probability

November 28, 2017

1 Coin Flips and Die Rolls

Use NumPy to create simulations and compute proportions for the following outcomes. The first one is done for you.

```
In [5]: # import numpy
        import numpy as np
```

1.0.1 1. Two fair coin flips produce exactly two heads

```
In [6]: # simulate 1 million tests of two fair coin flips
        tests = np.random.randint(2, size=(int(1e6), 2))

        # sums of all tests
        test_sums = tests.sum(axis=1)

        # proportion of tests that produced exactly two heads
        (test_sums == 0).mean()

Out[6]: 0.24943100000000001

In [8]: tests2 = np.random.randint(2, size=(int(5), 2))
        tests2

Out[8]: array([[1, 0],
               [1, 0],
               [1, 1],
               [1, 1],
               [1, 0]])

In [11]: tests2.sum(axis=1)

Out[11]: array([1, 1, 2, 2, 1])
```

1.0.2 2. Three fair coin flips produce exactly one head

In [12]:
```
aaa = np.random.randint(2, size=(5, 3))
aaa
```

Out[12]:
```
array([[0, 1, 1],
       [0, 0, 0],
       [1, 1, 0],
       [1, 1, 0],
       [1, 0, 1]])
```

In [14]:
```
aaas = aaa.sum(axis=1)
aaas
```

Out[14]:
```
array([2, 0, 2, 2, 2])
```

In [15]:
```
# simulate 1 million tests of three fair coin flips
tests = np.random.randint(2, size=(int(1e6), 3))

# sums of all tests
test_sums = tests.sum(axis=1)

# proportion of tests that produced exactly one head
# 1 head means 2 tails
(test_sums == 2).mean()
```

Out[15]:
```
0.37559199999999998
```

1.0.3 3. Three biased coin flips with P(H) = 0.6 produce exactly one head

In [16]:
```
bbb = np.random.choice([0, 1], size=(5, 3), p=[0.6, 0.4])
bbb
```

Out[16]:
```
array([[0, 0, 1],
       [1, 1, 1],
       [1, 0, 1],
       [1, 0, 0],
       [0, 0, 0]])
```

In [17]:
```
bbbs = bbb.sum(axis=1)
bbbs
```

Out[17]:
```
array([1, 3, 2, 1, 0])
```

In [18]:
```
# simulate 1 million tests of three bias coin flips
# hint: use np.random.choice()
tests = np.random.choice([0, 1], size=(int(1e6), 3), p=[0.6, 0.4])

# sums of all tests
test_sums = tests.sum(axis=1)
```

```
         # proportion of tests that produced exactly one head
         # 1 head means 2 tails
         (test_sums == 2).mean()
```

```
Out[18]: 0.288244
```

1.0.4 4. A die rolls an even number

```
In [20]: ccc = np.random.choice([1,2,3,4,5,6], size=(10))
         ccc
```

```
Out[20]: array([6, 1, 5, 4, 6, 1, 3, 5, 1, 2])
```

```
In [27]: ddd = list(filter(lambda x:x%2==0, ccc))
         len(ddd)
```

```
Out[27]: 4
```

```
In [28]: def proportion_even(l):
             evens = list(filter(lambda x:x%2==0, l))
             return len(evens)/len(l)
```

```
In [29]: proportion_even(ccc)
```

```
Out[29]: 0.4
```

```
In [30]: # simulate 1 million tests of one die roll
         tests = np.random.choice([1,2,3,4,5,6], size=(int(1e6)))

         # proportion of tests that produced an even number
         proportion_even(tests)
```

```
Out[30]: 0.499714
```

1.0.5 5. Two dice roll a double

```
In [31]: eee = np.random.choice([1,2,3,4,5,6], size=(5,2))
         eee
```

```
Out[31]: array([[4, 6],
                [4, 4],
                [6, 2],
                [5, 5],
                [4, 1]])
```

```
In [34]: fff = list(filter(lambda x:x[0]==x[1], eee))
         fff
```

```
Out[34]: [array([4, 4]), array([5, 5])]
```

In [35]:
```
len(fff)
```

Out[35]: 2

In [36]:
```
def proportion_same(l):
    same = list(filter(lambda x:x[0]==x[1], l))
    return len(same)/len(l)
```

In [37]:
```
proportion_same(eee)
```

Out[37]: 0.4

In [38]:
```
# simulate the first million die rolls
first = np.random.choice([1,2,3,4,5,6], size=(int(1e6),2))

# simulate the second million die rolls
second = np.random.choice([1,2,3,4,5,6], size=(int(1e6),2))

# proportion of tests where the 1st and 2nd die rolled the same number
print(proportion_same(first))
print(proportion_same(second))
```

```
0.166908
0.166807
```