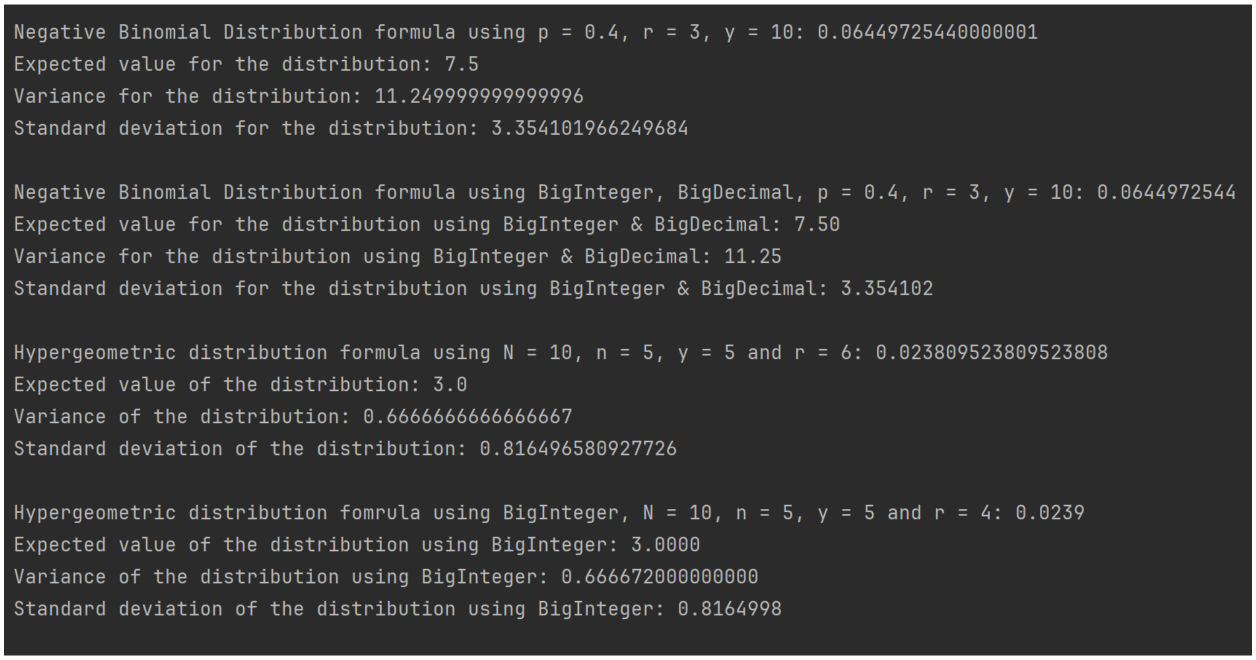
```
Permutations of n = 6 and r = 2: 30
Permutations of n = 6 and r = 2 using BigInteger: 30
Testing example 2.8 from the book (answer should be 24360): 24360

Conditional Probability: 0.4

Independence of P(A) = .40, P(B) = .37, P(A and B) = 0.10: false
Independence of P(A) = .40 and P(A|B) = 0.27: false
Independence of P(B) = .5 and P(B|A) = .5: true
```



```
mn rule for 1, 2, 3, 4, 5: 120

Multinomial coefficient: 210
Multinomial coefficient: 210

Multiplicative rule (independent): 0.083333335
Multiplicative rule (dependent): 0.25739999999999996

Additive rule (independent): 0.66666
Additive rule (dependent): 0.29999999999999993

Law of Total Probability: 0.0731542
Bayes Rule: 0.3698352247717835
Final calculation: 0.6301647752282165

Finding P(A) using P(A'), where P(A') = 0.2: P(A) = 0.8

Union: [Friday, Monday, Thursday, Tuesday, Wednesday]
Intersect: [Wednesday]
Complement: [Thursday, Friday, Saturday, Sunday]

Values of Y: [1, 2, 3, 4, 5]
Probabilities of Y: [0.3, 0.25, 0.1, 0.15, 0.2]
Expected value of Y: 2.7
Variance of Y: 2.3099999999999987
Standard Deviation: 1.5198684153570658
Expected value of g(Y): 5.4
Binomial distribution formula using p = 0.8, n = 4, y = 2: 0.15359999999999996
Expected value of this distribution: 3.2
Variance of this distribution: 0.6399999999999999
Standard deviation of this distribution: 0.7999999999999999
```



```
Binomial distribution formula using BigInteger, BigDecimal, p = 0.8, n = 4, y = 2: 0.1536000000000000384
Expected value of this distribution using BigInteger & BigDecimal: 3.2
Variance of this distribution using BigInteger & BigDecimal: 0.64
Standard deviation of this distribution using BigInteger & BigDecimal: 0.8

Geometric distribution formula using p = 0.3 and y = 5: 0.07202999999999998
Expected value of the distribution: 3.3333333333333335
Variance of the distribution: 7.777777777777778
Standard deviation of the distribution: 2.788866755113585
```



```
Negative Binomial Distribution formula using p = 0.4, r = 3, y = 10: 0.06449725440000001
Expected value for the distribution: 7.5
Variance for the distribution: 11.249999999999996
Standard deviation for the distribution: 3.354101966249684

Negative Binomial Distribution formula using BigInteger, BigDecimal, p = 0.4, r = 3, y = 10: 0.0644972544
Expected value for the distribution using BigInteger & BigDecimal: 7.50
Variance for the distribution using BigInteger & BigDecimal: 11.25
Standard deviation for the distribution using BigInteger & BigDecimal: 3.354102

Hypergeometric distribution formula using N = 10, n = 5, y = 5 and r = 6: 0.023809523809523808
Expected value of the distribution: 3.0
Variance of the distribution: 0.6666666666666667
Standard deviation of the distribution: 0.816496580927726

Hypergeometric distribution fomrula using BigInteger, N = 10, n = 5, y = 5 and r = 4: 0.0239
Expected value of the distribution using BigInteger: 3.0000
Variance of the distribution using BigInteger: 0.666672000000000
Standard deviation of the distribution using BigInteger: 0.8164998
```