

## ▼ RANDOMNESS AND SAMPLING

by Dr Juan H Klopper

- Research Fellow
- School for Data Science and Computational Thinking
- Stellenbosch University



## ▼ INTRODUCTION

This notebook is a high-level look at the basics of randomness and patterns in data. Note that (almost) none of the code that we see here, will be used for actual calculations in later notebooks. It is nonetheless a very important notebook as the code is here to illustrate the meaning behind key ideas in Data Science.

## ▼ PACKAGES USED IN THIS NOTEBOOK

```
1 %config InlineBackend.figure_format = "retina" # For high-DPI displays

1 %load_ext google.colab.data_table

1 import numpy as np # Numerical calculations and random values
2 from scipy import stats # Statistical functions and statistical distributions
3 import pandas as pd # Data import and manipulation

1 # Plotting
2 import plotly.graph_objects as go
3 import plotly.io as pio
4 import plotly.express as px
5 import plotly.figure_factory as ff # Another plotly module

1 pio.templates.default = 'plotly white'
```

## ▼ RANDOMNESS

Randomness can be a difficult concept to understand or to simulate. When asked to come up with 100 random numbers from 0 to 9, very few of us will generate a truly random set of values. Even if tasked with writing down the outcomes of 100 coin flips, we inevitably fail.

Fortunately, computers can use various algorithms to produce **pseudo-random values**. By many measures they are random, yet they are produced by an algorithm with inputs and are not truly random as for example the decay of radioactive nuclei.

The numpy package contains a `random` module that can generate pseudo-random values. The `seed` function seeds the numpy random number generator (algorithm) with an integer value. When used, the same *random* values will always be produced. This is used in order to make code reproducible, i.e. we all get the same set of random values.

Below, we use the `choice` function to select a random item from a list. The list is passed as an argument to the `choice` function. The list contains two string elements, `Heads` and `Tails`. The `choice` function will choose one of the elements from the list. Each element has an equal likelihood of being selected at random.

```
1 np.random.seed(42) # Seed the pseudo-random number generator
2 np.random.choice(['Heads', 'Tails']) # Choose one of the two list items

'Heads'
```

`Heads` is selected at *random*.

Below, we add an integer value as argument to the `choice` function in order to return the specified number of random values.


```
1 np.random.seed(42) # Seed the pseudo-random number generator
2 np.random.choice(['Heads', 'Tails'], 10) # Ten random coin flips

array(['Heads', 'Tails', 'Heads', 'Heads', 'Heads', 'Tails', 'Heads',
       'Heads', 'Heads', 'Tails'], dtype='<U5')
```

The code iterates through the `choice` 10 times. After every random choice the selected item is *returned to the list* to be available to be selected again. This is termed **random selection with replacement**.

We can simulate a coin flip too. Below, we flip a coin 10000 times, assigning the result to a dataframe object.

```
1 np.random.seed(None)
2 flips = pd.DataFrame({'Flip':np.random.choice(['Heads', 'Tails'], 10000)}) # Ter
3 flips.head()
```

1 to 5 of 5 entries  

index	Flip
0	Heads
1	Heads
2	Heads
3	Heads
4	Heads

Show  per page

The `value_counts` method of the dataframe object, `flips`, shows a nearly equal frequency.

```
1 flips.Flip.value_counts()

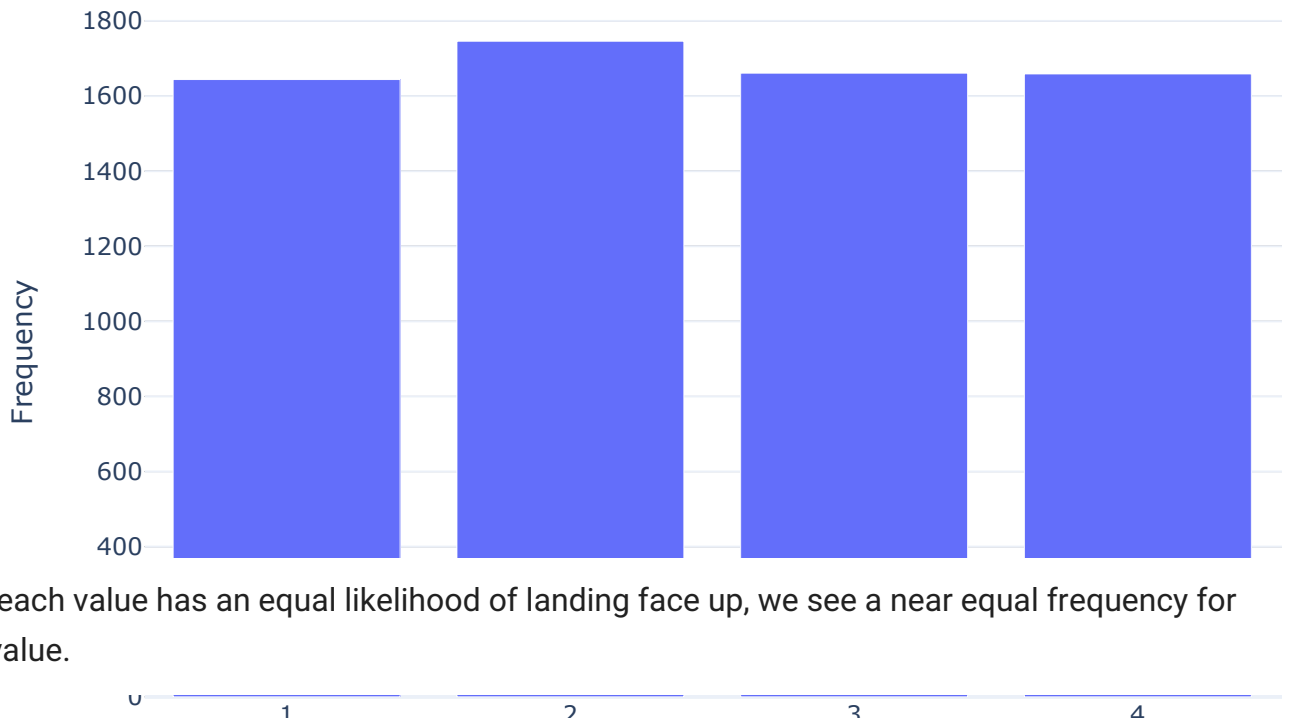
Heads    5073
Tails    4927
Name: Flip, dtype: int64
```

We can do the same for a fair, six-sided die, plotting the results as a bar plot. Below, we roll the die 10000 times and assign the resultant numpy array to a column in a dataframe object. Remember that with a single column, its is actually a series object.

```
1 np.random.seed()
2 sides = [1, 2, 3, 4, 5 ,6]
3 die = pd.DataFrame({'Die':np.random.choice(sides, 10000)})

1 px.bar(
2     x=sides,
3     y=die.Die.value_counts().sort_index().values.tolist(),
4     title='Frequency of die faces following 10000 rolls',
5     labels={'x':'Face value', 'y':'Frequency'}
6 )
```

## Frequency of die faces following 10000 rolls



Since each value has an equal likelihood of landing face up, we see a near equal frequency for each value.

There are many other random functions in numpy except for `choice`. We will learn about some of them later in this notebook.

## ▼ INTRODUCING PROBABILITIES

We recognized the fact that each element in the list objects that we created above to simulate a coin and a die, has an equal likelihood of being selected at random. From this we develop the idea of probability.

**Probability** is a branch of mathematics that allows us to investigate random events. With probability we focus on the occurrence of random events. By understanding these events, we can model real-world cases.

Probability theory requires a basic understanding of sets. Sets are collections of objects. We have already seen that these objects can be categorical or numerical in nature. There are many more sets, but these are the ones we are interested in.

We can create subsets of a set. A **subset** of a list contains some of the elements in a larger set. We can examine the intersection of sets, the union of sets, and the membership of a set.

Consider the sets of side-effects of two new drugs being created in a pharmacological laboratory. If a side-effect is present in one or both of the sets then it is a **member** of that set (or those sets). The **intersection** of the two sets, is the set of elements (side effects) that they have in common. The **union** of the two sets combines (without repetition) all the elements (side-effects) of both sets.

We usually consider a universal set. A **universal set** is a collection of all possible elements. In our example of the side-effects of a drug, we might consider a long list of side-effects. Each of the new drug only have some of the side-effects in the long list, which is the universal set. The universal set can be seen as the **sample space** of all possible elements. To date, we have included very specific elements in the sample space and only allowed the elements that actually occurred in our study as the sample space. The be sure, a sample space can contain more elements than just the ones we captured for our study.

If a set is contained within a universal set, the **complement** of the set are all the elements that are in the universal set, but not in the contained set itself.

## ▼ THE PROBABILITY OF OUTCOMES

Probability theory examines the random nature with which events occur. In probability theory an **event** is the outcome of a random selection of an element in a set. We are interested in the likelihood of the occurrence of that event. An **experiment** allows one of the elements of a set to be chosen at random. The event is the outcome of that experiment, selecting one of the elements in the set.

Our interest usually lie in independent events. For **independent events**, the current outcome of an experiments is unaffected by any prior outcome(s).

One very common probability is that of flipping a fair coin. If fair, the side that lands face up is an independent *event*. Flipping a coint twice would result in one the following possible *outcomes*, where H is heads and T is tails:

- HH
- HT
- TH
- TT

Fliiping the coin twice can only have one of these outcomes. Together they make up the sample space of our experiment of flipping the die twice. The actual outcome of two flips is a subset of

the sample space (consisting of four elements). In this case, each outcome has a 25% likelihood of being the outcome. We write that  $P(HH) = 0.25$  (the probability of H and H is 25%).

We can state a different outcome entirely. If the outcome is *at least one head in two flips*, we would have  $P(\text{at least one head}) = 0.75$ , as three of the four equally likely outcomes has a head in it.

What if the outcome is *at least one head in three flips*? We can answer this question by considering the sample space of outcomes. In each of the three flips, we can get either a head or a tail as event. Each flip is independent of the one that came before. Both head and tail each have a likelihood of 0.5 of landing face up. The possible outcomes are:

1. HHH
2. HHT
3. HTH
4. THH
5. HTT
6. THT
7. TTH
8. TTT

Any one of the outcomes has a one-in-eight likelihood. The probability of at least one head is  $P(\text{at least one head}) = 7/8$  as it is only the last sample space element that does not have at least one head.

```
1 # Probability
2 7/8

0.875
```

Let's have some fun and simulate flipping a coin three times and doing this 1000 times in a row. With the `choice()` function in the `random` module of `numpy` we can create two elements. We'll choose 0 and 1 to represent tails and heads respectively. The second argument states how many times we want to choose from this set, which is 3 in our case, and then we set the `replace` argument to `True`, meaning once 0 or 1 is selected it goes back in the *basket* to be selected again (selection with replacement). We need to do this for our example, as we are simulating three flips.

```
1 # Everytime this code is run, a new outcome will appear
2 np.random.choice(np.array([0, 1]), 3, replace=True)

array([0, 0, 1])
```

We can use list comprehension to run the code five times.

```
1 [np.random.choice(np.array([0, 1]), 3, replace=True) for i in range(5)]

[array([0, 1, 1]),
 array([0, 0, 1]),
 array([1, 0, 0]),
 array([0, 1, 0]),
 array([0, 0, 0])]
```

If we sum each array and it is more than 0, we know that we have a head. We can now start a count at 0, flip the coin three times, see if the sum is more than 0, and if so, add to the counter. Let's see if we get close to a probability of 0.875 as theoretically predicted, when we repeated our experiment 1000 times.

```
1 np.random.seed(2) # Seeding the pseudo-random number generator
2 count = 0 # Counter
3
4 for i in range(1000):
5     flip_3 = sum(np.random.choice(np.array([0, 1]), 3, replace=True))
6
7     if flip_3 > 0:
8         count += 1
9
10 print('Total number of experiments: 1000', '\n', 'Total number with at least one

Total number of experiments: 1000
Total number with at least one heads: 877
```

Remember that seeding the pseudo-random number generator means that we will get the same *random* output every time. Our result is 0.877. No too far off from the prediction. Try other pseudo-random number generator integers such as 12 or increase the number of cases to 10000 or more.

From our current narrative it should be clear that probabilities range from 0.0 to 1.0. We can never get a negative probability or an outcome with a probability of more than 100%. Furthermore, if an outcome either occurs or does not occur, we note (1) for the probability of the outcome occurring versus the probability of it not occurring. This means that the probability of mutually exclusive, collectively exhaustive outcomes sum to 1.0.

$$P(\text{event not occurring}) = 1 - P(\text{event occurring}) \quad (1)$$

## ▼ USING NEGATION

Here, we make use of (1) above.

Let's use our fair die again. We ask what the probability of rolling a 6 is, given a single roll. What about the probability of at least one six in two consecutive rolls (only a single six is required)? Then what about three, four, and more rolls. We have seen that the probability of rolling a 6 at a single roll is one-sixth.

To answer this type of question we look at the *event not occurring*. The probability of a 6 not been rolled on a single roll is shown in (2).

$$P(\neg 6) = 1 - \frac{1}{6} = \frac{5}{6} \quad (2)$$

The  $\neg$  symbol is used for negation. It reads *the probability of not being a 6*. From (1), we have (3), the probability of a 6.

$$P(6) = 1 - P(\neg 6) = 1 - \frac{5}{6} \quad (3)$$

The probability of there not being a 6 in two rolls is shown in (4), where we use powers.

$$P(6 \text{ in two rolls}) = 1 - [P(\neg 6)]^2 = 1 - \left(\frac{5}{6}\right)^2 = 1 - \frac{25}{36} = \frac{11}{36} \quad (4)$$

In  $n$  rolls we have (5).

$$P(6 \text{ in } n \text{ rolls}) = 1 - \left(\frac{5}{6}\right)^n \quad (5)$$

We can use list comprehension to calculate show the probability of a six in 1 through 10 rolls.

```
1 six = [1 - (5/6)**n for n in range(1, 11)]
2 six
[0.16666666666666663,
0.305555555555555547,
0.42129629629629617,
0.5177469135802468,
0.598122427983539,
0.6651020233196159,
0.7209183527663465,
0.7674319606386221,
```

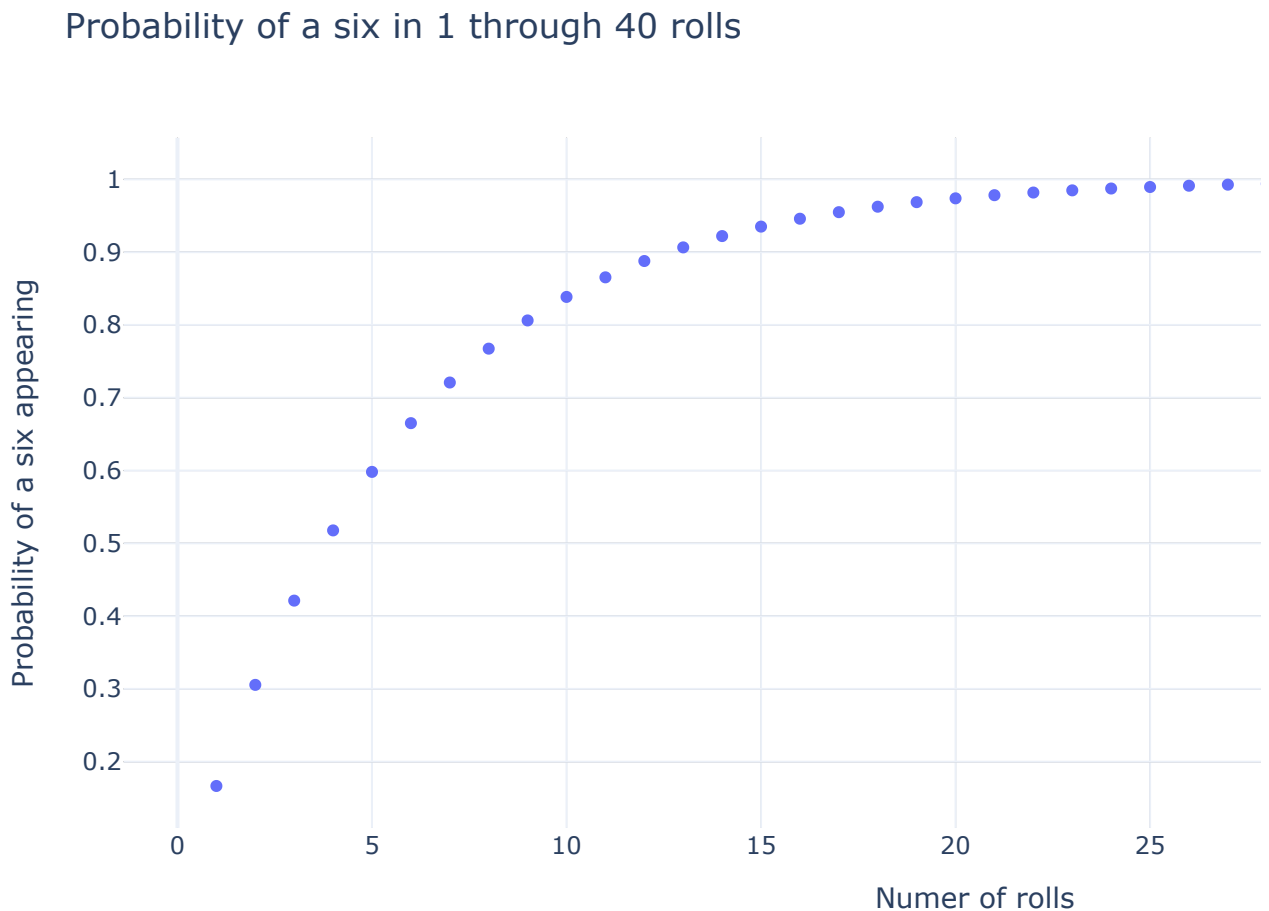


```
0.8061933005321851,
0 030404411711015421
```

With 10 rolls there is an 83.8% probability of a 6 landing face up.

We can plot these probabilities for 40 rolls.

```
1 px.scatter(
2     x=range(1, 41),
3     y=[1 - (5/6)**n for n in range(1, 41)],
4     title='Probability of a six in 1 through 40 rolls',
5     labels={
6         'x':'Numer of rolls',
7         'y':'Probability of a six appearing'
8     }
9 )
```




## ▼ TYPES OF PROBABILITIES

We need to discuss the types of probabilities. Outcomes are often stated as relative frequencies instead of absolute counts. It is important to be able to work with relative frequencies when considering probabilities.

There are three types of probabilities, namely unconditional, joint, and conditional probabilities. **Unconditional probability** (or marginal probability) is a probability of an outcome given the sample space for a single variable. We consider two containers, I and II, each containing balls of three individual colors: red, green, and blue. Below, we create a dataframe object with relative frequencies for all entries.

```
1 # Balls only have a single color
2 df = pd.DataFrame({'Container':['I', 'II'], 'Red':[0.26, 0.09], 'Green':[0.36, 0.07], 'Blue':[0.18, 0.04]})
3 df.set_index('Container', inplace=True) # Setting the first column as the index
4 df
```

1 to 2 of 2 entries  

Container	Red	Green	Blue
I	0.26	0.36	0.18
II	0.09	0.07	0.04

Show  per page

Since the values are all expressed as relative frequencies (the frequency divided by the sample size), we note that they add to 1.0 for all the balls.

```
1 0.26 + 0.36 + 0.18 + 0.09 + 0.07 + 0.04
1.0
```

The relative frequency of a red ball in container I is 0.26, and so on. We can now say that the unconditional probability of a ball being in container I is the total of all the relative frequencies in the first row. We can calculate the sum of each of the rows with the `.sum()` method. The `axis=1` argument sums along each of the columns (i.e. the rows).

```
1 df.sum(axis=1)

Container
I      0.8
II     0.2
dtype: float64
```

We see that the unconditional probability for a ball being in container I is 0.8.

The unconditional probability of being a green ball, would be the column total (i.e. the sums along each row for that column). We use `axis=0`.

```
1 df.sum(axis=0)

Red      0.35
```

```
Green    0.43
Blue     0.22
dtype: float64
```

We see the unconditional probability of a green ball is 0.43 and this is independent on which container the ball is in.

In a **joint probability** we are interested in two outcomes at once. The joint probability of a ball being in container I and being green is the intersection of the probabilities of being in I and being green. In our table, it is 0.36, the relative frequency.

A **conditional probability** is the probability of an outcome, given that another has occurred. The conditional probability of being in container I given that a green ball is chosen at random. We write  $P(I|\text{green})$ . This is the ratio of a joint probability to an unconditional probability. The joint probability of being a green ball and being in container I is 0.36. The probability of being in container I is 0.8. The conditional probability of being in container I given that the ball is green is then calculated below.

The equation for a conditional probability is shown in (6).

$$P(A|B) = \frac{P(A \cap B)}{P(B)} \quad (6)$$

We will use the  $A$  in (6) to be container I and  $B$  to be a green ball. The intersection symbol,  $\cap$ , refers to the joint probability. We see the result below, where the probability of being in container I and being green is  $P(A \cap B) = 0.36$  and the probability of being a green ball is  $P(B) = 0.43$ .

```
1 0.36 / 0.43
```

```
0.8372093023255813
```

There is a 83.7% probability that the ball is from container I.

We can use conditional probabilities to see if two events are independent of each other. For two outcomes A and B, we have independence if  $P(A|B) = P(A)$  and  $P(B|A) = P(B)$ .

Let's see if the probability of selecting a green ball at random (B) is independent of the container that it is from (A). We can use the two equations below in (7).

$$P(A|B) = \frac{P(A \cap B)}{P(B)} \quad (7)$$

$$P(B|A) = \frac{P(B \cap A)}{P(A)}$$

```

1 p_A_given_B = 0.36/0.8 # Probability of being in container I given a green ball
2 p_B_given_A = 0.36/0.43 # Probability of a green ball given that it was chosen
3 p_A = 0.43 # Probability of a green ball
4 p_B = 0.8 # Probability of being in container I

```

```
1 p_A_given_B == p_A
```

```
False
```

```
1 p_B_given_A == p_B
```

```
False
```

These outcomes are not independent.

If we were to select a ball at random, what is the probability that it is red (denoted as A) **OR** is in container II (B)? Here we are dealing with the addition of probabilities. For this we can use equation (8) for two probabilities A and B.

$$P(A \cup B) = P(A) + P(B) - P(A \cap B) \quad (8)$$

```

1 p_red = 0.26 + 0.09
2 p_II = 0.09 + 0.07 + 0.04
3 p_red_II = 0.09
4
5 p_red + p_II - p_red_II

0.46000000000000001

```

There is a 46% probability that ball will either be red or in container II.

What about the probability that our random ball is both red **AND** is from container II. This requires the multiplication of the probabilities. We can use either of the following rearrangement of conditional probability, shown in (9).

$$P(A \cap B) = P(A|B) P(B) \quad (9)$$

$$P(A \cap B) = P(B|A) P(A)$$

```

1 p_red_given_II = 0.09 / 0.2 # PROBABILITY OF A RED BALL GIVEN IT IS CHOSEN FROM
2 p_II = 0.2 # Probability of container II
3 p_red_given_II * p_II

0.09

```

We note that it is 9%. It is nothing but the relative frequency.

The complement of a probability, i.e. of a specified event not occurring is shown in (10).

$$P(A^c) = 1 - P(A) \quad (10)$$

## ▼ EXAMPLE PROBLEM

Imagine three cards. One is red on both sides, one is white on both sides, and one is red on one side and red on the other. If a card is chosen at random and one side is shown to you and it is red, what is the probability that the card's other side is red?

Stop and think before reading on. What do you think the probability is?

Consider all the probabilities. They are listed below. We use the letters R for red and W for white. The first letter in every pair is the side facing you.

- RR
- RR
- RW
- WR
- WW
- WW

It is important to remember that the all red and all white cards can be shown to you in two different orientations. We then note that being shown one red face means that only the first three cases above apply. We note that for the three remaining possibilities, we have two red and a single wide side at the back. The probability that the other side is red is therefor two-thirds.

## ▼ RANDOM VARIABLES

In this and previous notebooks, we view a statistical variable as the name of a column in a spreadsheet. The data point values that we capture for a specific variable has a certain type, i.e.

categorical or numerical. This view simplifies our later use of data analysis in Python. There are some deeper subtleties, though, some of which we will briefly touch upon.

A **random variable** is a function that maps the outcome of an experiment to a number. Imagine that we are rolling two normal dice and we add up the values that land face up. The minimum value is two and the maximum is 12. Let's see all the outcomes (summed over each die):

1.  $1 + 1 = 2$
2.  $1 + 2 = 3$
3.  $2 + 1 = 3$
4.  $2 + 2 = 4$
5.  $1 + 3 = 4$
6.  $3 + 1 = 4$
7.  $2 + 3 = 5$
8.  $3 + 2 = 5$
9.  $1 + 4 = 5$
10.  $4 + 1 = 5$
11.  $3 + 3 = 6$
12.  $2 + 4 = 6$
13.  $4 + 2 = 6$
14.  $1 + 5 = 6$
15.  $5 + 1 = 6$
16.  $3 + 4 = 7$
17.  $4 + 3 = 7$
18.  $2 + 5 = 7$
19.  $5 + 2 = 7$
20.  $1 + 6 = 7$
21.  $6 + 1 = 7$
22.  $4 + 4 = 8$
23.  $3 + 5 = 8$
24.  $5 + 3 = 8$
25.  $2 + 6 = 8$
26.  $6 + 2 = 8$
27.  $4 + 5 = 9$
28.  $5 + 4 = 9$
29.  $3 + 6 = 9$
30.  $6 + 3 = 9$
31.  $5 + 5 = 10$
32.  $4 + 6 = 10$
33.  $6 + 4 = 10$
34.  $5 + 6 = 11$

$$35. 6 + 5 = 11$$

$$36. 6 + 6 = 12$$

If we make the random variable  $X$  (in a spreadsheet it would be a column header such as `sum_of_two_dice`), we see the sample space as 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, and 12. Every time we run our experiment (rolling the dice) we get a random variable (one of the 11 sample space elements).

We say that all the totals that sum to 2 *maps*  $X$  to 2. All the values that sum to 3 *maps*  $X$  to 3, and so on.

Some of the elements are more likely to occur. Below we see the probabilities for the 11 elements:

$$P(X = 2) \rightarrow \frac{1}{36}$$

$$P(X = 3) \rightarrow \frac{2}{36}$$

$$P(X = 4) \rightarrow \frac{3}{36}$$

$$P(X = 5) \rightarrow \frac{4}{36}$$

$$P(X = 6) \rightarrow \frac{5}{36}$$

$$P(X = 7) \rightarrow \frac{6}{36}$$

$$P(X = 8) \rightarrow \frac{5}{36}$$

$$P(X = 9) \rightarrow \frac{4}{36}$$

$$P(X = 10) \rightarrow \frac{3}{36}$$

$$P(X = 11) \rightarrow \frac{2}{36}$$

$$P(X = 12) \rightarrow \frac{1}{36}$$

(These are based on how many ways there are to get to a specific summed value.)

We can use list comprehension to calculate the 11 probabilities.

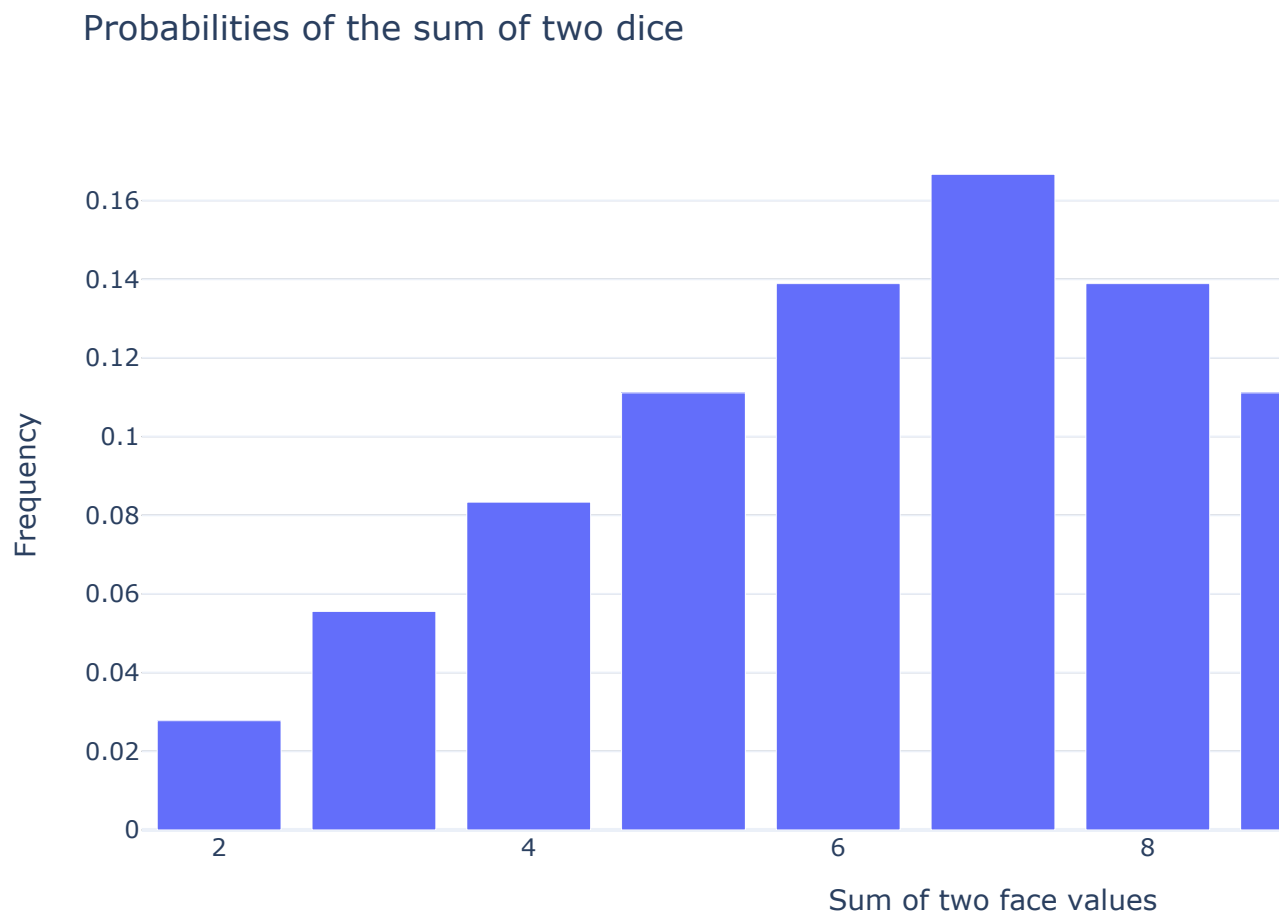
```
1 [i / 36 for i in [1, 2, 3, 4, 5, 6, 5, 4, 3, 2, 1]]

[0.027777777777777776,
 0.055555555555555555,
 0.083333333333333333,
 0.111111111111111111,
 0.13888888888888889,
 0.166666666666666666,
 0.13888888888888889,
 0.111111111111111111,
```

```
0.08333333333333333,
0.05555555555555555,
0.027777777777777776]
```

We have a here an **theoretical distribution**, the theoretical pattern of frequencies of random events.

```
1 px.bar(x=[2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12],
2       y=[i / 36 for i in [1, 2, 3, 4, 5, 6, 5, 4, 3, 2, 1]],
3       title='Probabilities of the sum of two dice',
4       labels={'x':'Sum of two face values', 'y':'Frequency'})
```



We note that it is most likely that you roll a seven and (equally) least likely to roll a 2 or a 12.

Since these probabilities are mutually exclusive events, we can sum the probabilities.

The probability that  $X$  is more than or equal to 10 is thus

$P(X \geq 10) = 0.083 + 0.055 + 0.028 = 0.166$ . There is a 16.6% probability of rolling a 10 or more.

```
1 0.083 + 0.055 + 0.028
```




0.166

Now that we know more about the theoretical distribution, let's simulate our experiment 10000 times and look at an **empirical distribution** (of the actual random values).

```
1 np.random.seed(3)
2 roll_totals = pd.DataFrame({'RollTotal':[np.sum(np.random.randint(1, high=7, size=5)) for _ in range(10000)]})

1 roll_totals.head()
```

1 to 5 of 5 entries  

index	RollTotal
0	4
1	6
2	2
3	7
4	10

Show  per page

We can use the `value_counts` method to see how close we get to the theoretical probabilities.

```
1 roll_totals.RollTotal.value_counts(normalize=True)
```

```
7      0.1699
6      0.1405
8      0.1337
5      0.1111
9      0.1047
10     0.0854
4      0.0853
11     0.0559
3      0.0554
12     0.0299
2      0.0282
Name: RollTotal, dtype: float64
```

The `sort_index` method is a way to get our index in order.

```
1 roll_totals.RollTotal.value_counts().sort_index()
```

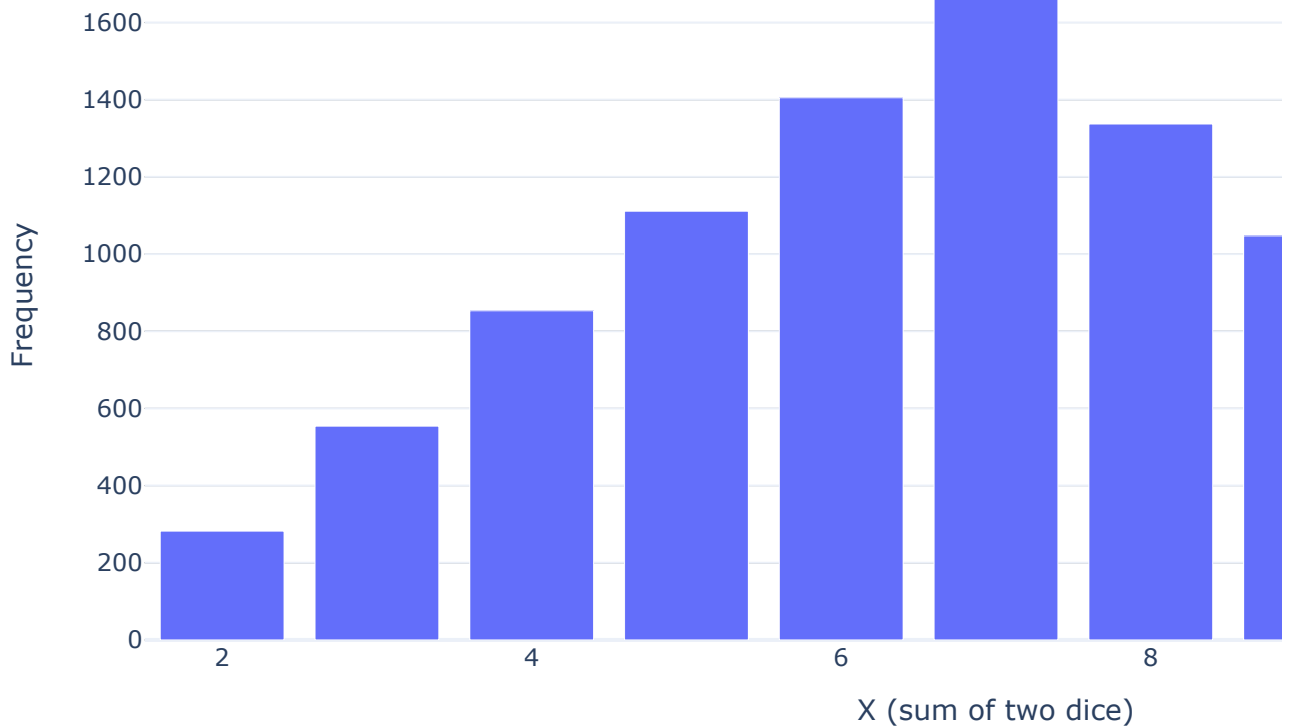
```
2      282
3      554
4      853
5     1111
6     1405
7     1699
8     1337
9     1047
10     854
```

```
11     559
12     299
Name: RollTotal, dtype: int64
```

Not too bad. Below, we create a bar graph to view all the results.

```
1 px.bar(x=range(2, 13),
2         y=roll_totals.RollTotal.value_counts().sort_index().tolist(),
3         title='Bar chart showing frequency of each total',
4         labels={'x': 'X (sum of two dice)', 'y': 'Frequency'}).show()
```

Bar chart showing frequency of each total



This is what random variables are all about. In these notebooks, though, we will use the terms **variable** or **statistical variable** to refer to the name of the variable. Each row in our dataset will refer to a subject in a study (making each row an observation). Each data point value collected for a variable for a specific subject will be a random value from the sample space of the statistical variable.

An unbiased random selection of subjects where the outcome (value of a statistical variable) is only determined by the variable under consideration (which is another variable) is always the aim. This is not always possible. In real life, it is very easy to introduce bias into a study. The values that we capture are not guaranteed to be random at all! It is also common for other

variable to have an influence on the outcome variable. If this variable is not included in the analysis it is termed a **confounding factor**.

## ▼ DISTRIBUTIONS

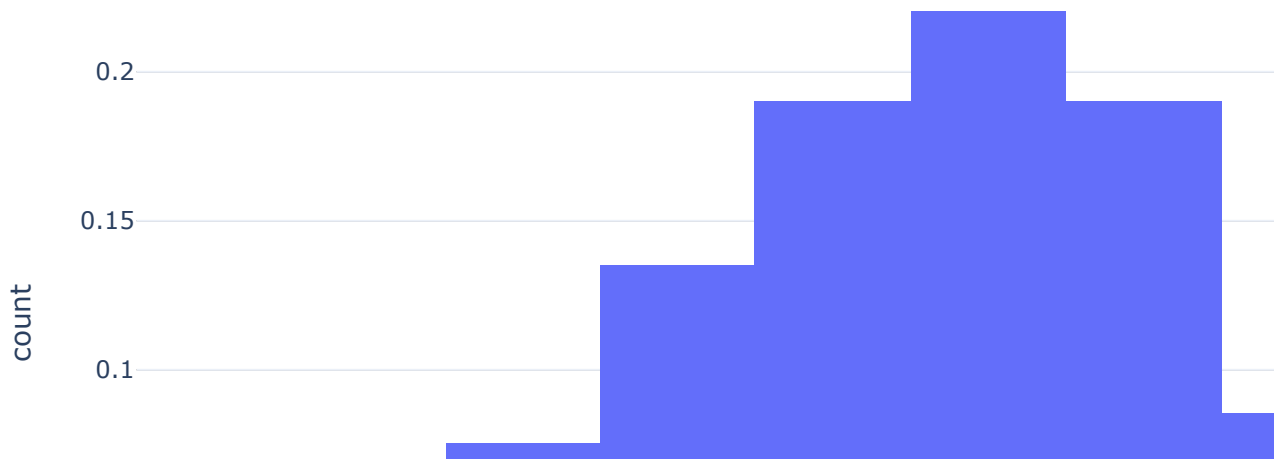
Very simply stated, a **distribution** is a pattern of a list of the data values captured for some variable. Below, we create a list of pseudo-random values representing the height (in cm) of 200 women (taken from a normal distribution, which we will learn about later in this notebook). The stats module from the scipy package has a `norm` keyword that refers to the normal distribution. The `rvs` function generates random values. We set the argument `loc` (the mean) to 160, the `scale` (standard deviation) to 10, and specify that we want 200 samples, using the `size` argument. The `rvs` function has an in built `random_state` arguement that allows us to seed the pseudo-random number generator in scipy.

```
1 # Using the stats module in the scipy package
2 height = stats.norm.rvs(loc=160,
3                           scale=10,
4                           size=200,
5                           random_state=1)
```

We can create a histogram of relative frequencies of this continuous numerical variable. The relative frequency histogram visualizes a probability distribution (the distribution of probabilities). This is done instead of the default frequency (absolute count) histogram and is achieved adding the `histnorm` argument and setting it to `probability`.

```
1 px.histogram(
2     x=height,
3     histnorm='probability',
4     title='Probability distribution',
5     labels={
6         'x': 'Height',
7         'y': 'Relative frequency'
8     }
9 )
```

## Probability distribution



We can take this opportunity to introduce the concept of a distribution plot. Here, the geometric area of each rectangle (base times height) in the histogram represents the probability of a value in that rectangle's interval. The `create_distplot` function from the `figure_factory` module in `plotly` creates a probability density. It adds a *smooth* version, called a kernel density estimate.

```
1 height_hist = ff.create_distplot([height],
2                                 ['Height'],
3                                 bin_size=5,
4                                 show_curve=True)
5
6 height_hist.update_layout(title='Distribution plot of heights')
7
8 height_hist.show()
```

## Distribution plot of heights



We see a probability density histogram and a rug plot (individual data point values). The visualizations show us that there are more data point values (heights) close to 160 and the further we get from 160, the less likely it is to find a specific value. This represents a distribution of the density of the values.



What we see above is an example of an **empirical distribution**. This is the distribution of collected values from our sample of individuals.

A **theoretical distribution** is designed through mathematical reason and construction. The **Law of averages** states that as we repeat an experiment over and over again, an empirical distribution approximates a theoretical distribution.

170 150 100 170

### ▼ SAMPLING FROM A POPULATION

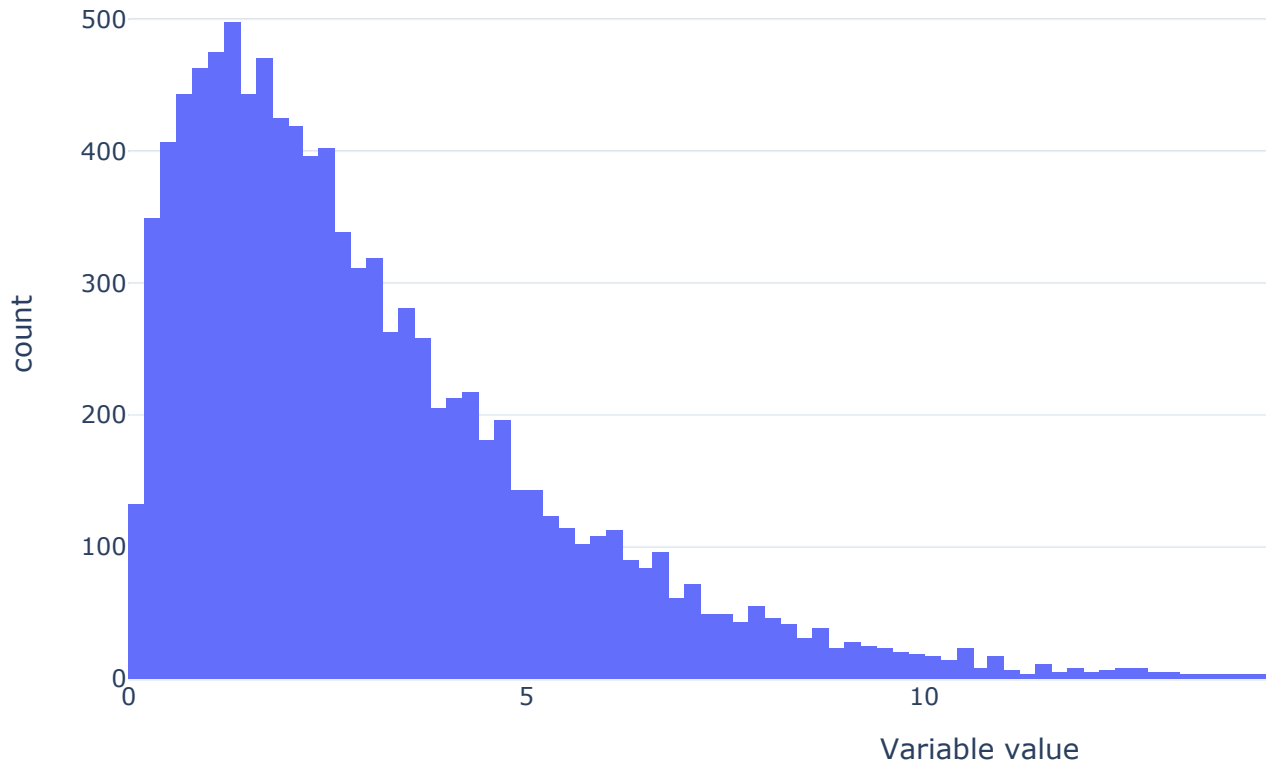
In many research projects, samples are selected from a population. This makes the projects viable with respect to cost, time consumed, and human resources available to do the projects. It is most often impossible to collect data from a whole population unless that population is a well defined scarce resource. Examples of the latter include a very specific genetic mutation created in a laboratory and not existing anywhere else or a set of very rare astronomical events.

Below we simulate data for a variable for a population of 10000 subjects and then visualize the dataset with a histogram. The  $\chi^2$  distribution is used here.

```
1 np.random.seed(42)
2 population = np.round(np.random.chisquare(3, 10000), 1)

1 px.histogram(
2     x=population,
3     title='Histogram of variable for complete population',
4     labels={
5         'x': 'Variable value',
6         'y': 'Frequency'
7     }
8 )
```

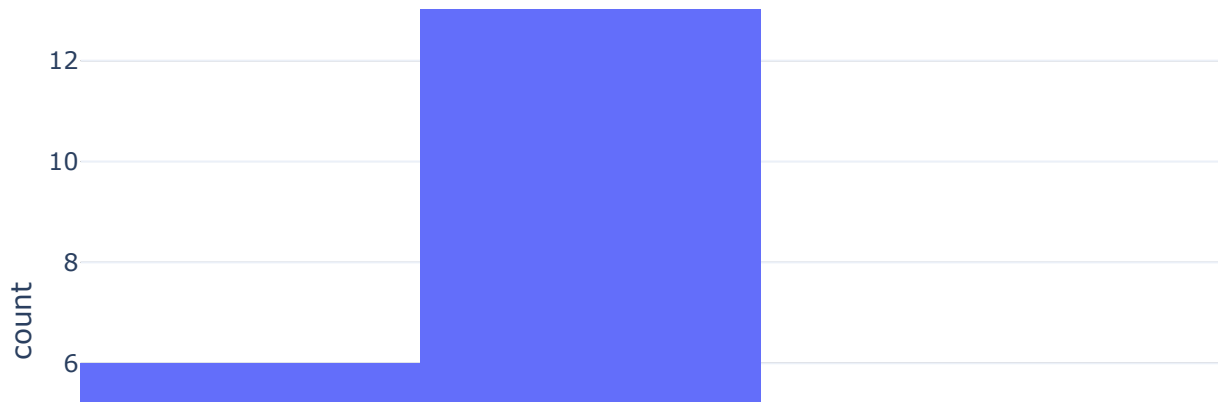
## Histogram of variable for complete population



We can select random samples from the population using the `choice` function. Below, we simulate selecting 30 individuals and show a histogram of the data.

```
1 px.histogram(  
2     x=np.random.choice(  
3         population,  
4         size=30  
5     ),  
6     title='Frequency chart of 30 samples',  
7     labels={  
8         'x':'Variable value'  
9     }  
10 )
```

## Frequency chart of 30 samples



Note the difference between the population and the sample when it comes to values that were rare in the population.



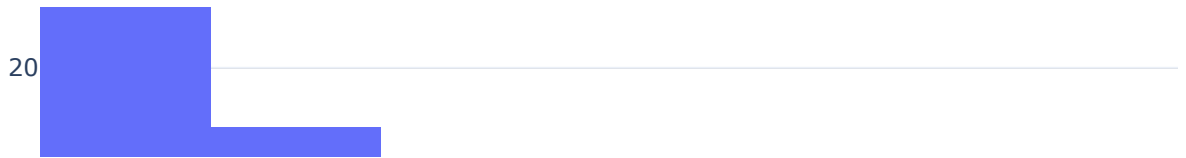
The larger the sample size, the closer it resembles the population. Below, we sample 100 subjects.

```

1 px.histogram(
2     x=np.random.choice(
3         population,
4         size=100
5     ),
6     title='Frequency chart of 100 samples',
7     labels={
8         'x':'Variable value'
9     }
10 )

```

## Frequency chart of 100 samples



What about 1000 samples.

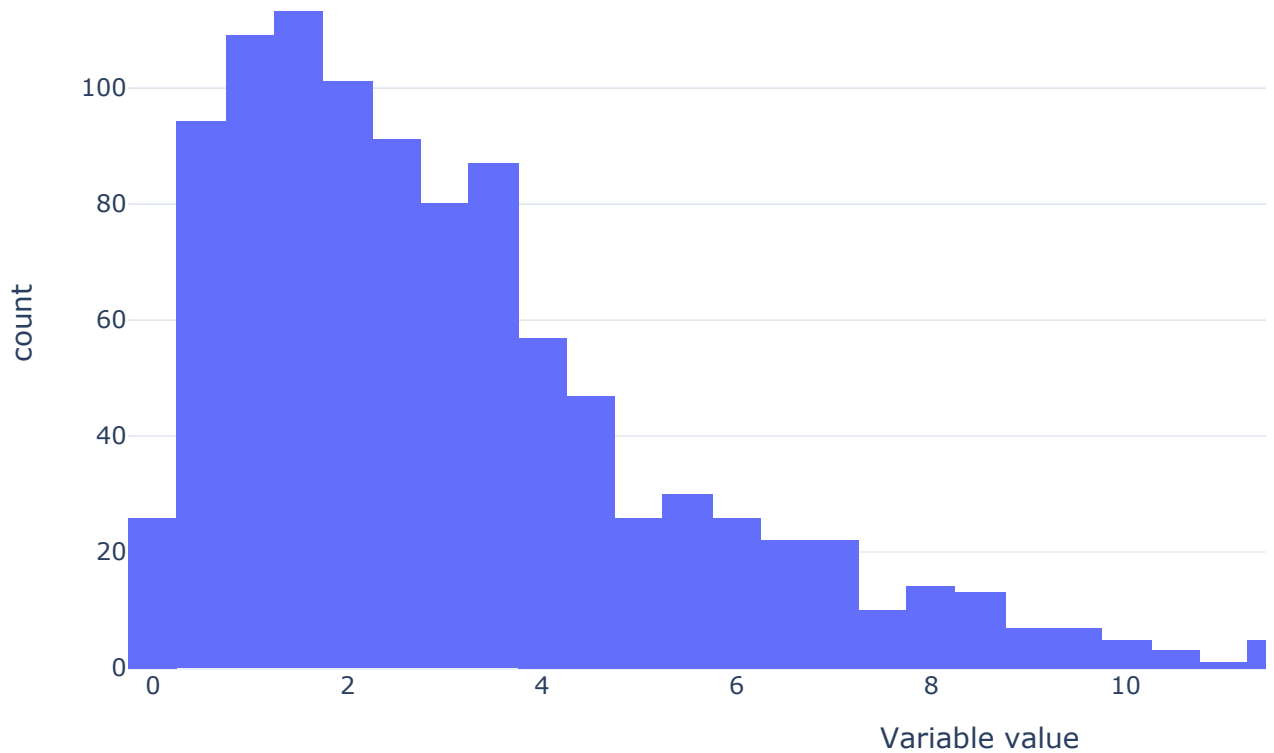


```

1 px.histogram(
2     x=np.random.choice(
3         population,
4         size=1000
5     ),
6     title='Frequency chart of 1000 samples',
7     labels={
8         'x':'Variable value'
9     }
10 )

```

## Frequency chart of 1000 samples





The sample starts to approximate the population, with the law of averages at play. A larger sample size is advantages when it comes to analysis and inference (inferring the results of the sample to the greater population).

## ▼ SAMPLING DISTRIBUTIONS

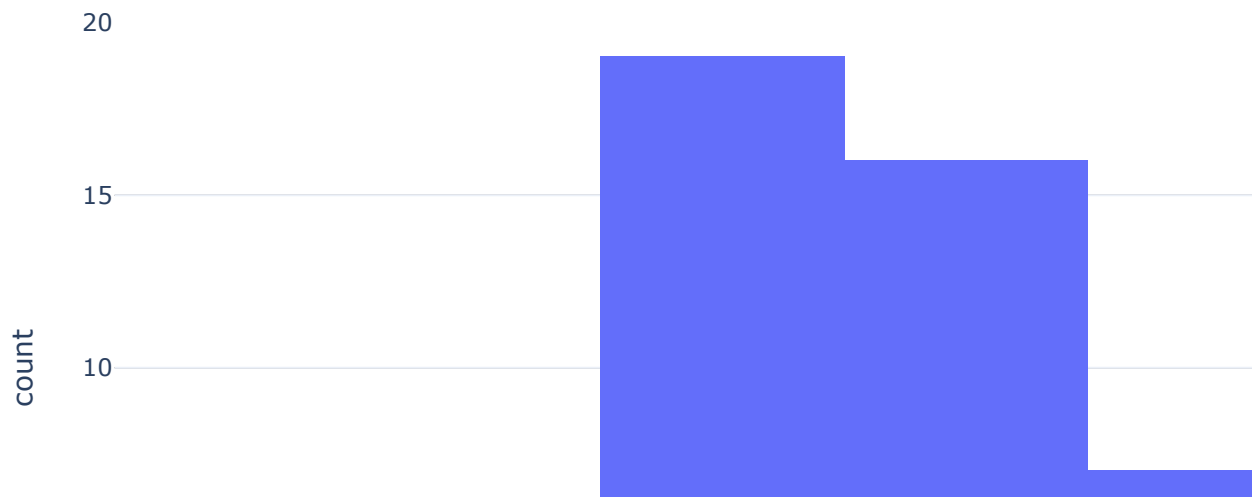
Once we take a sample from a population and capture values for a variable, we can summarize the random variables (data values). We studied many such statistics in a previous notebook. Here, we consider the mean. Below, we calculate the mean from a single sample of 30 individuals from our population created above.

```
1 np.mean(np.random.choice(  
2     population,  
3     30  
4 ))  
  
3.1833333333333333
```

Since we have access to a powerful computer language and computational resources, we can simulate taking repeated samples. At every repeat of the experiment, we can calculate a mean and record it in a list. We do this 50 times using list comprehension. We refer to this list of means as **sampling means**. The distribution of all these means are an example of a **sampling distribution**. Other statistics also have sampling distributions. Below, we visualize it as a histogram.

```
1 mean_50 = [np.mean(np.random.choice(population, 30)) for i in range(50)]  
  
1 px.histogram(  
2     x=mean_50,  
3     title='Sampling distribution of 50 means',  
4     labels={  
5         'x': 'Mean values'  
6     }  
7 )
```

## Sampling distribution of 50 means

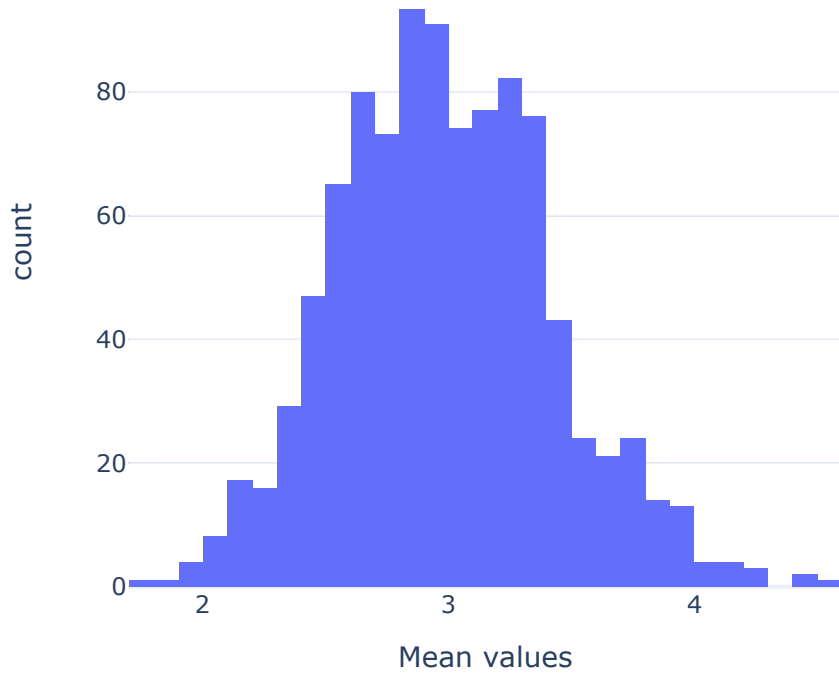


We can simulate even more repeat studies. Below, it is done 1000 times. These distributions of the mean values are empirical distributions.

```
1 mean_1000 = [np.mean(np.random.choice(population, 30)) for i in range(1000)]
2
3 px.histogram(
4     x=mean_1000,
5     title='Sampling distribution of 1000 means',
6     labels={
7         'x': 'Mean values'
8     }
9 )
```

### Sampling distribution of 1000 means

1



✓ 0s completed at 13:58

