Final Project

DS 5100 | Summer 2024 | Residential

Overview

For your Final Project, you will write, test, use, package, and publish a Python module and accompanying files.

The module will implement a simple Monte Carlo simulator using a set of three related classes — a Die class, a Game class, and an Analyzer class.

The classes are related in the following way: Game objects are initialized with a Die object, and Analyzer objects are initialized with a Game object.

[Die] → [Game] → [Analyzer]

In this simulator, a "die" can be any discrete random variable associated with a stochastic process, such as using a deck of cards, flipping a coin, rolling an actual die, or speaking a language.

The project is designed to integrate what you have learned in this class by calling upon the following areas of knowledge:

- Basic syntax, expressions, and statements in Python.
- Python Classes with initialization methods.
- Data manipulation with NumPy and Pandas.
- Literate programming with docstrings and documentation.
- Unit testing with Unittest.
- Simple plotting with Pandas.
- Program modularization and packaging with Setuptools.
- GitHub for managing and sharing code.

Class Definitions

The following class definitions provide blueprints for creating the classes for your simulator.

Note that although these provide some specific instructions, some elements of your code are left to your interpretation.

The Die class

General Definition

- A die has N sides, or "faces", and W weights, and can be rolled to select a face.

- For example, a "die" with $N = 2$ is a coin, and a one with $N = 6$ is a standard die.
- Normally, dice and coins are "fair," meaning that the each side has an equal weight. An unfair die is one where the weights are unequal.
- Each side contains a unique symbol. Symbols may be all alphabetic or all numeric.
- W defaults to 1.0 for each face but can be changed after the object is created.
- The weights are just positive numbers (integers or floats, including 0), not a normalized probability distribution.
- The die has one behavior, which is to be rolled one or more times.

Specific Methods and Attributes

An initializer.

- Takes a NumPy array of faces as an argument. Throws a `TypeError` if not a NumPy array.
- The array's data type `dtype` may be strings or numbers.
- The array's values must be distinct. Tests to see if the values are distinct and raises a `ValueError` if not.
- Internally initializes the weights to 1.0 for each face.
- Saves both faces and weights in a private data frame with faces in the index.

A method to change the weight of a single side.

- Takes two arguments: the face value to be changed and the new weight.
- Checks to see if the face passed is valid value, i.e. if it is in the die array. If not, raises an `IndexError`.
- Checks to see if the weight is a valid type, i.e. if it is numeric (integer or float) or castable as numeric. If not, raises a `TypeError`.

A method to roll the die one or more times.

- Takes a parameter of how many times the die is to be rolled; defaults to 1.
- This is essentially a random sample with replacement, from the private die data frame, that applies the weights.
- Returns a Python list of outcomes.
- Does not store internally these results.

A method to show the die's current state.

- Returns a copy of the private die data frame.

The Game class

General Definition

- A game consists of rolling of one or more similar dice (Die objects) one or more times.
- By similar dice, we mean that each die in a given game has the same number of sides and associated faces, but each die object may have its own weights.
- Each game is initialized with a Python list that contains one or more dice.
- Game objects have a behavior to play a game, i.e. to roll all of the dice a given number of times.
- Game objects only keep the results of their most recent play.

Specific Methods and Attributes

An initializer.

- Takes a single parameter, a list of already instantiated similar dice.
- Ideally this would check if the list actually contains Die objects and that they all have the same faces, but this is not required for this project.

A play method.

- Takes an integer parameter to specify how many times the dice should be rolled.
- Saves the result of the play to a private data frame.
- The data frame should be in wide format, i.e. have the roll number as a named index, columns for each die number (using its list index as the column name), and the face rolled in that instance in each cell.

A method to show the user the results of the most recent play.

- This method just returns a copy of the private play data frame to the user.
-

Takes a parameter to return the data frame in narrow or wide form which defaults to wide form.

- The narrow form will have a `MultiIndex`, comprising the roll number and the die number (in that order), and a single column with the outcomes (i.e. the face rolled).
- This method should raise a `ValueError` if the user passes an invalid option for narrow or wide.

The Analyzer class

General Definition

An Analyzer object takes the results of a single game and computes various descriptive statistical properties about it.

Specific Methods and Attributes

An initializer.

- Takes a game object as its input parameter. Throws a `ValueError` if the passed value is not a Game object.

A jackpot method.

- A jackpot is a result in which all faces are the same, e.g. all ones for a six-sided die.
- Computes how many times the game resulted in a jackpot.
- Returns an integer for the number of jackpots.

A face counts per roll method.

- Computes how many times a given face is rolled in each event.
    - For example, if a roll of five dice has all sixes, then the counts for this roll would be 5 for the face value '6' and 0 for the other faces.
- Returns a data frame of results.
- The data frame has an index of the roll number, face values as columns, and count values in the cells (i.e. it is in wide format)..

A combo count method.

- Computes the distinct combinations of faces rolled, along with their counts.
- Combinations are order-independent and may contain repetitions.
- Returns a data frame of results.
- The data frame should have a MultiIndex of distinct combinations and a column for the associated counts.

An permutation count method.

- Computes the distinct permutations of faces rolled, along with their counts.
- Permutations are order-dependent and may contain repetitions.

- Returns a data frame of results.
- The data frame should have a MultiIndex of distinct permutations and a column for the associated counts.

General Requirements for Classes

- All classes and methods must have appropriate docstrings.
- Class docstrings should describe the general purpose of the class.
- Method docstrings should describe the purpose of the method, any input arguments, any return values if applicable, and any changes to the object's state that the user should know about.
- Input argument descriptions should describe data types and formats as well as any default values.
- You may use language included in this document to create these docstrings.

Unit Tests

Write a unit test file using the Unittest package containing **at least one method for each method in each of the three classes** above. As a general rule, each test method should verify that the target method creates an appropriate data structure.

In [ ]:

Scenarios

To demonstrate the use of your simulator, you will apply your package to three scenarios, each of which consists of a set of task. These scenarios are found in the template file you will use to submit your assignment.

About the README file

The `README.md` file will be your the main source of documentation for your users, in addition to your use of docstrings in your code. The file should consist of the following sections:

- **Metadata**: Specify your name and the project name (i.e. Monte Carlo Simulator).
- **Synopsis:** Show brief demo code of how the classes are used, i.e. code snippets showing how to install, import, and use the code to (1) create dice, (2) play a game, and (3) analyze a game. You can use preformatted blocks for the code.

- **API description**: A list of all classes with their public methods and attributes. Each item should show their docstrings. All parameters (with data types and defaults) should be described. All return values should be described. Do not describe private methods and attributes.

Collaboration

You may work in groups to discuss idea and approaches, but all deliverables must be yours.

Submission

When finished completing the above tasks, save this file to your project repo, and them push it to your GitHub repo.

Then convert this file to a PDF and submit it to GradeScope according to the assignment instructions in Canvas.