Probability 2: Loaded dice

In this assignment you will be reinforcening your intuition about the concepts covered in the lectures by taking the example with the dice to the next level.

This assignment will not evaluate your coding skills but rather your intuition and analytical skills. You can answer any of the exercise questions by any means necessary, you can take the analytical route and compute the exact values or you can alternatively create some code that simulates the situations at hand and provide approximate values (grading will have some tolerance to allow approximate solutions). It is up to you which route you want to take!

Note that every exercise has a blank cell that you can use to make your calculations, this cell has just been placed there for you convenience but **will not be graded** so you can leave empty if you want to.

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import utils
%matplotlib inline
```

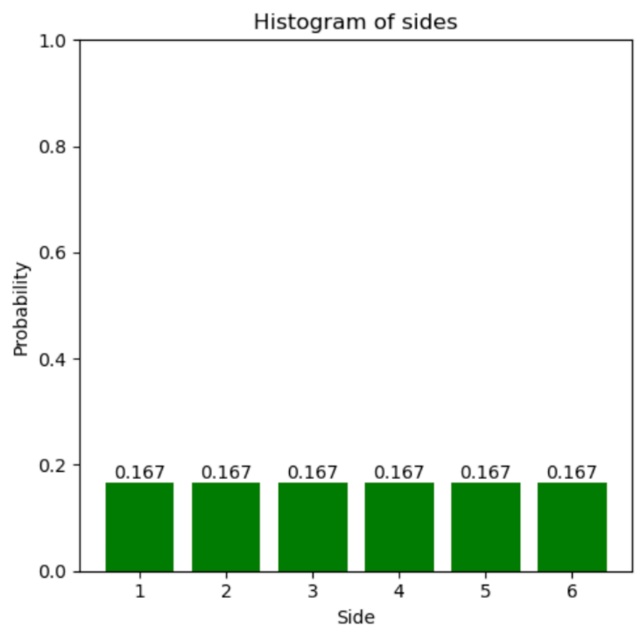
Some concept clarifications 🎲🎲🎲

During this assignment you will be presented with various scenarios that involve dice. Usually dice can have different numbers of sides and can be either fair or loaded.

- A fair dice has equal probability of landing on every side.
- A loaded dice does not have equal probability of landing on every side. Usually one (or more) sides have a greater probability of showing up than the rest.

Let's get started!

Given a 6-sided fair dice (all of the sides have equal probability of showing up), compute the mean and variance for the probability distribution that models said dice. The next figure shows you a visual represenatation of said distribution:



Submission considerations:

- Submit your answers as floating point numbers with three digits after the decimal point
- Example: To submit the value of 1/4 enter 0.250

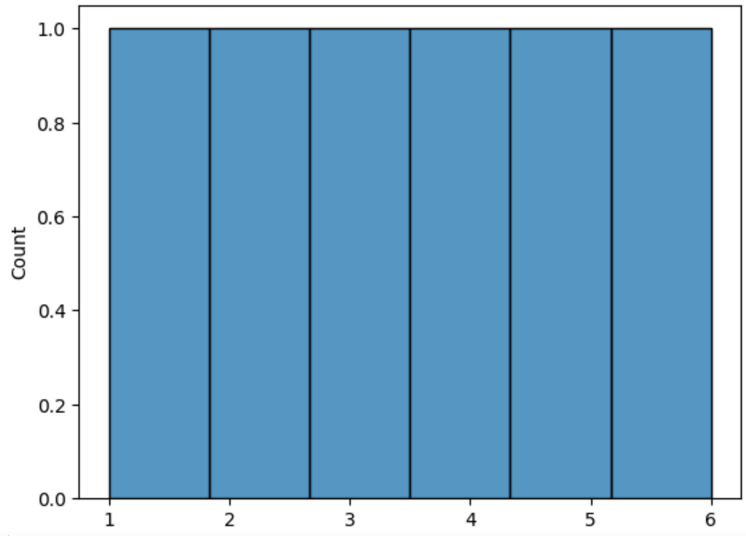
Hints:

- You can use np.random.choice to simulate a fair dice.
- You can use np.mean and np.var to compute the mean and variance of a numpy array.

```
# You can use this cell for your calculations (not graded)
x = [1,2,3,4,5,6]
```

```

sns.histplot(x, bins=6)
mean = np.mean(x)
var = np.var(x)
print(mean,var)
```

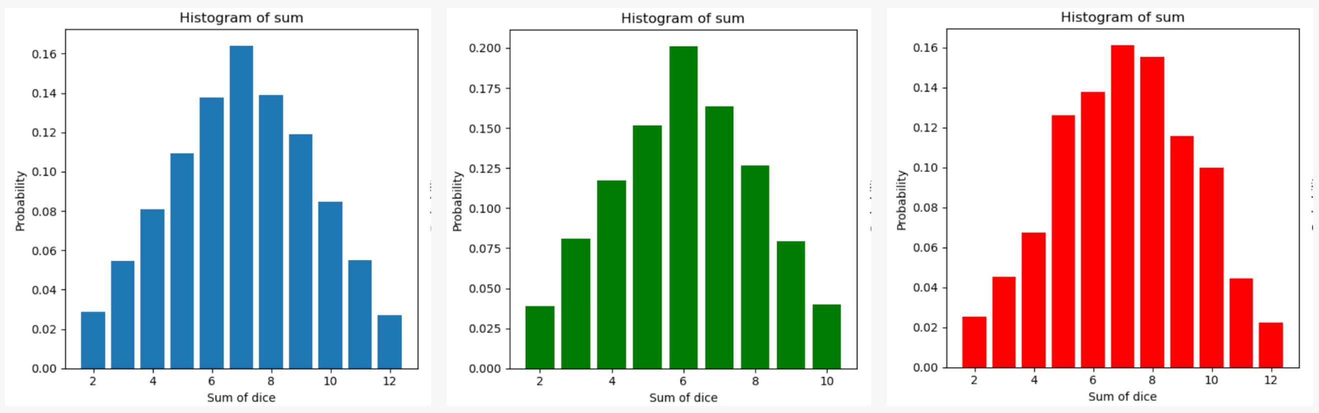
```
3.5 2.9166666666666665
```



```
# Run this cell to submit your answer
utils.exercise_1()
```

Exercise 2:

Now suppose you are throwing the dice (same dice as in the previous exercise) two times and recording the sum of each throw. Which of the following `probability mass functions` will be the one you should get?

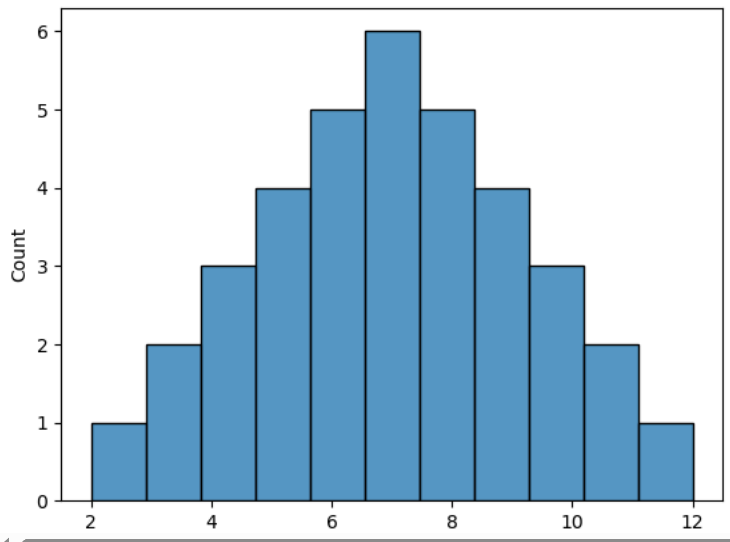


Hints:

- You can use numpy arrays to hold the results of many throws.
- You can sum to numpy arrays by using the `+` operator like this: `sum = first_throw + second_throw`
- To simulate multiple throws of a dice you can use list comprehension or a for loop

```
# You can use this cell for your calculations (not graded)
ans_arr = []
for i in x:
    for j in x:
        ans_arr.append(i+j)
sns.histplot(x=ans_arr,bins=11)
```

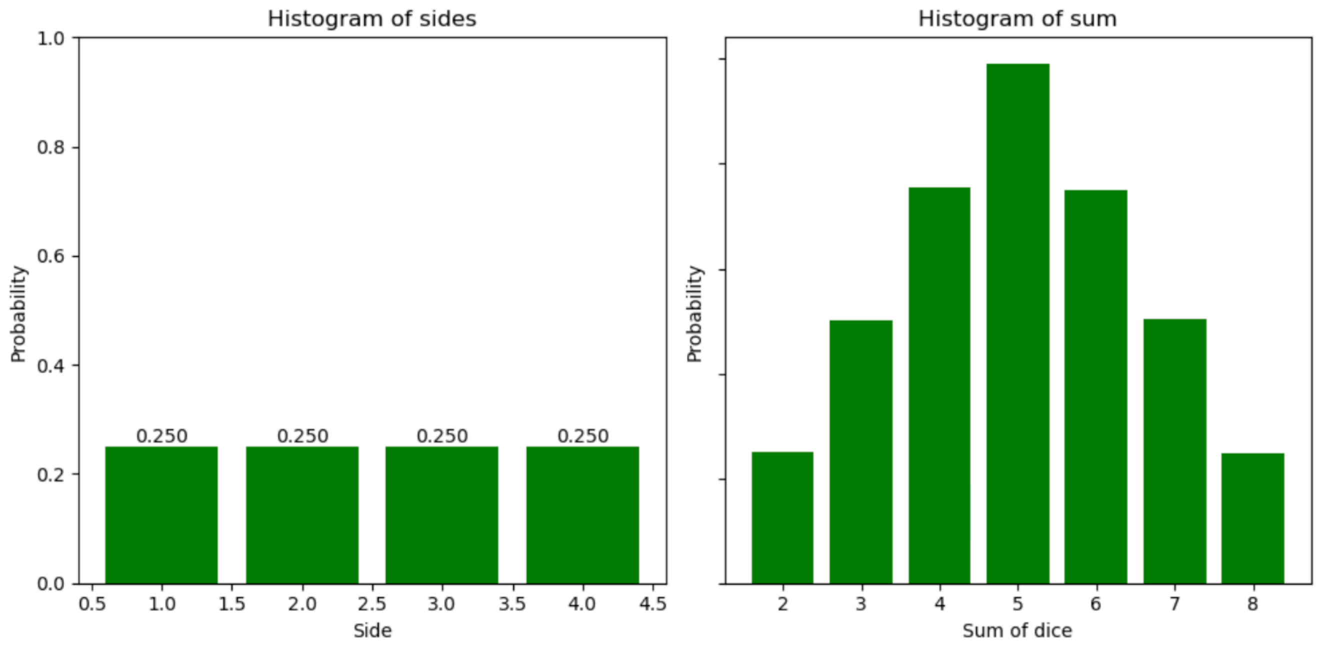
```
<Axes: ylabel='Count'>
```



```
# Run this cell to submit your answer
utils.exercise_2()
```

Exercise 3:

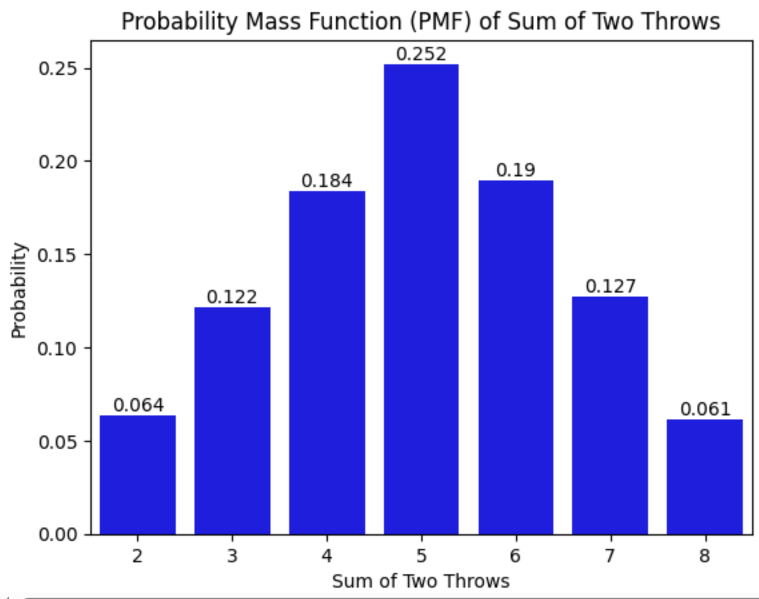
Given a fair 4-sided dice, you throw it two times and record the sum. The figure on the left shows the probabilities of the dice landing on each side and the right figure the histogram of the sum. Fill out the probabilities of each sum (notice that the distribution of the sum is symetrical so you only need to input 4 values in total):



Submission considerations:

- Submit your answers as floating point numbers with three digits after the decimal point
- Example: To submit the value of 1/4 enter 0.250

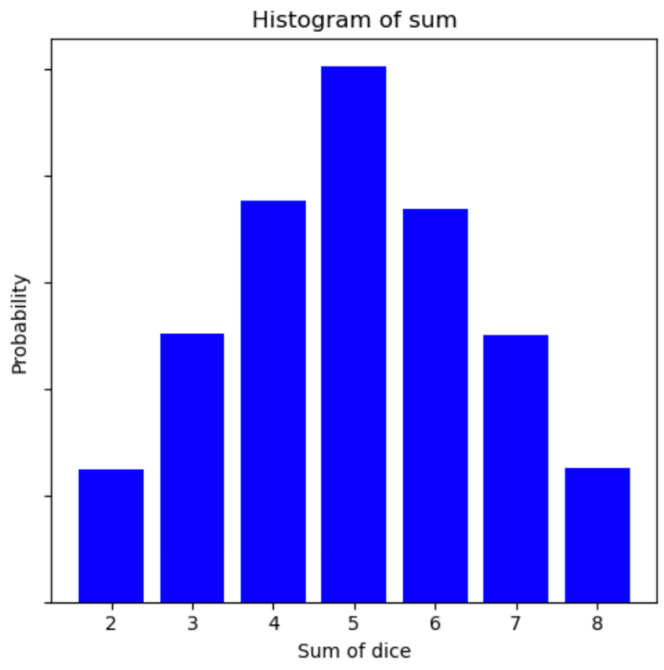
```
# You can use this cell for your calculations (not graded)

def roll_fair_dice(n_side=6, throw_times=2, num_throw=10000, pmf_plot=False):
    # Define the probabilities of each side of the die
    probabilities = [1/n_side]*n_side

    # throw each die
    throws = np.random.choice(np.arange(1,n_side+1), size=(num_throw, throw_times), p=probabilities)
    sums = np.sum(throws, axis=1)
    cov = np.cov(*throws.T)

    if pmf_plot:
         # Calculate the PMF for each possible sum
        unique_sums, counts = np.unique(sums, return_counts=True)
        pmf = counts / len(sums)
        # Plot the PMF using seaborn
        sns.barplot(x=unique_sums, y=pmf, color='blue')
        plt.xlabel('Sum of Two Throws')
        plt.ylabel('Probability')
        plt.title('Probability Mass Function (PMF) of Sum of Two Throws')
        # show bar value
        for i, v in enumerate(pmf):
            plt.text(i, v, str(round(v, 3)), ha='center', va='bottom')
        plt.show()
    else:
        return sums, cov

roll_fair_dice(4, pmf_plot=True)
```



```
# Run this cell to submit your answer
utils.exercise_3()
```

Exercise 4:

Using the same scenario as in the previous exercise. Compute the mean and variance of the sum of the two throws and the covariance between the first and the second throw:



Hints:

- You can use np.cov to compute the covariance of two numpy arrays (this may not be needed for this particular exercise).

```
# You can use this cell for your calculations (not graded)
```

```
throws_sums_result, cov = roll_fair_dice(n_side=4,throw_times=2,num_throw=10000)
print("mean:", round(np.mean(throws_sums_result),3), "\nvariance:", round(np.var(throws_sums_result),3))
print("covariance:", cov)
```

```
mean: 5.005
variance: 2.518
covariance: [[1.24410841 0.00559816]
 [0.00559816 1.26332597]]
```

```
# Run this cell to submit your answer
utils.exercise_4()
```

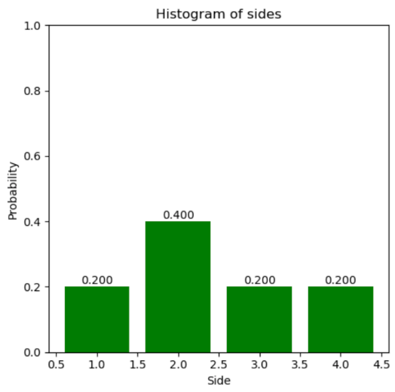
Mean:	5.000
Variance:	2.518
Covariance:	0

Save your answer!

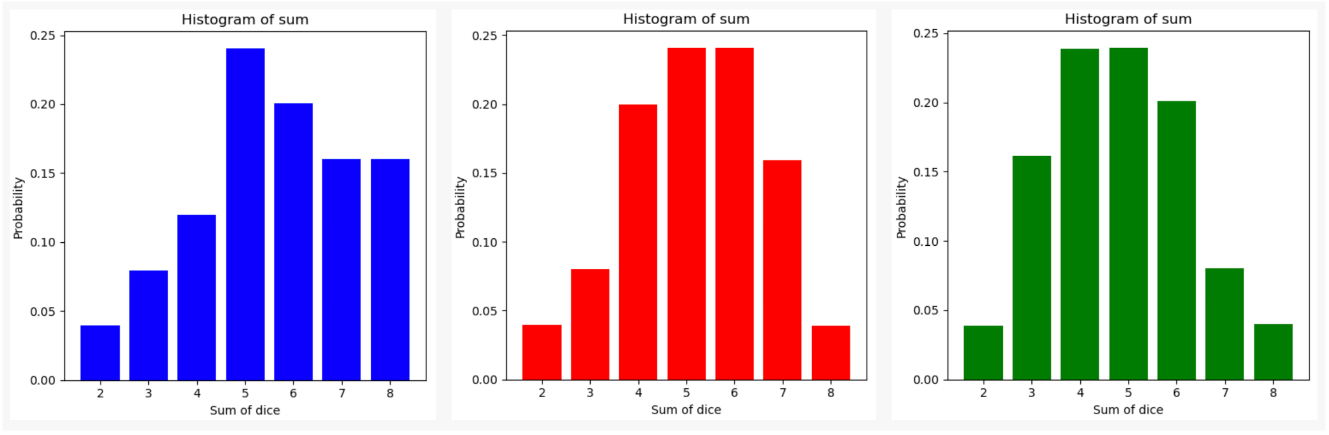
```
Answer for exercise 4 saved
```

Exercise 5:

Now suppose you are have a loaded 4-sided dice (it is loaded so that it lands twice as often on side 2 compared to the other sides):



You are throwing it two times and recording the sum of each throw. Which of the following `probability mass functions` will be the one you should get?

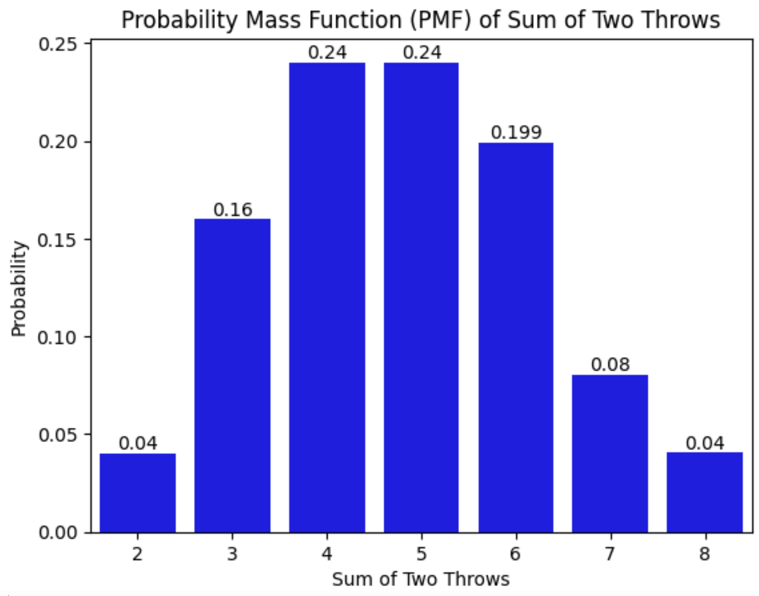


Hints:

- You can use the `p` parameter of np.random.choice to simulate a loaded dice.

```
# You can use this cell for your calculations (not graded)
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

def roll_loaded_dice(n_side=6, loaded_position = 1, throw_times=2, num_throw=10000, pmf_plot=False):
    # The die is lands twice as often on the loaded side compared to the other sides

    # Define the probabilities of each side of the dice
    probabilities = [1/(n_side+1)]*(loaded_position-1) + [2/(n_side+1)] + [1/(n_side+1)]*(n_side-loaded_position)

    assert np.sum(probabilities) == 1, "The sum of the probabilities should be 1"

    # Perform the two throws and calculate the sum
```

```python
    throws = np.random.choice(np.arange(1,n_side+1), size=(num_throw, throw_times), p=probabilities)
    sums = np.sum(throws, axis=1)
    cov = np.cov(*throws.T)

    if pmf_plot:
        # Calculate the PMF for each possible sum
        unique_sums, counts = np.unique(sums, return_counts=True)
        pmf = counts / len(sums)

        # Plot the PMF using seaborn
        sns.barplot(x=unique_sums, y=pmf, color='blue')
        plt.xlabel('Sum of Two Throws')
        plt.ylabel('Probability')
        plt.title('Probability Mass Function (PMF) of Sum of Two Throws')
        for i, v in enumerate(pmf):
            plt.text(i, v, str(round(v, 3)), ha='center', va='bottom')
        plt.show()
    else:
        return sums, cov

roll_loaded_dice(n_side=4, loaded_position=2, throw_times=2, num_throw=1000000, pmf_plot=True)
```



```python
res, cov = roll_loaded_dice(n_side=4, loaded_position=2, throw_times=2, num_throw=1000000)
print("mean:", np.mean(res), "\nvariance:", np.var(res), "\ncov:", cov)
```

```
mean: 4.799068
variance: 2.078756331376
cov: [[1.03916319e+00 2.09181662e-04]
 [2.09181662e-04 1.03917686e+00]]
```

```python
# Run this cell to submit your answer
utils.exercise_5()
```

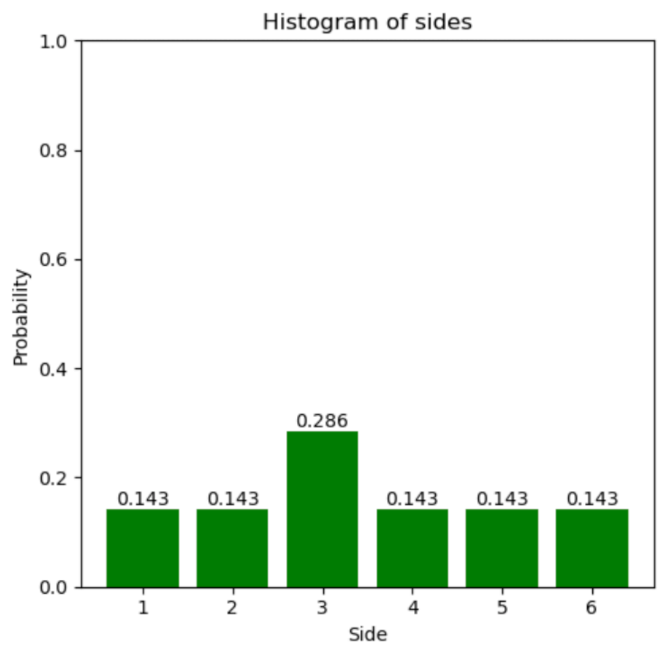
Your answer: left center [✓ right]

Save your answer!

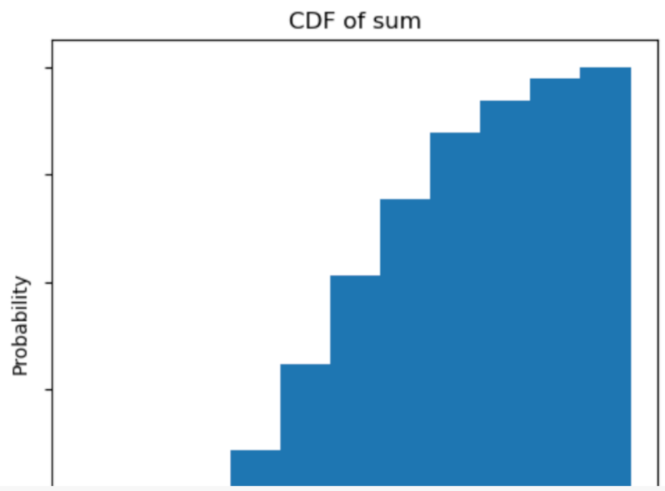
```
Answer for exercise 5 saved.
```

Exercise 6:

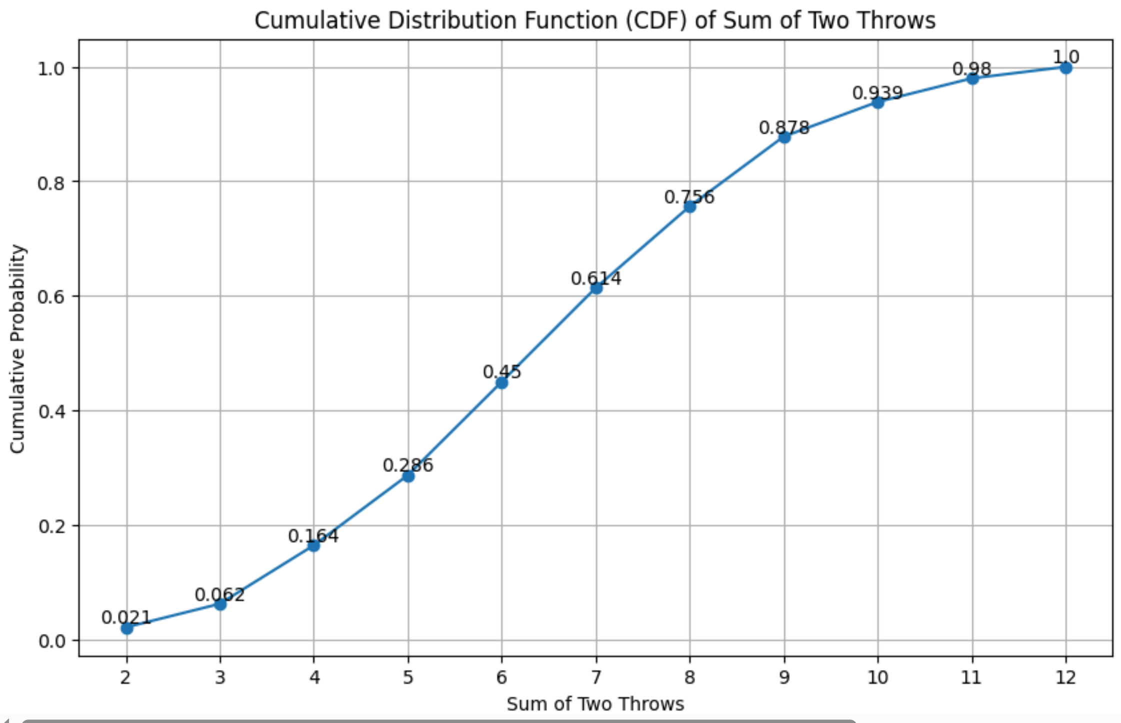
You have a 6-sided dice that is loaded so that it lands twice as often on side 3 compared to the other sides:



You record the sum of throwing it twice. What is the highest value (of the sum) that will yield a cumulative probability lower or equal to 0.5?



```
# You can use this cell for your calculations (not graded)
import numpy as np

def cdf_loaded_dice(n_side=6, loaded_position=3, throw_times=2, num_throw=10000, cdf_plot=True):
    throws, _ = roll_loaded_dice(n_side=n_side, loaded_position=loaded_position, throw_times=throw_times, num_throw=num_throw, pmf_pl

    unique_sums, counts = np.unique(throws, return_counts=True)
    pmf = counts / len(throws)
    cdf = np.cumsum(pmf)

    if cdf_plot:
        plt.figure(figsize=(10, 6))
        plt.grid(True)
        plt.plot(unique_sums, cdf, marker='o')
        plt.xlabel('Sum of Two Throws')
        plt.ylabel('Cumulative Probability')
        plt.title('Cumulative Distribution Function (CDF) of Sum of Two Throws')
        plt.xticks(unique_sums)
        for i, v in enumerate(cdf):
            plt.text(i+2, v, str(round(v, 3)), ha='center', va='bottom')
        plt.show()
    else:
        return cdf

cdf_loaded_dice(n_side=6, loaded_position=3, throw_times=2, num_throw=1000000, cdf_plot=True)
```



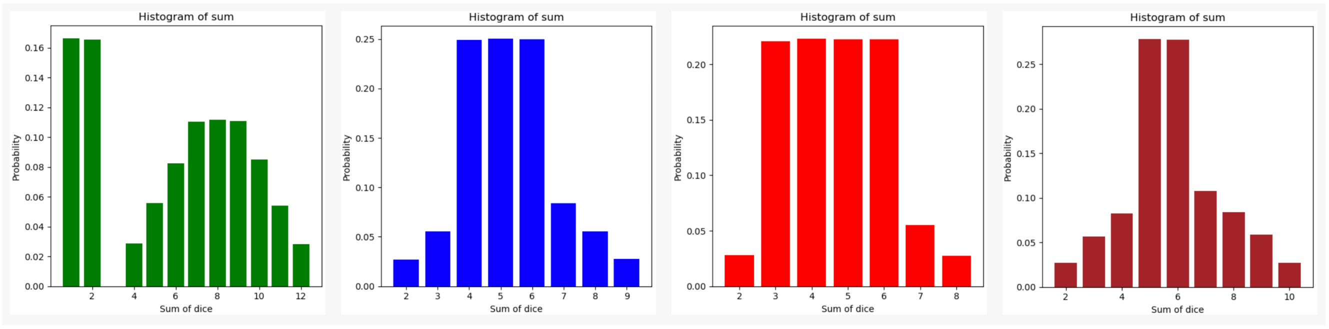
```
# Run this cell to submit your answer
utils.exercise_6()
```

Sum: 6

Save your answer!

Exercise 7:

Given a 6-sided fair dice you try a new game. You only throw the dice a second time if the result of the first throw is **lower** or equal to 3. Which of the following `probability mass functions` will be the one you should get given this new constraint?

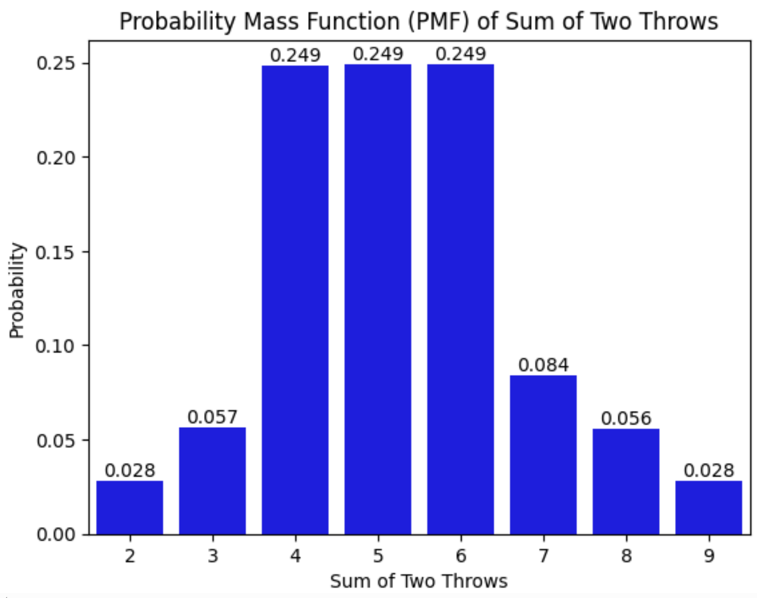


Hints:

- You can simulate the second throws as a numpy array and then make the values that met a certain criteria equal to 0 by using np.where

```
import numpy as np
import matplotlib.pyplot as plt

def exercise_7(n_side=6, throw_times=2, num_throw=100000, pmf_plot=False):
    # Define the probabilities of each side of the die
    probabilities = [1/n_side]*n_side

    # throw each die
    throws = np.random.choice(np.arange(1,n_side+1), size=(num_throw, throw_times), p=probabilities)
    throws[:,1] = np.where(throws[:,0] <= 3, throws[:,1], 0)
    sums = np.sum(throws, axis=1)

    if pmf_plot:
         # Calculate the PMF for each possible sum
        unique_sums, counts = np.unique(sums, return_counts=True)
        pmf = counts / len(sums)

        # Plot the PMF using seaborn
```

```python
        sns.barplot(x=unique_sums, y=pmf, color='blue')
        plt.xlabel('Sum of Two Throws')
        plt.ylabel('Probability')
        plt.title('Probability Mass Function (PMF) of Sum of Two Throws')
        # show bar value
        for i, v in enumerate(pmf):
            plt.text(i, v, str(round(v, 3)), ha='center', va='bottom')
        plt.show()
    else:
        return sums

exercise_7(pmf_plot=True)
```



```python
# Run this cell to submit your answer
utils.exercise_7()
```

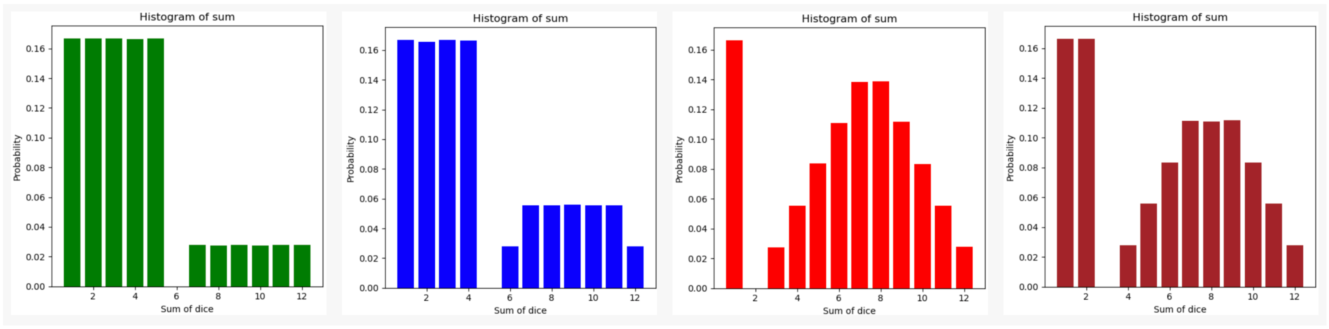
Your answer: left-most left-center right-center right-most

Save your answer!

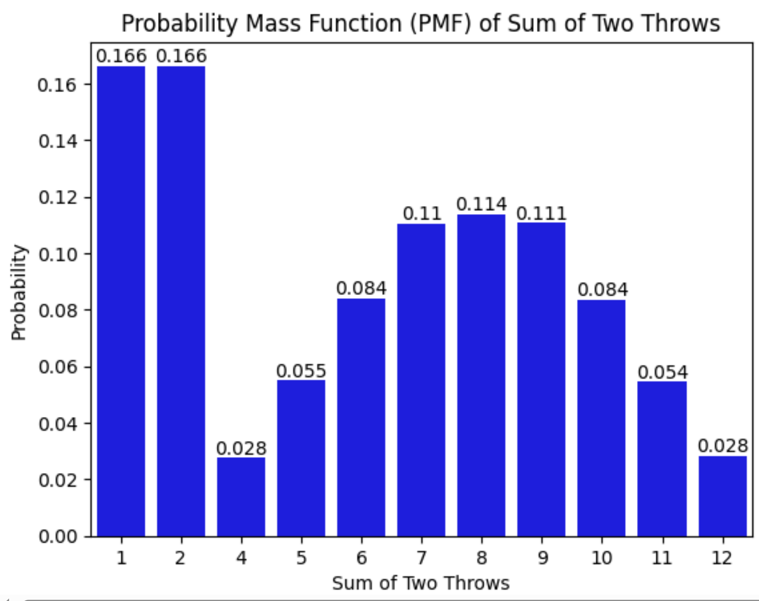
Answer for exercise 7 saved

Exercise 8:

Given the same scenario as in the previous exercise but with the twist that you only throw the dice a second time if the result of the first throw is **greater** or equal to 3. Which of the following `probability mass functions` will be the one you should get given this new constraint?



```python
# You can use this cell for your calculations (not graded)

def exercise_8(n_side=6, throw_times=2, num_throw=100000, pmf_plot=False):
    # Define the probabilities of each side of the die
    probabilities = [1/n_side]*n_side

    # throw each die
    throws = np.random.choice(np.arange(1,n_side+1), size=(num_throw, throw_times), p=probabilities)
    throws[:,1] = np.where(throws[:,0] >= 3, throws[:,1], 0)
    sums = np.sum(throws, axis=1)

    if pmf_plot:
         # Calculate the PMF for each possible sum
        unique_sums, counts = np.unique(sums, return_counts=True)
        pmf = counts / len(sums)

        # Plot the PMF using seaborn
        sns.barplot(x=unique_sums, y=pmf, color='blue')
        plt.xlabel('Sum of Two Throws')
        plt.ylabel('Probability')
        plt.title('Probability Mass Function (PMF) of Sum of Two Throws')
        # show bar value
        for i, v in enumerate(pmf):
            plt.text(i, v, str(round(v, 3)), ha='center', va='bottom')
        plt.show()
    else:
        return sums

exercise_8(pmf_plot=True)
```



```python
# Run this cell to submit your answer
utils.exercise_8()
```

Your answer: left-most left-center right-center [✓ right-most]

Save your answer!

Answer for exercise 8 saved

Exercise 9:

Given a n-sided fair dice. You throw it twice and record the sum. How does increasing the number of sides `n` of the dice impact the mean and variance of the sum and the covariance of the joint distribution?

```python
# You can use this cell for your calculations (not graded)
for i in range(2,11):
    throws_result, cov = roll_fair_dice(n_side=i,throw_times=2,num_throw=100000)
    print(f"n_side: {i} \nmean: {np.mean(throws_result)} \nvariance: {np.var(throws_result)} \ncov: {cov[0][1]} \n")
```

```
n_side: 2
mean: 3.0
variance: 0.50064
cov: 0.000320849608496093

n_side: 3
mean: 4.00109
variance: 1.3400688119000002
cov: 0.005162460624606193

n_side: 4
mean: 4.99141
variance: 2.4956362119
cov: -0.002861720017200082

n_side: 5
mean: 6.00415
variance: 4.0030127774999995
cov: 0.005990504905049128

n_side: 6
mean: 6.99524
```

```
variance: 5.8388973424
cov: 0.00013438534385360376

n_side: 7
mean: 7.98635
variance: 7.9477036775
cov: -0.024719326793268118

n_side: 8
mean: 8.99574
variance: 10.5569018524
cov: 0.012055906159061812

n_side: 9
mean: 10.01462
variance: 13.2920062556
cov: -0.024260250602506205

n_side: 10
mean: 11.01417
variance: 16.5486492111
cov: 0.018935092150922067
```

```
# Run this cell to submit your answer
utils.exercise_9()
```

```
As the number of sides in the die increases:
```

The mean … stays the same [✓ increases] decreases

The varian… stays the same [✓ increases] decreases

The covari… [✓ stays the same] increases decreases

Save your answer!

[illegible]

Exercise 10:

Given a 6-sided loaded dice. You throw it twice and record the sum. Which of the following statemets is true?

```
# You can use this cell for your calculations (not graded)

loaded_position = np.arange(1,7)
for i in loaded_position:
    throws_result, cov = roll_loaded_dice(n_side=6, loaded_position=i,throw_times=2,num_throw=100000)
    print(f"loaded_position: {i} \nmean: {np.mean(throws_result)} \nvariance: {np.var(throws_result)} \ncov: {cov[0][1]} \n")
```

```
loaded_position: 1
mean: 6.27902
variance: 6.5114478396000015
cov: 0.0014715670156702373

loaded_position: 2
mean: 6.56284
variance: 5.520831134399999
cov: -0.010056558565585646

loaded_position: 3
mean: 6.86398
variance: 5.0760185596000005
cov: 0.005384933849338481

loaded_position: 4
mean: 7.14074
variance: 5.0194522524
cov: -0.01020568855688556

loaded_position: 5
mean: 7.425
variance: 5.595975
cov: 0.023824356643566292

loaded_position: 6
mean: 7.72464
variance: 6.5002568704
cov: -0.006355140351403741
```

```
# Run this cell to submit your answer
utils.exercise_10()
```

- the mean and variance is the same regardless of which side is loaded
- having the sides 3 or 4 loaded will yield a higher covariance than any other sides
- the mean will decrease as the value of the loaded side increases
- [✓ changing the loaded side from 1 to 6 will yield a higher mean but the same variance]

Save your answer!

```
Answer for exercise 10 saved
```

the mean and variance is the same regardless of which side is loaded

having the sides 3 or 4 loaded will yield a higher covariance than any other sides

the mean will decrease as the value of the loaded side increases

changing the loaded side from 1 to 6 will yield a higher mean but the same variance

Exercise 11:

Given a fair n-sided dice. You throw it twice and record the sum but the second throw depends on the result of the first one such as in exercises 7 and 8. Which of the following statements is true?

```
# You can use this cell for your calculations (not graded)

print(
    """ChatGPT explain:
Modeling a Dice Example
Assume:
We have a 6-sided die with values from 1 to 6.
First roll: Rolled normally, called X.
Second roll: Depends on X, called Y.
---

Case A: Y ≤ X (i.e., the second roll is less than or equal to the first)
Examples:
If the first roll is 3, then the second roll can only be 1, 2, or 3.
If the first roll is 6, then the second roll can be 1, 2, 3, 4, 5, or 6.
Result:
When X is large → Y has a wider range → Y tends to increase.
When X is small → Y is more restricted → Y tends to be smaller.
→ Therefore, as X increases, Y also tends to increase → Cov(X, Y) > 0.
---

Case B: Y ≥ X (the second roll is greater than or equal to the first)
The opposite happens:
If the first roll is 3, then the second roll can be 3, 4, 5, or 6.
If the first roll is 6, then the second roll can only be 6.
→ When X is small → Y has more room to be larger → Y tends to increase.
→ When X is large → Y is limited → Y tends to be smaller.
→ As X increases, Y tends to decrease → Cov(X, Y) < 0.
    """
)
```

```
ChatGPT explain:
Modeling a Dice Example
Assume:
We have a 6-sided die with values from 1 to 6.
First roll: Rolled normally, called X.
Second roll: Depends on X, called Y.
---

Case A: Y ≤ X (i.e., the second roll is less than or equal to the first)
Examples:
If the first roll is 3, then the second roll can only be 1, 2, or 3.
If the first roll is 6, then the second roll can be 1, 2, 3, 4, 5, or 6.
Result:
When X is large → Y has a wider range → Y tends to increase.
When X is small → Y is more restricted → Y tends to be smaller.
→ Therefore, as X increases, Y also tends to increase → Cov(X, Y) > 0.
---

Case B: Y ≥ X (the second roll is greater than or equal to the first)
The opposite happens:
If the first roll is 3, then the second roll can be 3, 4, 5, or 6.
If the first roll is 6, then the second roll can only be 6.
→ When X is small → Y has more room to be larger → Y tends to increase.
→ When X is large → Y is limited → Y tends to be smaller.
→ As X increases, Y tends to decrease → Cov(X, Y) < 0.
```

```
# Run this cell to submit your answer
utils.exercise_11()
```

- [✓ changing the direction of the inequality will change the sign of the covariance]
- changing the direction of the inequality will change the sign of the mean
- changing the direction of the inequality does not affect the possible values of the sum
- covariance will always be equal to 0

Save your answer!

changing the direction of the inequality will change the sign of the covariance

changing the direction of the inequality will change the sign of the mean

changing the direction of the inequality does not affect the possible values of the sum

covariance will always be equal to 0

Exercise 12:

Given a n-sided dice (could be fair or not). You throw it twice and record the sum (there is no dependance between the throws). If you are only given the histogram of the sums can you use it to know which are the probabilities of the dice landing on each side?

```
# You can use this cell for your calculations (not graded)

print(" Yes, I can find the n-sided of the dice, then solve system of equations the find the probability of each side.\n Noted:\n -
```

```
 Yes, I can find the n-sided of the dice, then solve system of equations the find the probability of each side.
 Noted:
 - That in a fair dice, there only one result of the equation is 1/n for every side.
 - But in loaded dice, there a many result for the equation, mean that the are many different loaded dice can
achive the same histogram of the sums.
```

```
# Run this cell to submit your answer
utils.exercise_12()
```

- yes, but only if one of the sides is loaded
- no, regardless if the die is fair or not
- yes, but only if the die is fair
- [✓ yes, regardless if the die is fair or not]

Save your answer!

Before Submitting Your Assignment

Run the next cell to check that you have answered all of the exercises

```
utils.check_submissions()
```

```
All answers saved, you can submit the assignment for grading!
```

Congratulations on finishing this assignment!

During this assignment you tested your knowledge on probability distributions, descriptive statistics and visual interpretation of these concepts. You had the choice to compute everything analytically or create simulations to assist you get the right answer. You probably also realized that some exercises could be answered without any computations just by looking at certain hidden queues that the visualizations revealed.

Keep up the good work!