Monte Carlo Simulator

Synopsis

This application creates three related classes ('Die', 'Game', and 'Analyzer') to simulate processes that rely on repeated random sampling. Below are the steps to install, import, and use the code to create a simulation.

Step 1: Install

```
pip install -e .
```

Step 2: Import

```
from montecarlo import Die, Game, Analyzer
```

Step 3: Create Die

i. create 'Die' object with numpy array

```
dieA = Die(np.array([1,2,3,4,5,6]))
```

ii. peek at attribute 'faces'

```
dieA.faces
```

iii. peek at attribute 'weights'

```
dieA.weights
```

iv. use method 'chng_wght' where **a** is the face and **b** is the new weight

```
a = 4
b = 3
dieA.chng_wght(a,b)

print("new die weights:" + str(dieA.weights))
```

v. use method 'roll_die' where **x** is the number of rolls

```
x = 10
dieA.roll_die(x)
```

vi. use method 'dataframe' to see a dataframe consisting of faces and weights

```
dieA.dataframe()
```

Step 4: Create Game

i. create list 'dice' with 'Die' objects

```
dieA = Die(np.array([1,2,3,4,5,6]))
dieB = Die(np.array([1,2,3,4,5,6]))
dieC = Die(np.array([1,2,3,4,5,6]))

dice=[dieA, dieB, dieC] # All 'Die' objects need to have same 'faces' for 'Game'
```

ii. create 'Game' object

```
game = Game(dice)
game.dice   # should return list of 'Die' objects
```

iii. use method 'play' where **x** is number of rolls dice and method 'play_results' to wide dataframe

```
game.play(4)
game.play_results()
```

iv. use method 'play_results' and specify form as 'narrow' to see narrow dataframe

```
game.play_results(form='narrow')
```

Step 5: Create Analyzer

i. create 'Die' objects and 'Game' object

```
dieA = Die(np.array([1,2,3,4,5,6]))
dieB = Die(np.array([1,2,3,4,5,6]))
dieC = Die(np.array([1,2,3,4,5,6]))

dice=[dieA, dieB, dieC] # All 'Die' objects need to have same 'faces' for 'Game'
game = Game(dice)
```

ii. create 'Analyzer' object

```
analyzed = Analyzer(game)
```

iii. use method 'jackpot' to see if any dice had all the same sides

```
analyzed.jackpot()
```

iv. use method 'face_count' to see the faces value counts for each round of rolls

```
analyzed.face_count()
```

v. use method 'combo_count' to see the combination counts (order independent)

```
analyzed.combo_count()
```

v. use method 'perm_count' to see the permuation counts (order dependent)

```
analyzed.perm_count()
```

API Description

Class: *Die*

A class to represent a die.

Attributes

- faces : np.ndarray
    - faces of die
- weights : np.ndarray
    - weights for the faces all set to 1

Methods

- **init**(faces) : Constructs attributes for the Die object
    - Parameters
        - faces : np.ndarray
            - faces of die
    - Returns
        - None
- chng_wght(face_val, new_wght) : Changes weight of specified die face
    - Parameters
        - face_val : int, str
            - value of face to change
        - new_wght : int, float, castable str
            - new weight to change face to
    - Returns
        - face value '{face_val}' has been given new weight '{new_wght}'
- roll_die(rolls=1) : Roll die to produce values based on random sampling
    - Parameters
        - rolls : int
            - number of samples, defaults to 1
    - Returns
        - list with length 'rolls' consisting of face values from random sampling
- dataframe( ) : Returns dataframe consisting of faces and weights
    - Parameters
        - None
    - Returns
        - dataframe consisting of faces and weights

Class: *Game*

*A class to represent a game using '*Die*' objects.*

Attributes

- dice : list
    - list of 'Die' objects

Methods

- **init**(dice) : Constructs attributes for the 'Game' object
    - Parameters
        - dice : list
            - list of 'Die' objects
    - Returns
        - None
- play(rolls) : Rolls list of 'Die' objects specified times
    - Parameters
        - rolls : int
            - number of samples
    - Returns
        - None
- play_results(form = 'wide') : Returns dataframe with results of play() in wide or narrow format
    - Parameters
        - form : str
            - defaults to 'wide', but accepts 'narrow' to stack dataframe-
    - Returns
        - if form is 'wide', wide dataframe of play_results
        - if form is 'narrow', narrow dataframe of play_results

Class: *Analyzer*

*A class to represent analysis using a '*Game*' object.*

Attributes

- play_results : pd.DataFrame
    - dataframe of object's Game.play_results()
- face_array : np.ndarray

- array of faces from first die of object's Game.dice()

Methods

- **init**(game): Constructs attributes for the 'Analyzer' object
  - Parameters
    - game : 'Game' object
      - 'Game' object
  - Returns
    - None
- jackpot( ): Returns jackpot counts
  - Parameters
    - None
  - Returns
    - count of Jackpots
- face_count( ): Returns dataframe with face counts
  - Parameters
    - None
  - Returns
    - dataframe consisting of value counts for faces in each round of rolls
- combo_count( ): Returns dataframe with combination counts
  - Parameters
    - None
  - Returns
    - dataframe consisting of the count of face combinations (independent of order)
- perm_count( ): Returns dataframe with permutation counts
  - Parameters
    - None
  - Returns
    - dataframe consisting of the count of face permutations (dependent of order)

About

Created by Richard 'Ricky' Kuehn from the University of Virginia. To contact, please email *fmt2tg@virginia.edu*