Statistics in Python

What is statistics?

- The field of statistics – the practice and study of collecting and analyzing data.
- A summary statistics – a fact about or summary of some data.
- What can statistics do?
  - How likely is someone to purchase a product? Are people more likely to purchase it if they can use a different payment system?
  - How many occupants will your hotel have? How can you optimize occupancy?
  - How many sizes of jeans need to be manufactured so they can fit 95% of the population? Should the same number of each size be produced?
  - A/B tests: Which ad is more effective in getting people to purchase a product?

What can’t statistics do?

- Why is Game of Thrones so popular?

Instead…

- Are series with more violent scenes viewed by more people?

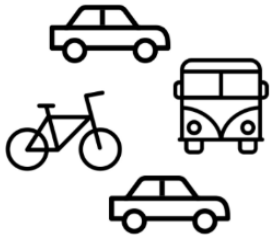
But…

- Even so, this can’t tell us if more violent scenes lead to more views.

Types of statistics

Descriptive statistics

- *Describe* and summarize data



- 50% of friends drive to work
- 25% take the bus
- 25% bike

Inferential statistics

- Use a sample of data to make *inferences* about a larger population

What percent of people drive to work?

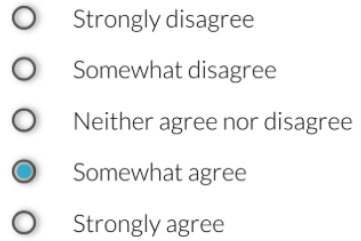
Types of data

Numeric (Quantitative)

- **Continuous (Measured)**
  - Airplane speed
  - Time spent waiting in line
- **Discrete (Counted)**
  - Number of pets
  - Number of packages shipped

Categorical (Qualitative)

- **Nominal (Unordered)**
  - Married/unmarried
  - Country of residence
- **Ordinal (Ordered)**



Categorical data can be represented as numbers

Nominal (Unordered)

- Married/unmarried (`1` / `0`)
- Country of residence (`1` , `2` , ...)

Ordinal (Ordered)

- Strongly disagree (`1`)
- Somewhat disagree (`2`)
- Neither agree nor disagree (`3`)
- Somewhat agree (`4`)
- Strongly agree (`5`)

Being able to identify data types is important since the type of data you're working with will dictate what kinds of summary statistics and visualizations make sense for your data, so this is an important skill to master. For numerical data, we can use summary statistics like mean, and plots like scatter plots, but these don't make a ton of sense for categorical data.

Descriptive	Inferential

Given data on every customer service request made, what's the average time it took to respond?

Given data on all 100,000 people who viewed an ad, what percent of people clicked on it?

After interviewing 100 customers, what percent of *all* your customers are satisfied with your product?

Given data on 20 fish caught in a lake, what's the average weight of all fish in the lake?

Measures of center

- Consider the following data about different mammals' sleep habits.

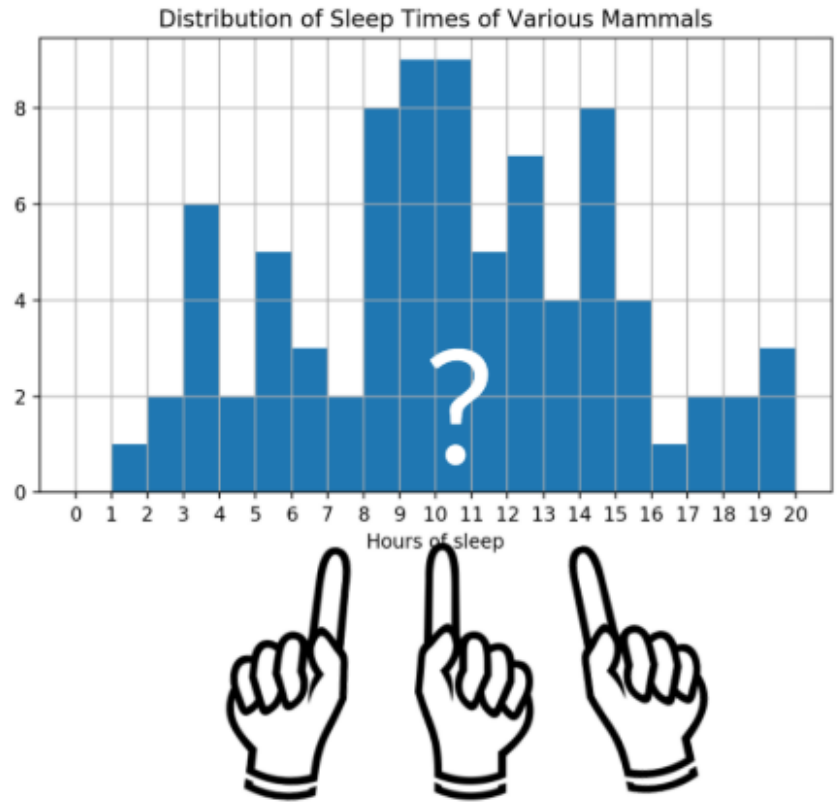
	name	genus	vore	order	...	sleep_cycle	awake	brainwt	bodywt
1	Cheetah	Acinonyx	carni	Carnivora	...	NaN	11.9	NaN	50.000
2	Owl monkey	Aotus	omni	Primates	...	NaN	7.0	0.01550	0.480
3	Mountain beaver	Aplodontia	herbi	Rodentia	...	NaN	9.6	NaN	1.350
4	Greater short-ta...	Blarina	omni	Soricomorpha	...	0.133333	9.1	0.00029	0.019
5	Cow	Bos	herbi	Artiodactyla	...	0.666667	20.0	0.42300	600.000
..	...	...	...	...	...	...	...	...	...
79	Tree shrew	Tupaia	omni	Scandentia	...	0.233333	15.1	0.00250	0.104
80	Bottle-nosed do...	Tursiops	carni	Cetacea	...	NaN	18.8	NaN	173.330
81	Genet	Genetta	carni	Carnivora	...	NaN	17.7	0.01750	2.000
82	Arctic fox	Vulpes	carni	Carnivora	...	NaN	11.5	0.04450	3.380
83	Red fox	Vulpes	carni	Carnivora	...	0.350000	14.2	0.05040	4.230

How long do mammals in the dataset typically sleep?

What's a typical value?

Where is the center of the data?

- Mean
- Median
- Mode



Measures of center: mean

	name	sleep_total
1	Cheetah	12.1
2	Owl monkey	17.0
3	Mountain beaver	14.4
4	Greater short-t...	14.9
5	Cow	4.0
..	...	...

```
import numpy as np
np.mean(msleep['sleep_total'])
```

```
10.43373
```

$$\text{Mean sleep time} = \frac{12.1 + 17.0 + 14.4 + 14.9 + \ldots}{83} = 10.43$$

Measures of center: median

```
msleep['sleep_total'].sort_values()
```

```
29     1.9
30     2.7
22     2.9
9      3.0
23     3.1
      ...
19    18.0
61    18.1
36    19.4
21    19.7
42    19.9
```

```
msleep['sleep_total'].sort_values().iloc[41]
```

```
10.1
```

```
np.median(msleep['sleep_total'])
```

```
10.1
```

Measures of center: mode

Most frequent value

```
msleep['sleep_total'].value_counts()
```

```
12.5    4
10.1    3
14.9    2
11.0    2
8.4     2
...
14.3    1
17.0    1
Name: sleep_total, Length: 65, dtype: int64
```

```
msleep['vore'].value_counts()
```

```
herbi      32
omni       20
carni      19
insecti     5
Name: vore, dtype: int64
```

```
import statistics
statistics.mode(msleep['vore'])
```

```
'herbi'
```

Effects of outliers

```
msleep[msleep['vore'] == 'insecti']
```

	name	genus	vore	order	sleep_total
22	Big brown bat	Eptesicus	insecti	Chiroptera	19.7
43	Little brown bat	Myotis	insecti	Chiroptera	19.9
62	Giant armadillo	Priodontes	insecti	Cingulata	18.1
67	Eastern american mole	Scalopus	insecti	Soricomorpha	8.4

```
msleep[msleep['vore'] == "insecti"]['sleep_total'].agg([np.mean, np.median])
```

```
mean      16.53
median    18.9
Name: sleep_total, dtype: float64
```

```
msleep[msleep['vore'] == 'insecti']
```

```
                   name         genus     vore         order  sleep_total
22          Big brown bat     Eptesicus  insecti    Chiroptera         19.7
43       Little brown bat        Myotis  insecti    Chiroptera         19.9
62        Giant armadillo    Priodontes  insecti     Cingulata         18.1
67  Eastern american mole      Scalopus  insecti  Soricomorpha          8.4
84    Mystery insectivore           ...  insecti           ...          0.0
```

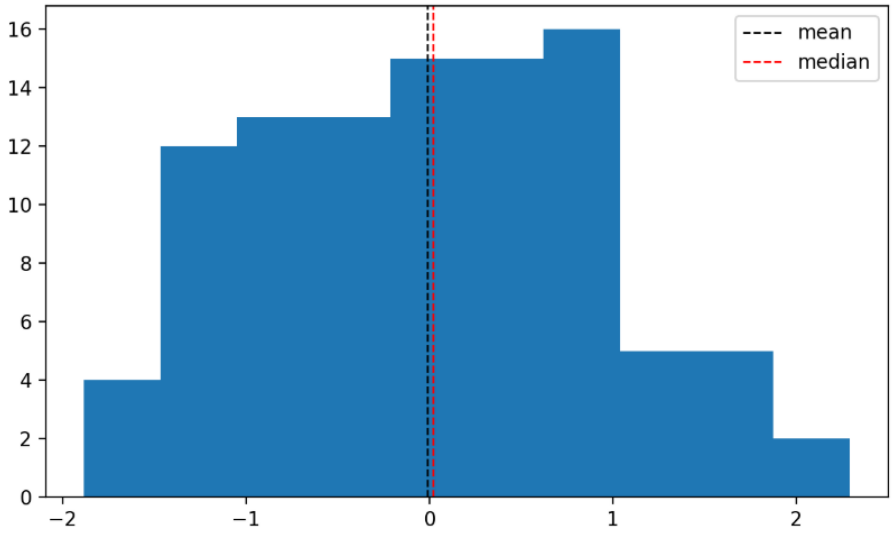
```
msleep[msleep['vore'] == "insecti"]['sleep_total'].agg([np.mean, np.median])
```

```
mean      13.22
median    18.1
Name: sleep_total, dtype: float64
```

Mean: 16.5 → 13.2

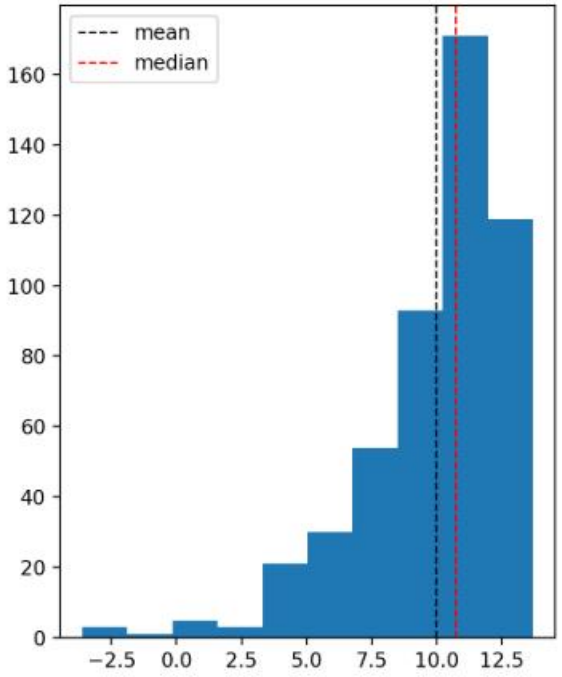
Median: 18.9 → 18.1

Which measure (mean vs. median) to use?

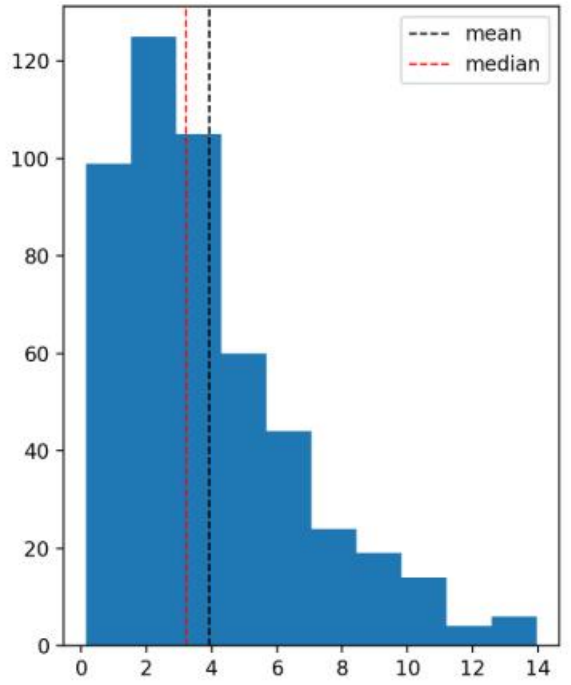


When data is skewed, the mean and median are different. The mean is pulled in the direction of the skew, so it's lower than the median on the left-skewed data, and higher than the median on the right-skewed data. Because the mean is pulled around by the extreme values, it's better to use the median since it's less affected by outliers.

Left-skewed



Right-skewed



Try

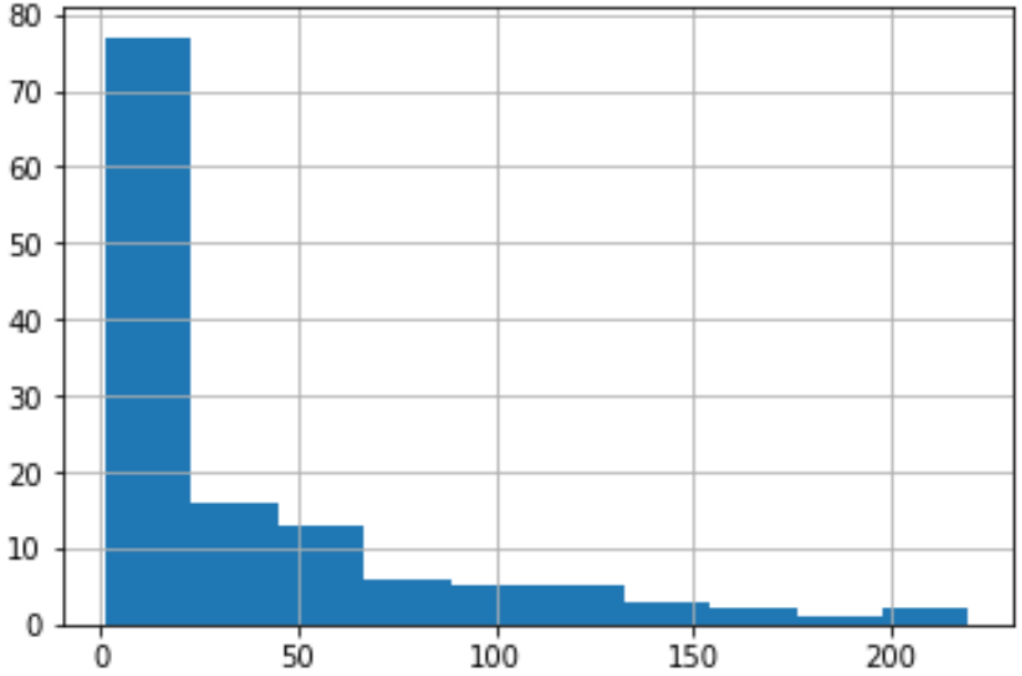
- The food_consumption dataset contains information about the kilograms of food consumed per person per year in each country in each food category (consumption) as well as information about the carbon footprint of that food category (co2_emissions) measured in kilograms of carbon dioxide, or CO2, per person per year in each country.
- In this exercise, you'll compute measures of center to compare food consumption in the US and Belgium using your pandas and numpy skills.
- Instructions#1
  - Create two DataFrames: one that holds the rows of food_consumption for 'Belgium' and another that holds rows for 'USA'. Call these be_consumption and usa_consumption.
  - Calculate the mean and median of kilograms of food consumed per person per year for both countries.
- Instructions#2
  - Subset food_consumption for rows with data about Belgium and the USA.
  - Group the subsetted data by country and select only the consumption column.
  - Calculate the mean and median of the kilograms of food consumed per person per year in each country using .agg().

```
Unnamed: 0      402.000
consumption      42.133
co2_emission    100.190
dtype: float64
Unnamed: 0      402.00
consumption      12.59
co2_emission     21.34
dtype: float64
Unnamed: 0       61.00
consumption      44.65
co2_emission    156.26
dtype: float64
Unnamed: 0      61.00
consumption     14.58
co2_emission    15.34
dtype: float64
```

country	mean	median
Belgium	42.132727	12.59
USA	44.650000	14.58

Mean vs. Median

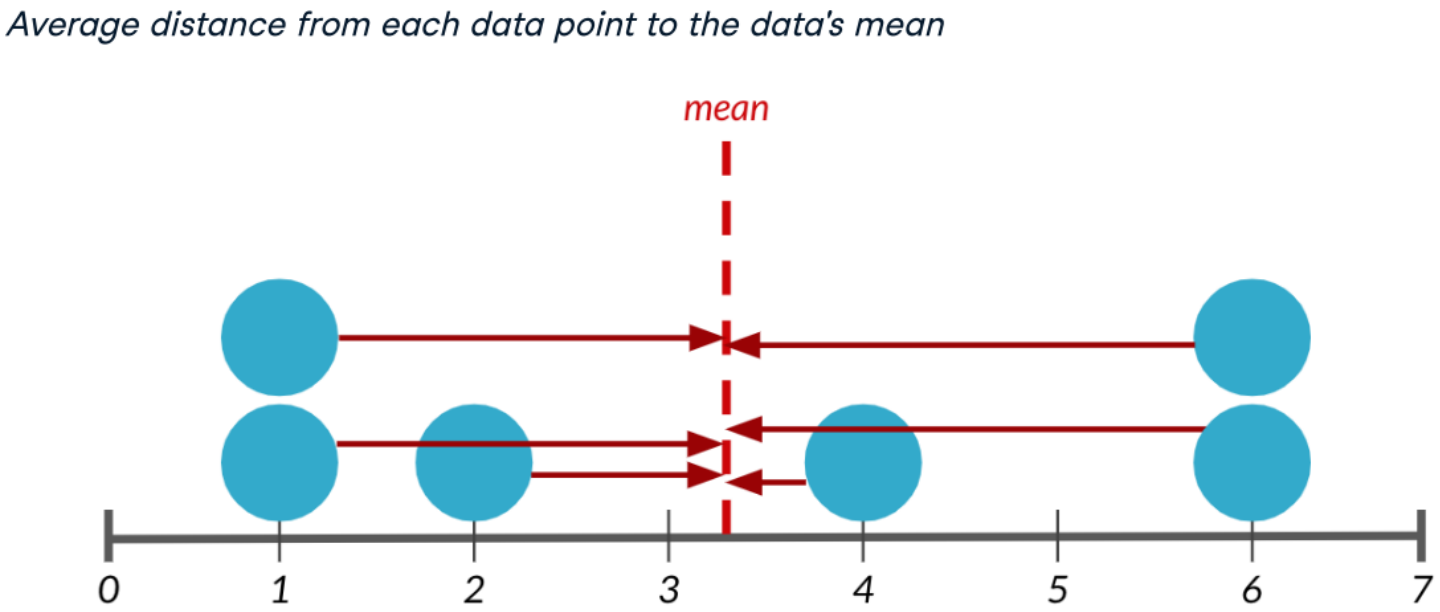
- Import matplotlib.pyplot with the alias plt.
- Subset food_consumption to get the rows where food_category is 'rice', and save the result to rice_consumption.
- Create a histogram of co2_emission for rice and show the plot.
- Use .agg() to calculate the mean and median of co2_emission for rice.



Left- or Right-Skewed?

Measures of spread

- What is spread?
  - Spread is just what it sounds like - it describes how spread apart or close together the data points are. Just like measures of center, there are a few different measures of spread.
- Variance - measures the average distance from each data point to the data's mean.



Calculating variance

1. Subtract mean from each data point

```
dists = msleep['sleep_total'] -
        np.mean(msleep['sleep_total'])
print(dists)
```

```
0     1.666265
1     6.566265
2     3.966265
3     4.466265
4    -6.433735
      ...
```

2. Square each distance

```
sq_dists = dists ** 2
print(sq_dists)
```

```
0      2.776439
1     43.115837
2     15.731259
3     19.947524
4     41.392945
      ...
```

Calculating variance

3. Sum squared distances

```
sum_sq_dists = np.sum(sq_dists)
print(sum_sq_dists)
```

```
1624.065542
```

4. Divide by number of data points - 1

```
variance = sum_sq_dists / (83 - 1)
print(variance)
```

```
19.805677
```

We can calculate the variance in one step using np.var, setting the ddof argument to 1. If we don't specify ddof equals 1, a slightly different formula is used to calculate variance that should only be used on a full population, not a sample.

Use `np.var()`

```
np.var(msleep['sleep_total'], ddof=1)
```

```
19.805677
```

*Without `ddof=1` , **population variance** is calculated instead of **sample variance**:*

```
np.var(msleep['sleep_total'])
```

```
19.567055
```

Standard deviation

- The standard deviation is another measure of spread, calculated by taking the square root of the variance. It can be calculated using np.std. Just like np.var, we need to set ddof to 1. The nice thing about standard deviation is that the units are usually easier to understand since they're not squared. It's easier to wrap your head around 4 and a half hours than 19-point-8 hours squared.

Quantiles

- Quantiles, also called percentiles, split up the data into some number of equal parts.
- Here, we call np.quantile, passing in the column of interest, followed by point-5. This gives us 10.1 hours, so 50% of mammals in the dataset sleep less than 10.1 hours a day, and the other 50% sleep more than 10.1 hours, so this is exactly the same as the median.

```
np.quantile(msleep['sleep_total'], 0.5)
```

```
10.1
```

0.5 quantile = median

Quartiles

- We can also pass in a list of numbers to get multiple quantiles at once. Here, we split the data into 4 equal parts. These are also called quartiles. This means that 25% of the data is between 1.9 and 7.85, another 25% is between 7.85 and 10.10, and so on.

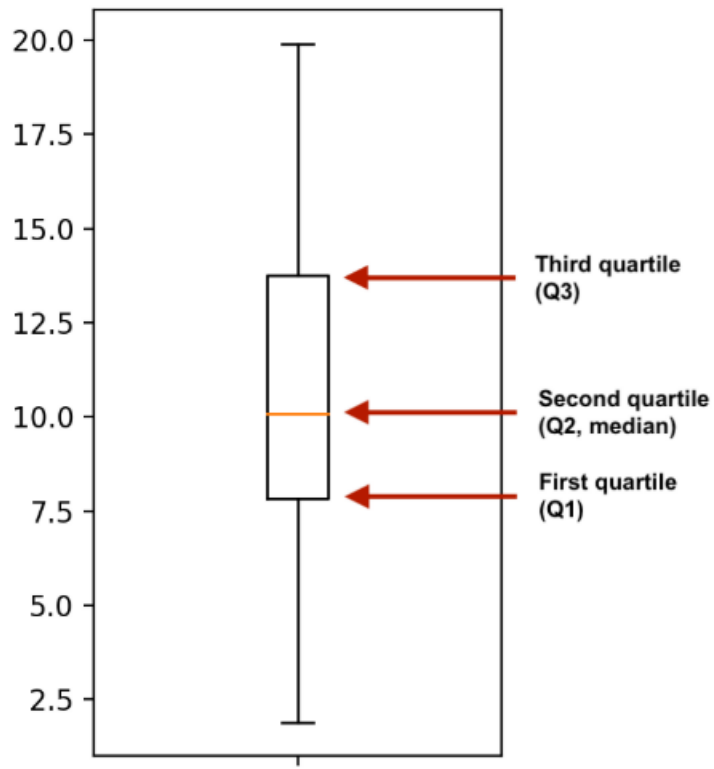
Quartiles:

```
np.quantile(msleep['sleep_total'], [0, 0.25, 0.5, 0.75, 1])
```

```
array([ 1.9 ,  7.85, 10.1 , 13.75, 19.9 ])
```

Boxplots use quartiles

```
import matplotlib.pyplot as plt
plt.boxplot(msleep['sleep_total'])
plt.show()
```



Quantiles using np.linspace()

```
np.quantile(msleep['sleep_total'], [0, 0.2, 0.4, 0.6, 0.8, 1])
```

```
array([ 1.9 ,  6.24,  9.48, 11.14, 14.4 , 19.9 ])
```

`np.linspace(start, stop, num)`

```
np.quantile(msleep['sleep_total'], np.linspace(0, 1, 5))
```

```
array([ 1.9 ,  7.85, 10.1 , 13.75, 19.9 ])
```

Interquartile range (IQR)

Height of the box in a boxplot

```
np.quantile(msleep['sleep_total'], 0.75) - np.quantile(msleep['sleep_total'], 0.25)
```

```
5.9
```

```
from scipy.stats import iqr
iqr(msleep['sleep_total'])
```

```
5.9
```

Outliers

Outlier: data point that is substantially different from the others

How do we know what a substantial difference is? A data point is an outlier if:

- $\text{data} < \text{Q1} - 1.5 \times \text{IQR}$ or
- $\text{data} > \text{Q3} + 1.5 \times \text{IQR}$

Identifying thresholds

```
# 75th percentile
seventy_fifth = salaries["Salary_USD"].quantile(0.75)

# 25th percentile
twenty_fifth = salaries["Salary_USD"].quantile(0.25)

# Interquartile range
salaries_iqr = seventy_fifth - twenty_fifth

print(salaries_iqr)
```

```
from scipy.stats import iqr
iqr = iqr(msleep['bodywt'])
lower_threshold = np.quantile(msleep['bodywt'], 0.25) - 1.5 * iqr
upper_threshold = np.quantile(msleep['bodywt'], 0.75) + 1.5 * iqr
```

```
msleep[(msleep['bodywt'] < lower_threshold) | (msleep['bodywt'] > upper_threshold)]
```

```
                   name   vore  sleep_total    bodywt
4                   Cow  herbi          4.0   600.000
20        Asian elephant  herbi          3.9  2547.000
22                 Horse  herbi          2.9   521.000
...
```

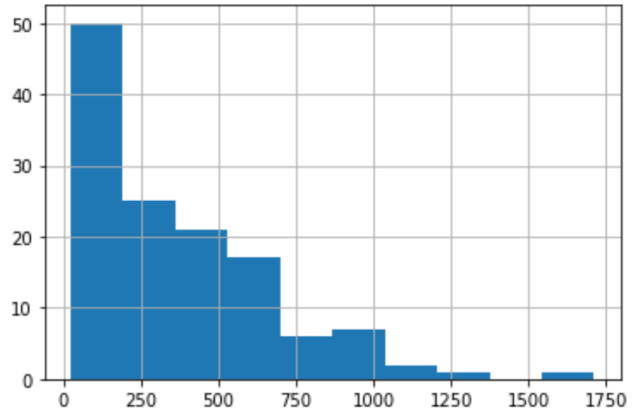
Complete the following tasks.

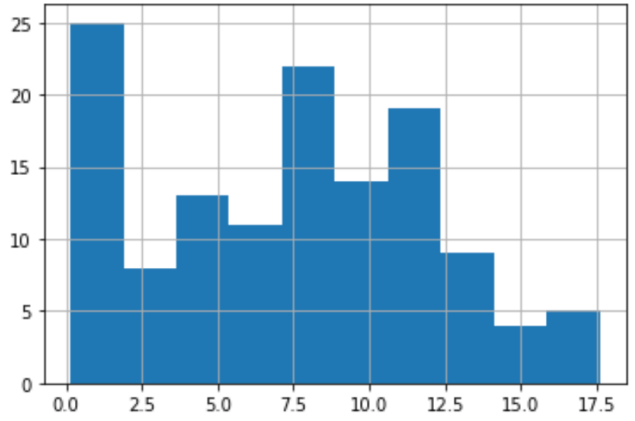
- Calculate the quartiles of the co2_emission column of food_consumption.
- Calculate the six quantiles that split up the data into 5 pieces (quintiles) of the co2_emission column of food_consumption.
- Calculate the eleven quantiles of co2_emission that split up the data into ten pieces (deciles).

Complete the following tasks

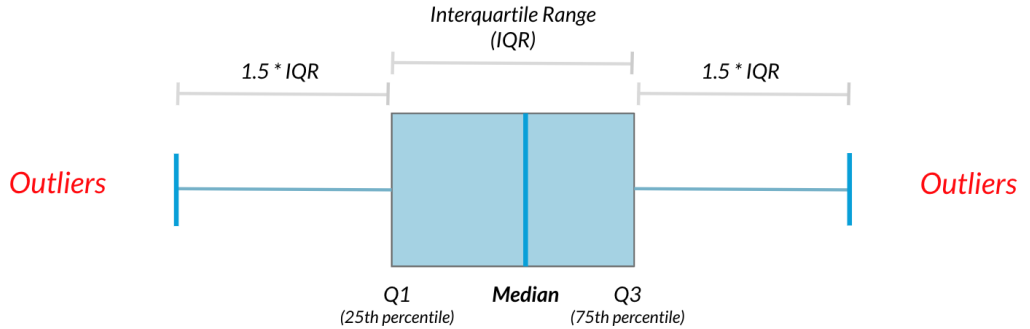
- Calculate the variance and standard deviation of co2_emission for each food_category by grouping (groupby food_category) and aggregating.
- Create a histogram of co2_emission for the beef food_category and show the plot. (subset food_category for beef, select co2_emission column and plot .hist())
- Create a histogram of co2_emission for the eggs food_category and show the plot.

food_category	var	std
beef	88748.408132	297.906710
dairy	17671.891985	132.935669
eggs	21.371819	4.622966
fish	921.637349	30.358481
lamb_goat	16475.518363	128.356996
nuts	35.639652	5.969895
pork	3094.963537	55.632396
poultry	245.026801	15.653332
rice	2281.376243	47.763754
soybeans	0.879882	0.938020
wheat	71.023937	8.427570





Finding outliers using IQR



- Outliers can have big effects on statistics like mean, as well as statistics that rely on the mean, such as variance and standard deviation. Interquartile range, or IQR, is another way of measuring spread that's less influenced by outliers. IQR is also often used to find outliers. If a value is less than Q1−1.5×IQR or greater than Q3+1.5×IQR, it's considered an outlier.
  - Subset emissions_by_country to get countries with a total emission greater than the upper cutoff or a total emission less than the lower cutoff.
  - Compute the first and third quartiles of emissions_by_country and store these as q1 and q3.
  - Calculate the interquartile range of emissions_by_country and store it as iqr.
  - Calculate the lower and upper cutoffs for outliers of emissions_by_country, and store these as lower and upper.
  - Subset emissions_by_country to get countries with a total emission greater than the upper cutoff or a total emission less than the lower cutoff.

```
# Calculate total co2_emission per country: emissions_by_country
emissions_by_country = food_consumption.groupby('country')['co2_emission'].sum()

# Compute the first and third quantiles and IQR of emissions_by_country
q1 = np.quantile(emissions_by_country, 0.25)
q3 = np.quantile(emissions_by_country, 0.75)
iqr = q3 - q1

# Calculate the lower and upper cutoffs for outliers
lower = q1 - 1.5 * iqr
upper = q3 + 1.5 * iqr

# Subset emissions_by_country to find outliers
outliers = emissions_by_country[(emissions_by_country < lower) | (emissions_by_country > upper)]
print(outliers)
```

What are chances?

People talk about chance pretty frequently, like what are the chances of closing a sale, of rain tomorrow, or of winning a game? But how exactly do we measure chance?

Measuring chance

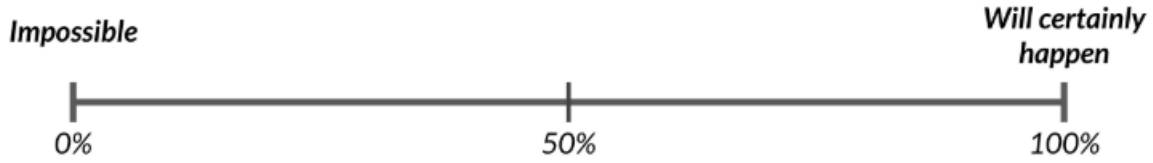
- We can measure the chances of an event using probability. We can calculate the probability of some event by taking the number of ways the event can happen and dividing it by the total number of possible outcomes.
- For example, if we flip a coin, it can land on either heads or tails. To get the probability of the coin landing on heads, we divide the 1 way to get heads by the two possible outcomes, heads and tails.

What's the probability of an event?

$$P(\text{event}) = \frac{\#\text{ ways event can happen}}{\text{total }\#\text{ of possible outcomes}}$$

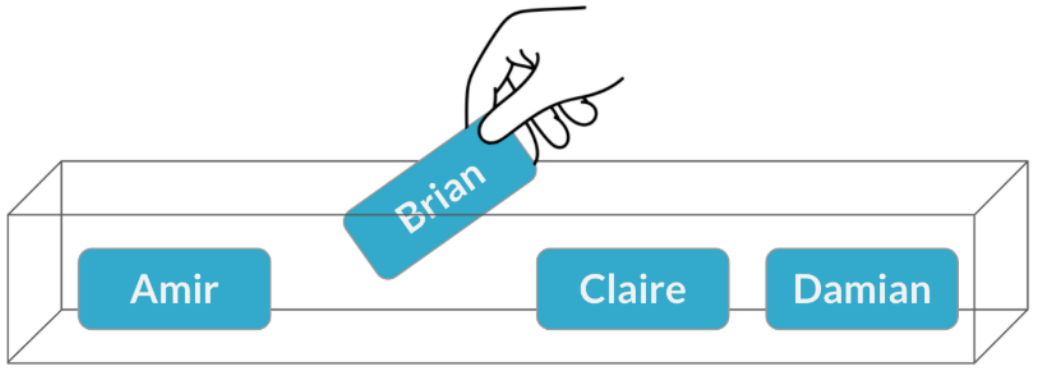
Example: a coin flip

$$P(\text{heads}) = \frac{1\text{ way to get heads}}{2\text{ possible outcomes}} = \frac{1}{2} = 50\%$$



Scenario

- Let's look at a more complex scenario. There's a meeting coming up with a potential client, and we want to send someone from the sales team to the meeting. We'll put each person's name on a ticket in a box and pull one out randomly to decide who goes to the meeting.
- Brian's name gets pulled out. The probability of Brian being selected is one out of four, or 25%.



$$P(\text{Brian}) = \frac{1}{4} = 25\%$$

Sampling from a DataFrame

- We can recreate this scenario in Python using the sample() method. By default, it randomly samples one row from the DataFrame.

```
print(sales_counts)
```

```
     name  n_sales
0    Amir      178
1   Brian      128
2  Claire       75
3  Damian       69
```

```
sales_counts.sample()
```

```
    name  n_sales
1  Brian      128
```

```
sales_counts.sample()
```

```
     name  n_sales
2  Claire       75
```

```
np.random.seed(10)
sales_counts.sample()
```

```
    name  n_sales
1  Brian      128
```

```
np.random.seed(10)
sales_counts.sample()
```

```
    name  n_sales
1  Brian      128
```

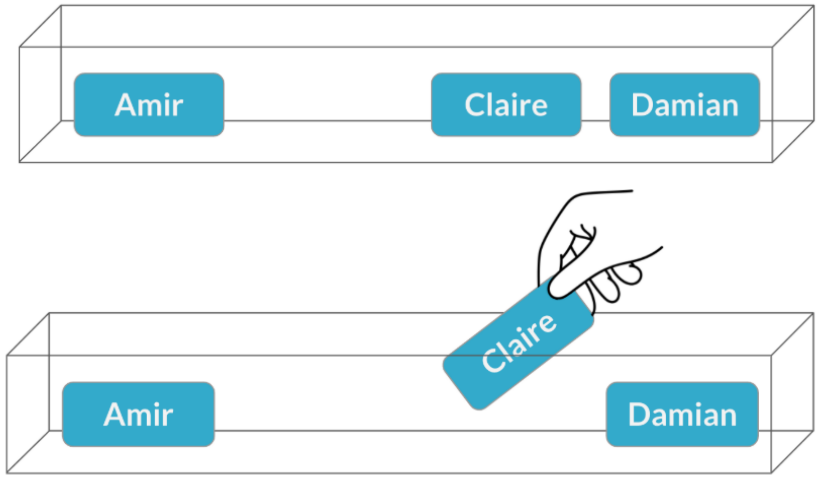
```
np.random.seed(10)
sales_counts.sample()
```

```
    name  n_sales
1  Brian      128
```

To ensure the same results are obtained when we run the script in front of the team, the random seed is to be set.

Scenario – a second meeting (sampling without replacement)

- Now there's another potential client who wants to meet at the same time, so we need to pick another salesperson. Brian haas already been picked and he can't be in two meetings at once, so we'll pick between the remaining three. This is called sampling without replacement, since we aren't replacing the name we already pulled out.
- This time, Claire is picked, and the probability of this is one out of three, or about 33%.
- To recreate this in Python, we can pass 2 into the sample method, which will give us 2 rows of the DataFrame.



$$P(\text{Claire}) = \frac{1}{3} = 33\%$$

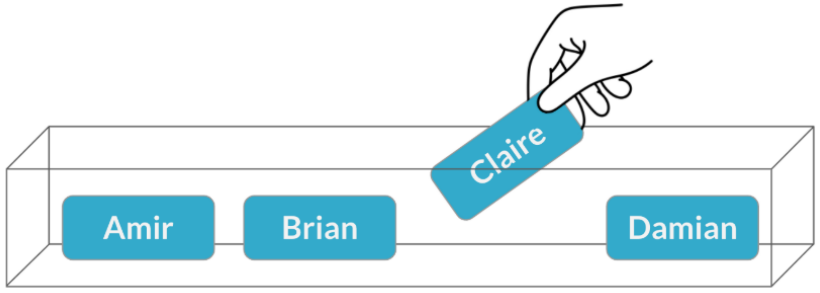
Sampling twice in Python

```
sales_counts.sample(2)
```

```
     name  n_sales
1   Brian      128
2  Claire       75
```

Sampling with replacement

- Now let's say the two meetings are happening on different days, so the same person could attend both. In this scenario, we need to return Brian's name to the box after picking it. This is called sampling with replacement.
- Claire gets picked for the second meeting, but this time, the probability of picking her is 25%.



$$P(\text{Claire}) = \frac{1}{4} = 25\%$$

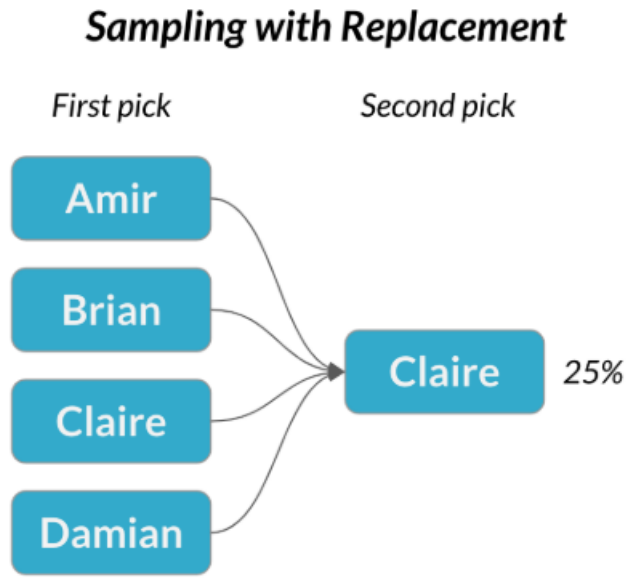
```
sales_counts.sample(5, replace = True)
```

	name	n_sales
1	Brian	128
2	Claire	75
1	Brian	128
3	Damian	69
0	Amir	178

Independent events

*Two events are **independent** if the probability of the second event **isn't** affected by the outcome of the first event.*

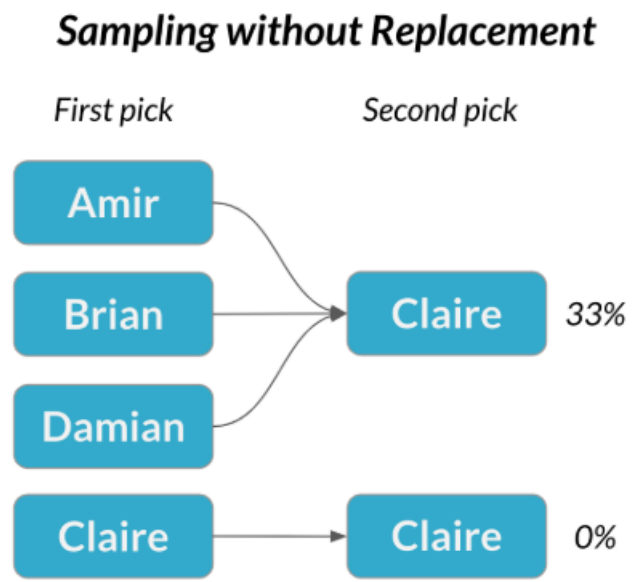
Sampling with replacement = each pick is independent



Dependent events

*Two events are **dependent** if the probability of the second event **is** affected by the outcome of the first event.*

Sampling without replacement = each pick is dependent



Calculating probabilities

$$P(\text{event}) = \frac{\#\text{ ways event can happen}}{\text{total }\#\text{ of possible outcomes}}$$

- You're in charge of the sales team, and it's time for performance reviews, starting with Amir. As part of the review, you want to randomly select a few of the deals that he's worked on over the past year so that you can look at them more deeply. Before you start selecting deals, you'll first figure out what the chances are of selecting certain deals.

- Complete the following tasks.
  - Load amir_deals.csv to amir_deals
  - Count the number of deals Amir worked on for each product type and store in counts.
  - Calculate the probability of selecting a deal for the different product types by dividing the counts by the total number of deals Amir worked on. Save this as probs.
  - If you randomly select one of Amir's deals, what's the probability that the deal will involve Product C? (15%, 80.43%, 8.43%, 22.5% or 124.3%)

```
amir_deals = pd.read_csv('amir_deals.csv')
# Count the deals for each product
counts = amir_deals['product'].value_counts()

# Calculate probability of picking a deal with each product
probs = counts / amir_deals.shape[0]
print(probs)
```

```
Product B    0.348315
Product D    0.224719
Product A    0.129213
Product C    0.084270
Product F    0.061798
Product H    0.044944
Product I    0.039326
Product E    0.028090
Product N    0.016854
Product G    0.011236
Product J    0.011236
```

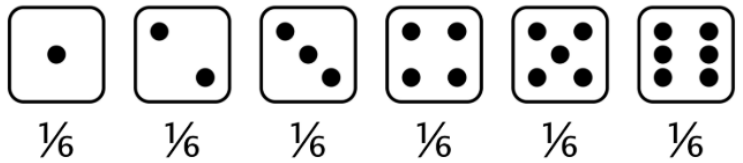
Sampling deals

- In the previous exercise, you counted the deals Amir worked on. Now it's time to randomly pick five deals so that you can reach out to each customer and ask if they were satisfied with the service they received. You'll try doing this both with and without replacement.
- Additionally, you want to make sure this is done randomly and that it can be reproduced in case you get asked how you chose the deals, so you'll need to set the random seed before sampling from the deals.
- Complete the following tasks.
  - Set the random seed to 24.
  - Take a sample of 5 deals without replacement and store them as sample_without_replacement.
  - Take a sample of 5 deals with replacement and save as sample_with_replacement.

```
# Set random seed
np.random.seed(24)

# Sample 5 deals without replacement
sample_without_replacement = amir_deals.sample(5)
print(sample_without_replacement)

# Sample 5 deals with replacement
sample_with_replacement = amir_deals.sample(5, replace=True)
print(sample_with_replacement)
```

Discrete distributions

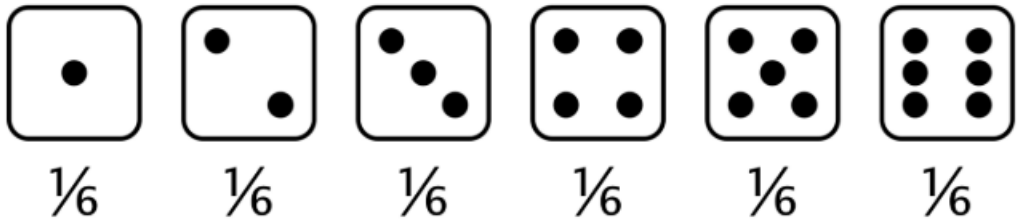
- We will take a deeper dive into probability and begin looking at probability distributions.
- Let's consider rolling a standard, six-sided die.
- There are six numbers, or six possible outcomes, and every number has one sixth, or about a 17 percent chance of being rolled. This is an example of a probability distribution.



Probability Distribution

- A probability distribution describes the probability of each possible outcome in a scenario. We can also talk about the expected value of a distribution, which is the mean of a distribution. We can calculate this by multiplying each value by its probability (one sixth in this case) and summing, so the expected value of rolling a fair die is 3.5.

Describes the probability of each possible outcome in a scenario



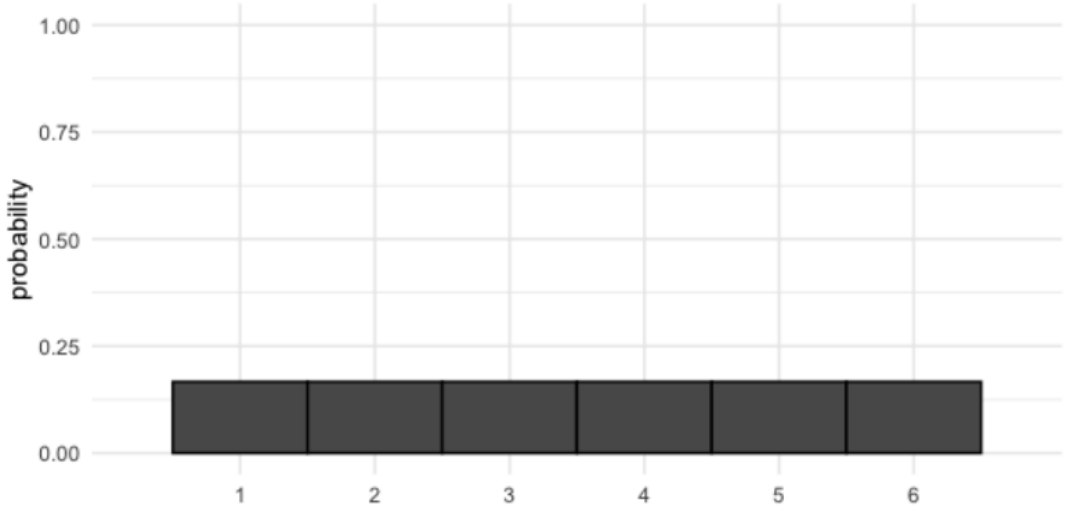
⅙ ⅙ ⅙ ⅙ ⅙ ⅙

Expected value: mean of a probability distribution

Expected value of a fair die roll =

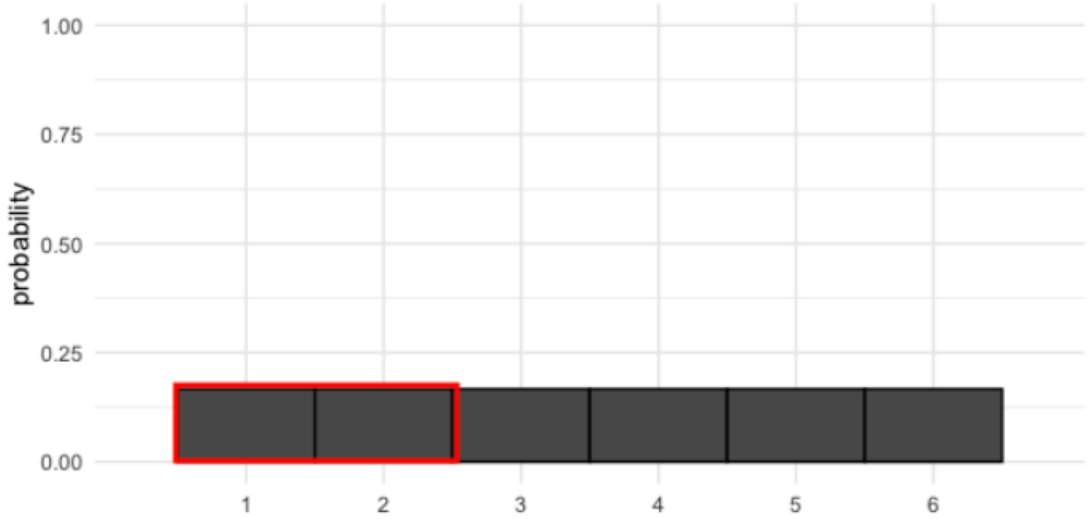
$$(1 \times \tfrac{1}{6}) + (2 \times \tfrac{1}{6}) + (3 \times \tfrac{1}{6}) + (4 \times \tfrac{1}{6}) + (5 \times \tfrac{1}{6}) + (6 \times \tfrac{1}{6}) = 3.5$$

Visualizing a probability distribution



Probability = area

$$P(\text{die roll}) \leq 2 = ?$$

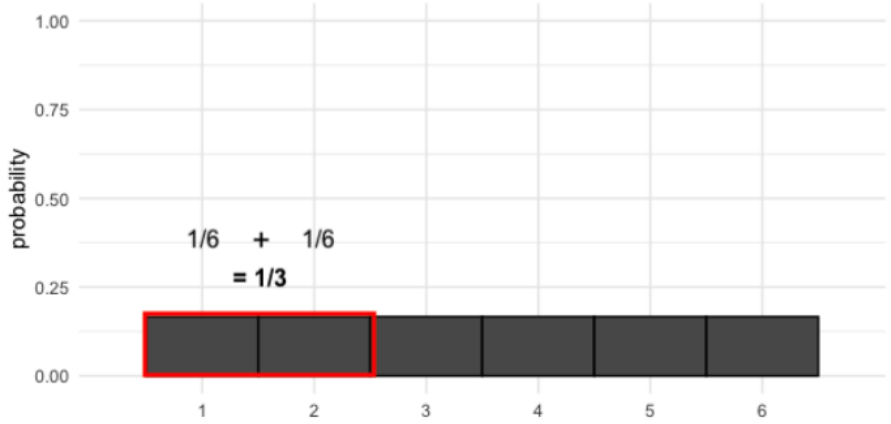


We can calculate probabilities of different outcomes by taking areas of the probability distribution. For example, what's the probability that our die roll is less than or equal to 2?

Each bar has a width of 1 and a height of one sixth, so the area of each bar is one sixth. We'll sum the areas for 1 and 2, to get a total probability of one third.

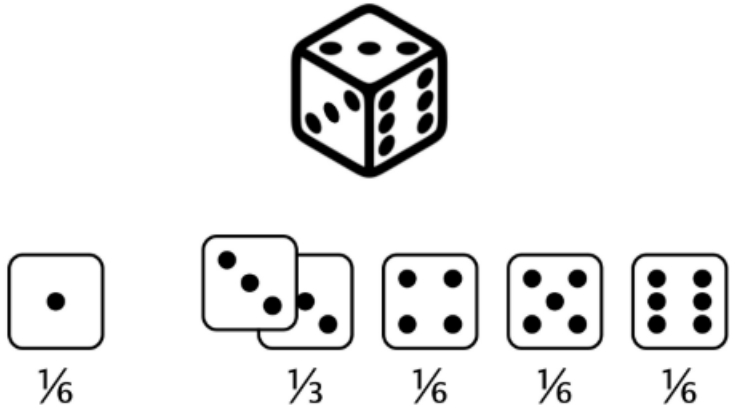
Probability = area

$$P(\text{die roll}) \leq 2 = 1/3$$



Uneven die

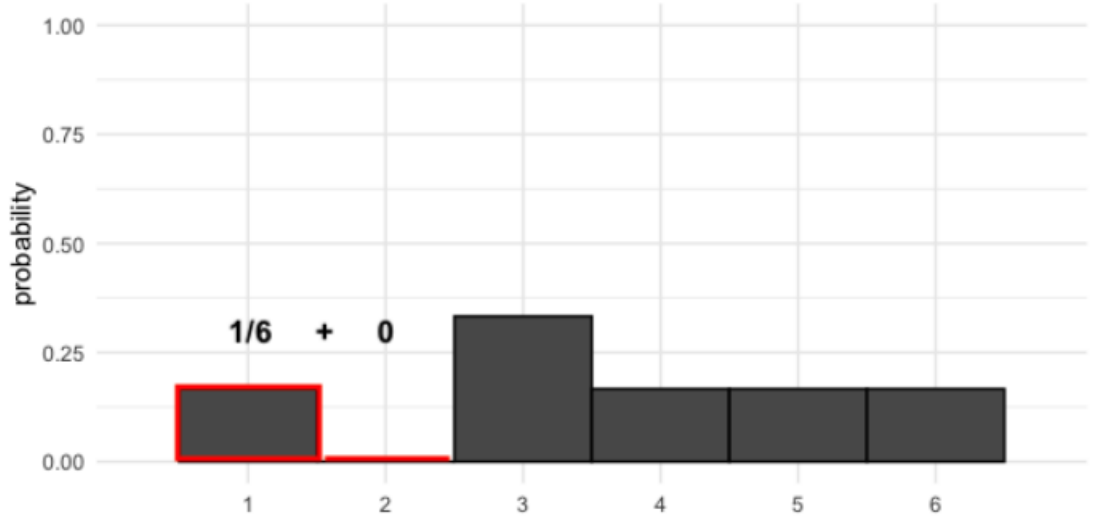
- Now let's say we have a die where the two got turned into a three. This means that we now have a 0% chance of getting a 2, and a 33% chance of getting a 3. To calculate the expected value of this die, we now multiply 2 by 0, since it's impossible to get a 2, and 3 by its new probability, one third. This gives us an expected value that's slightly higher than the fair die.



Expected value of uneven die roll =

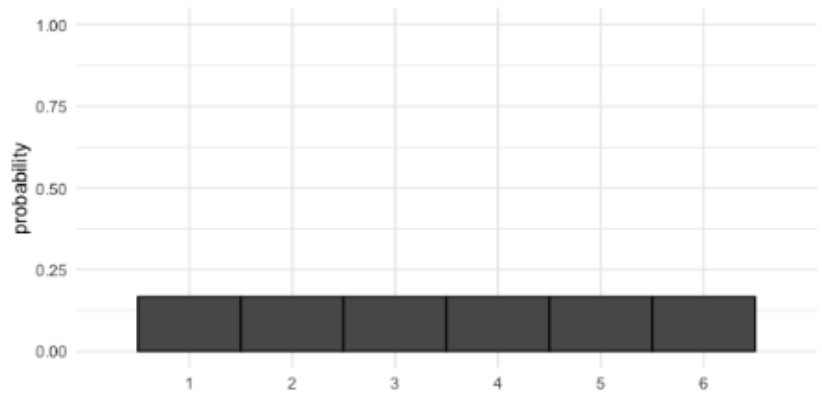
$$(1 \times \tfrac{1}{6}) + (2 \times 0) + (3 \times \tfrac{1}{3}) + (4 \times \tfrac{1}{6}) + (5 \times \tfrac{1}{6}) + (6 \times \tfrac{1}{6}) = 3.67$$

$$P(\text{uneven die roll}) \leq 2 = 1/6$$

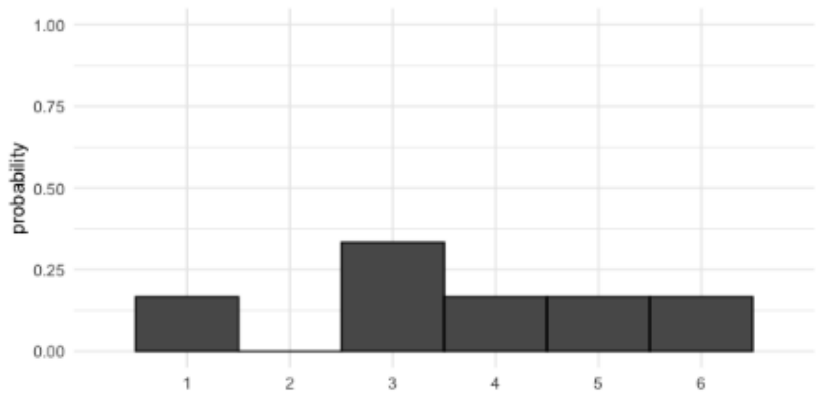


Describe probabilities for discrete outcomes

Fair die



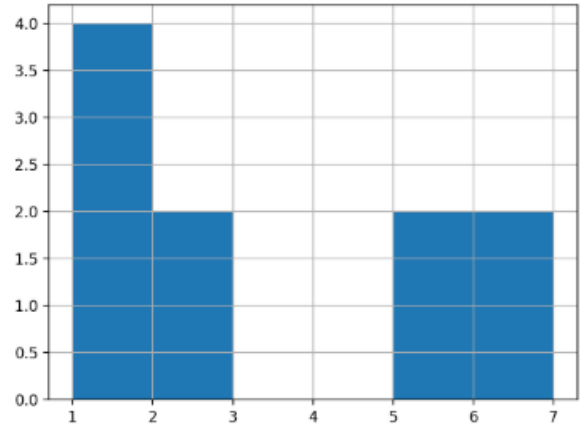
Uneven die



Discrete uniform distribution

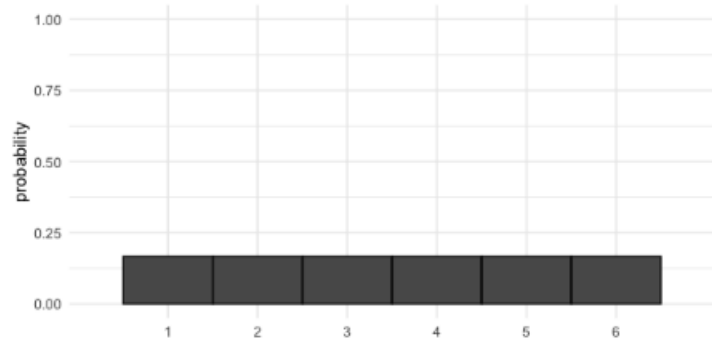
Sample distribution vs. theoretical distribution

Sample of 10 rolls



`np.mean(rolls_10['number'])` = `3.0`

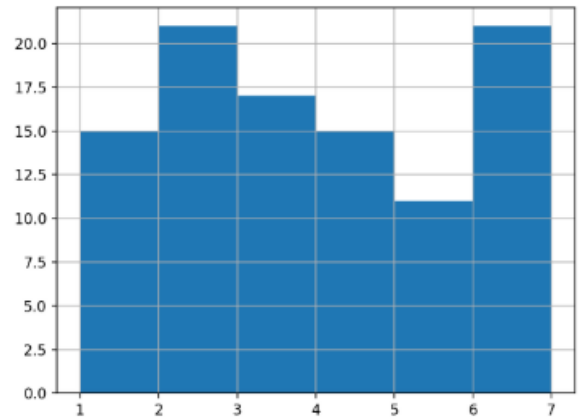
Theoretical probability distribution



`mean(die['number'])` = `3.5`

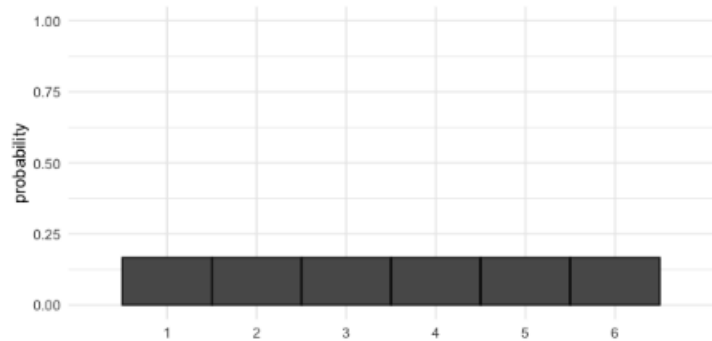
A bigger sample

Sample of 100 rolls



`np.mean(rolls_100['number'])` = `3.4`

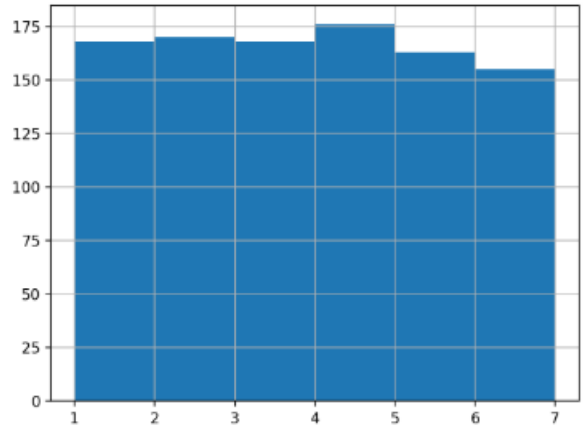
Theoretical probability distribution



`mean(die['number'])` = `3.5`

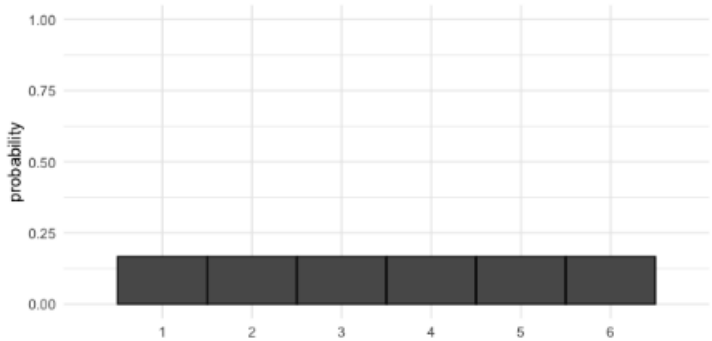
An even bigger sample

Sample of 1000 rolls



`np.mean(rolls_1000['number'])` = `3.48`

Theoretical probability distribution



`mean(die['number'])` = `3.5`

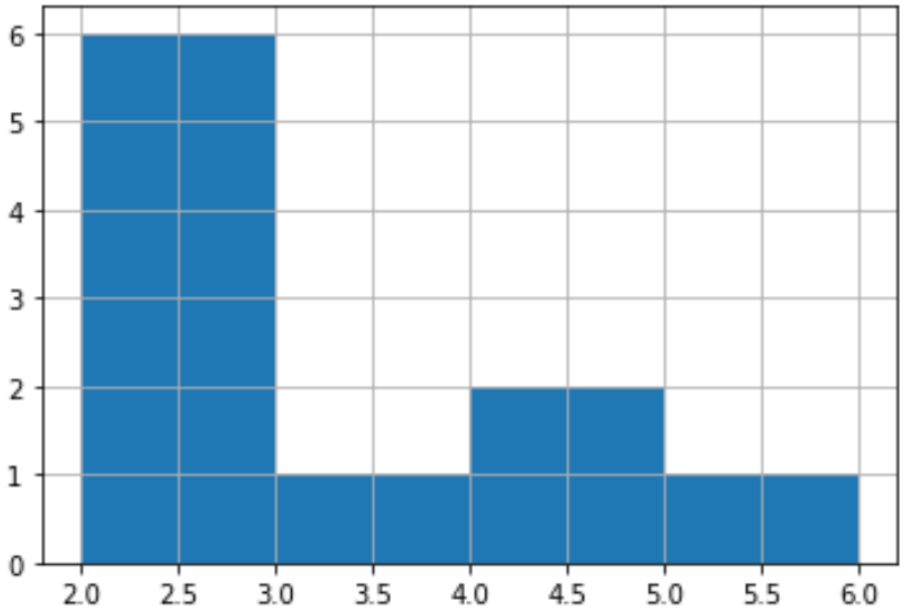
Law of large numbers

As the size of your sample increases, the sample mean will approach the expected value.

Sample size	Mean
10	3.00
100	3.40
1000	3.48

Creating a probability distribution

- A new restaurant opened a few months ago, and the restaurant's management wants to optimize its seating space based on the size of the groups that come most often. On one night, there are 10 groups of people waiting to be seated at the restaurant, but instead of being called in the order they arrived, they will be called randomly. In this exercise, you'll investigate the probability of groups of different sizes getting picked first. Data on each of the ten groups is contained in the restaurant_groups (from restaurant_groups.csv) DataFrame.
- **Complete the following tasks**
- Create a histogram of the group_size column of restaurant_groups, setting bins to [2, 3, 4, 5, 6]. Remember to show the plot.
- Count the number of each group_size in restaurant_groups, then divide by the number of rows in restaurant_groups to calculate the probability of randomly selecting a group of each size. Save as size_dist.
- Reset the index of size_dist.
- Rename the columns of size_dist to group_size and prob.
- Calculate the expected value of the size_distribution, which represents the expected group size, by multiplying the group_size by the prob and taking the sum.
- Calculate the probability of randomly picking a group of 4 or more people by subsetting for groups of size 4 or more and summing the probabilities of selecting those groups.

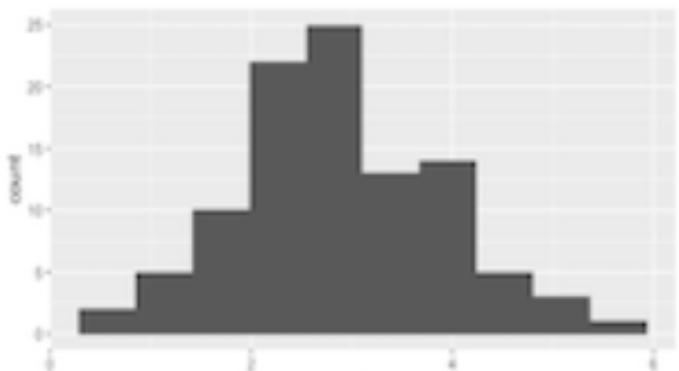



```
   group_size  prob
0           2   0.6
1           4   0.2
2           6   0.1
3           3   0.1
The expected value is 2.9000000000000004
The probability is  0.30000000000000004
```

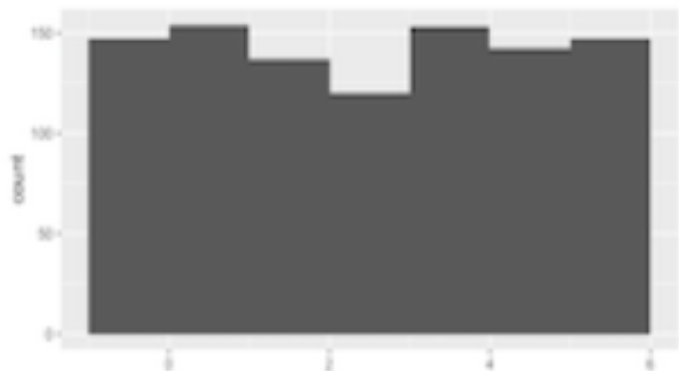
Identifying distributions

Which sample is most likely to have been taken from a uniform distribution?

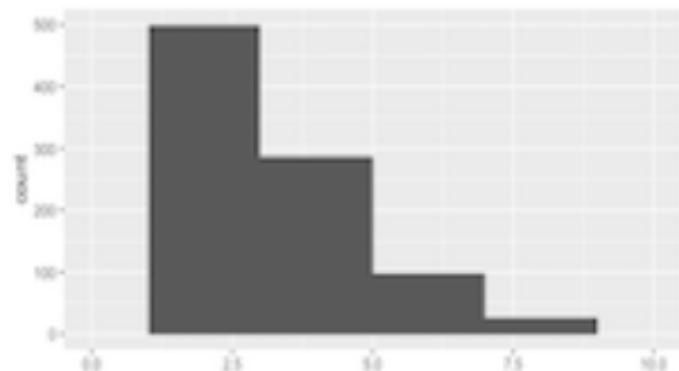
A



B

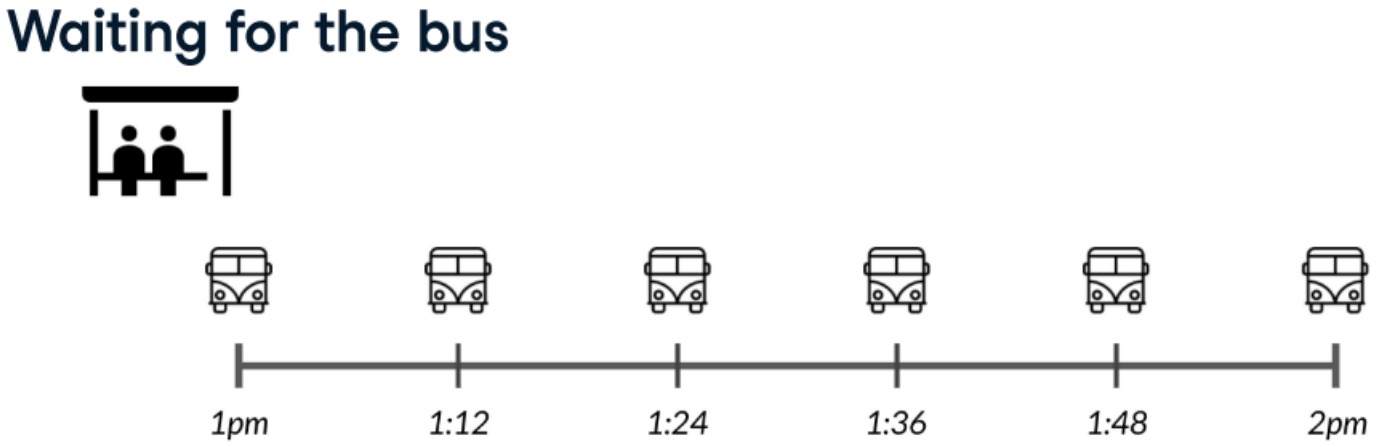


C



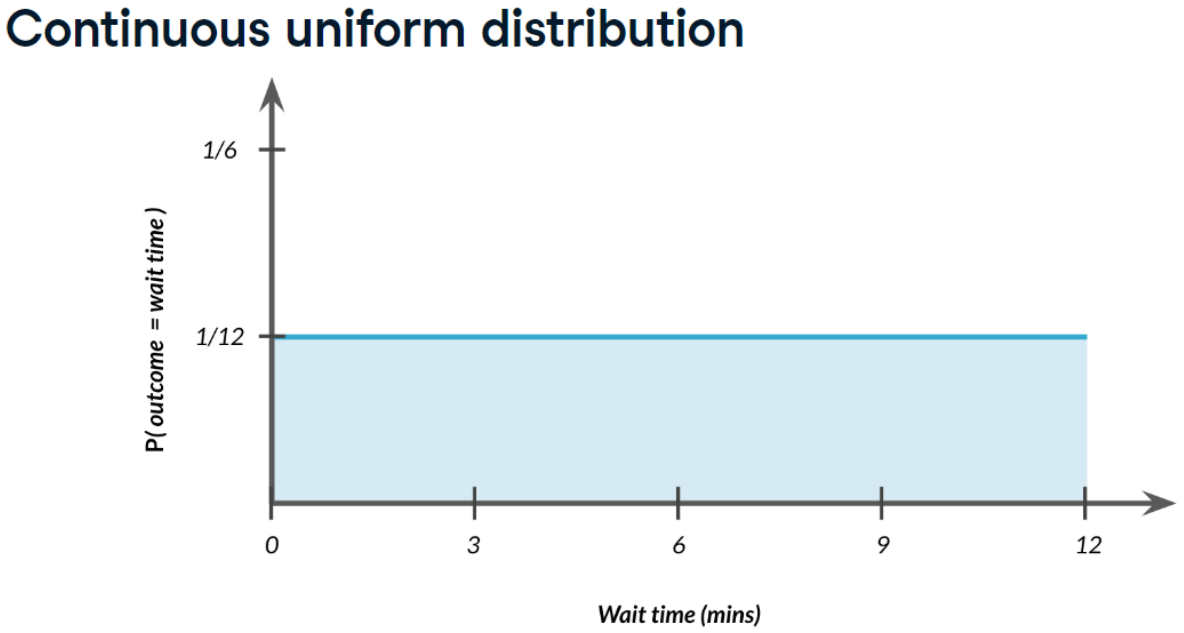
Continuous distributions

- We can use discrete distributions to model situations that involve discrete or countable variables, but how can we model continuous variables?
- Let's start with an example. The city bus arrives once every twelve minutes, so if you show up at a random time, you could wait anywhere from 0 minutes if you just arrive as the bus pulls in, up to 12 minutes if you arrive just as the bus leaves.



Let’s model a scenario

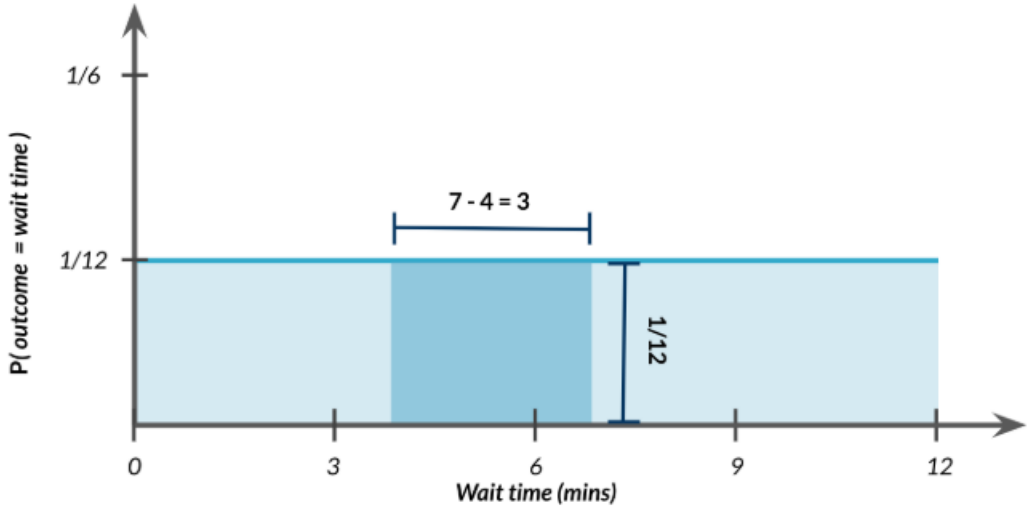
- Let's model this scenario with a probability distribution. There are an infinite number of minutes we could wait since we could wait 1 minute, 1.5 minutes, 1.53 minutes, and so on, so we can't create individual blocks like we could with a discrete variable.
- Instead, we'll use a continuous line to represent probability. The line is flat since there's the same probability of waiting any time from 0 to 12 minutes. This is called the continuous uniform distribution.



Probability still = area

- Now that we have our distribution, let's figure out what the probability is that we'll wait between 4 and 7 minutes. Just like with discrete distributions, we can take the area from 4 to 7 to calculate probability.
- The width of this rectangle is 7 minus 4 which is 3. The height is one twelfth.
- Multiplying those together to get area, we get 3/12 or 25%.

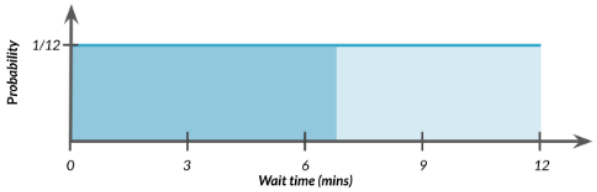
$$P(4 \leq \text{wait time} \leq 7) = 3 \times 1/12 = 3/12$$



Uniform distribution in Python

- Let's use the uniform distribution in Python to calculate the probability of waiting 7 minutes or less. We need to import uniform from scipy.stats. We can call uniform.cdf and pass it 7, followed by the lower and upper limits, which in our case is 0 and 12. The probability of waiting less than 7 minutes is about 58%.
- If we want the probability of waiting more than 7 minutes, we need to take 1 minus the probability of waiting less than 7 minutes.

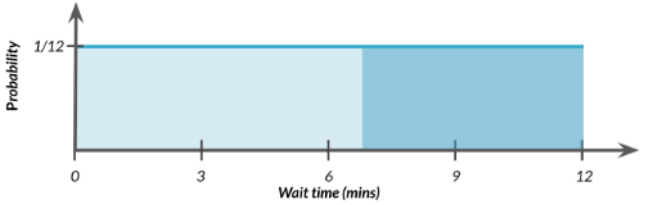
$P(\text{wait time} \leq 7)$




```
from scipy.stats import uniform
uniform.cdf(7, 0, 12)
```

```
0.5833333
```

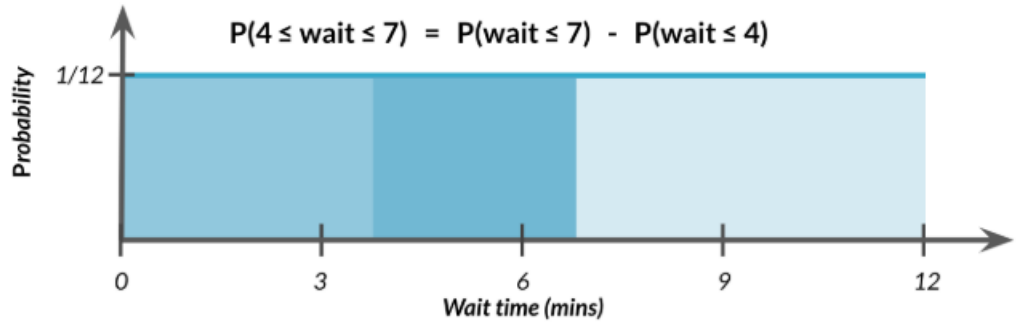
$P(\text{wait time} \geq 7) = 1 - P(\text{wait time} \leq 7)$




```
from scipy.stats import uniform
1 - uniform.cdf(7, 0, 12)
```

```
0.4166667
```

$$P(4 \leq \text{wait time} \leq 7)$$

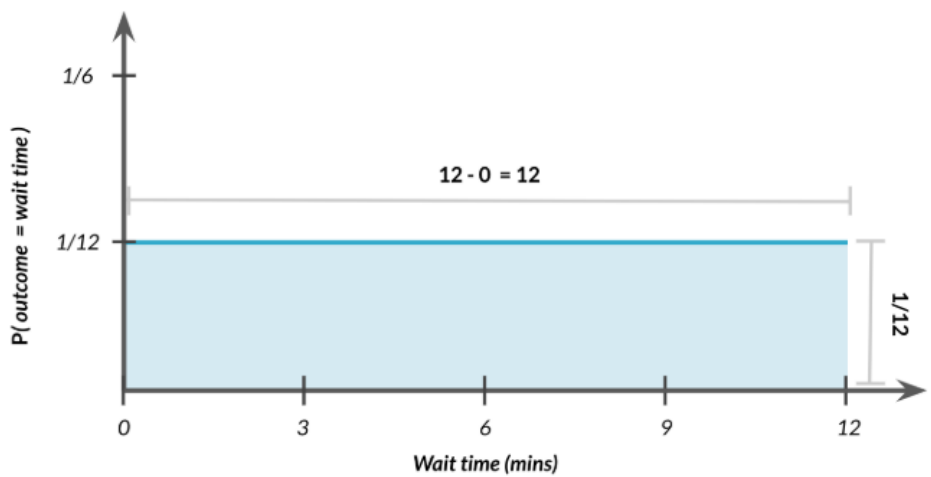



```
from scipy.stats import uniform
uniform.cdf(7, 0, 12) - uniform.cdf(4, 0, 12)
```

```
0.25
```

Total area = 1

$$P(0 \leq \text{outcome} \leq 12) = 12 \times 1/12 = 1$$

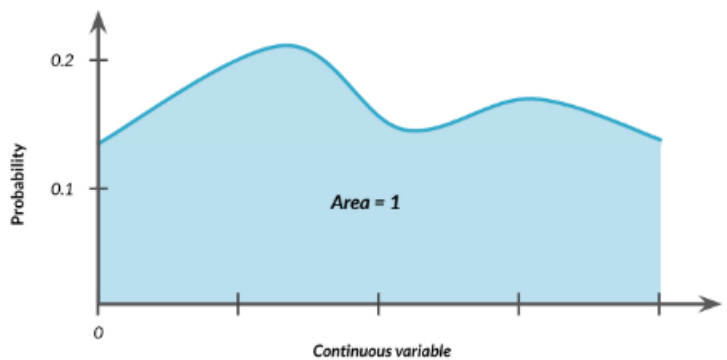


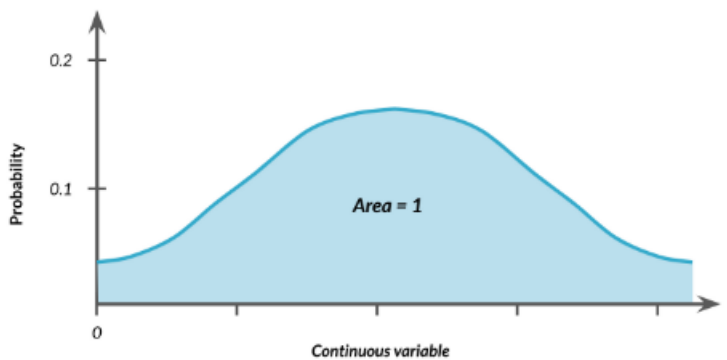
Generating random numbers according to uniform distribution

```
from scipy.stats import uniform
uniform.rvs(0, 5, size=10)
```

```
array([1.89740094, 4.70673196, 0.33224683, 1.0137103 , 2.31641255,
       3.49969897, 0.29688598, 0.92057234, 4.71086658, 1.56815855])
```

Other continuous distributions





Practice – Data back-ups

- The sales software used at your company is set to automatically back itself up, but no one knows exactly what time the back-ups happen. It is known, however, that back-ups happen exactly every 30 minutes. Amir comes back from sales meetings at random times to update the data on the client he just met with. He wants to know how long he'll have to wait for his newly-entered data to get backed up. Use your new knowledge of continuous uniform distributions to model this situation and answer Amir's questions.
- To model how long Amir will wait for a back-up using a continuous uniform distribution, save his lowest possible wait time as min_time and his longest possible wait time as max_time. Remember that back-ups happen every 30 minutes.
- Import uniform from scipy.stats and calculate the probability that Amir has to wait less than 5 minutes, and store in a variable called prob_less_than_5.
- Calculate the probability that Amir has to wait more than 5 minutes, and store in a variable called prob_greater_than_5.
- Calculate the probability that Amir has to wait between 10 and 20 minutes, and store in a variable called prob_between_10_and_20.

Practice – Data back-ups

- To model how long Amir will wait for a back-up using a continuous uniform distribution, save his lowest possible wait time as min_time and his longest possible wait time as max_time. Remember that back-ups happen every 30 minutes.
- Import uniform from scipy.stats and calculate the probability that Amir has to wait less than 5 minutes, and store in a variable called prob_less_than_5.
- Calculate the probability that Amir has to wait more than 5 minutes, and store in a variable called prob_greater_than_5.
- Calculate the probability that Amir has to wait between 10 and 20 minutes, and store in a variable called prob_between_10_and_20.

```
min_time = 0
max_time = 30

# Import uniform from scipy.stats
from scipy.stats import uniform

# Calculate probability of waiting more than 5 mins
prob_greater_than_5 = 1-uniform.cdf(5,0,30)
print(prob_greater_than_5)

# Calculate probability of waiting 10-20 mins
prob_between_10_and_20 = uniform.cdf(20,0,30)-uniform.cdf(10,0,30)
print(prob_between_10_and_20)
```

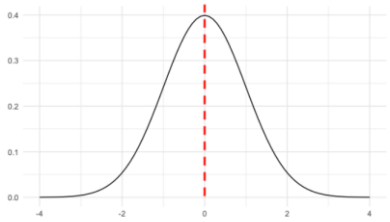
Simulating wait times

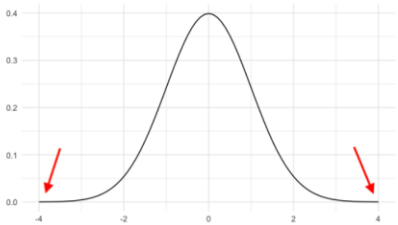
- To give Amir a better idea of how long he'll have to wait, you'll simulate Amir waiting 1000 times and create a histogram to show him what he should expect. Recall from the last exercise that his minimum wait time is 0 minutes and his maximum wait time is 30 minutes.
- Set the random seed to 334.
- Generate 1000 wait times from the continuous uniform distribution that models Amir's wait time. Save this as wait_times.
- Create a histogram of the simulated wait times and show the plot.

```
# Set random seed to 334
np.random.seed(334)

# Import uniform
from scipy.stats import uniform

# Generate 1000 wait times between 0 and 30 mins
wait_times = uniform.rvs(0, 30, size=1000)

# Create a histogram of simulated times and show plot
plt.hist(wait_times)
plt.show()
```

Normal distribution

- It's one of the most important probability distributions you'll learn about since a countless number of statistical methods rely on it, and it applies to more real-world situations than the distributions we've covered so far.
- The normal distribution looks like a "bell curve".
  - First, it's symmetrical, so the left side is a mirror image of the right.
  - Second, just like any continuous distribution, the area beneath the curve is 1.
  - The probability never hits 0, even if it looks like it does at the tail ends. Only 0-point-006% of its area is contained beyond the edges of this graph.



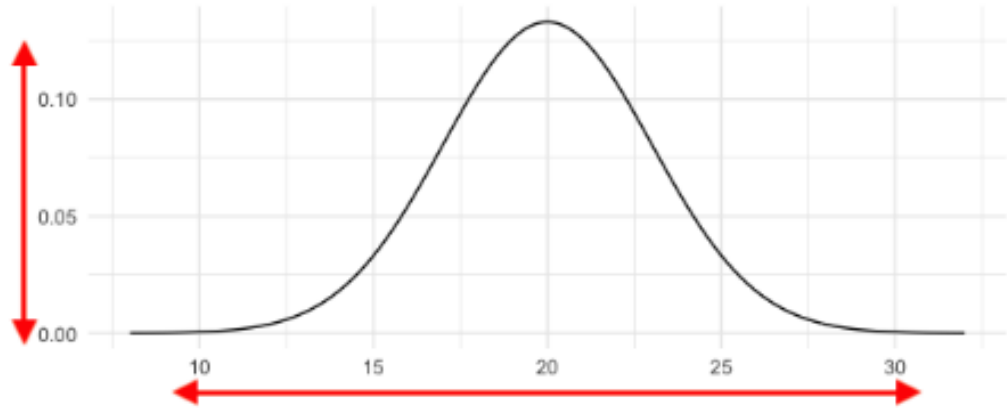


Described by mean and standard deviation

Mean: 20

Standard deviation:

3

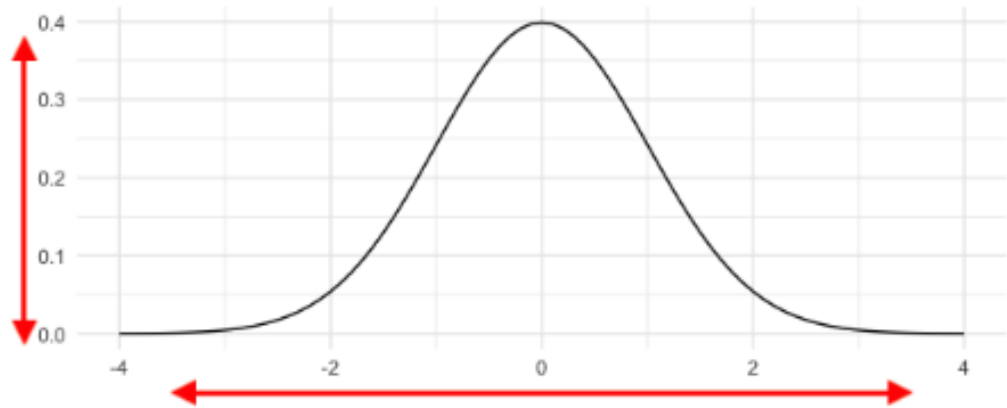


Standard normal distribution

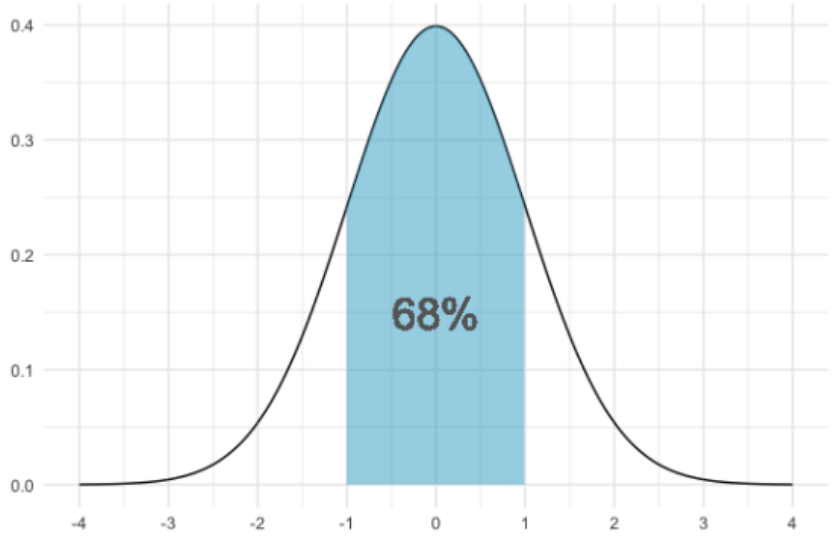
Mean: 0

Standard deviation:

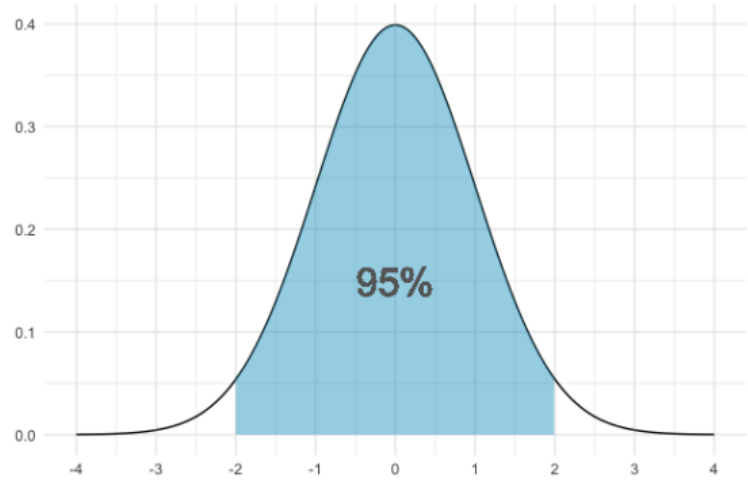
1



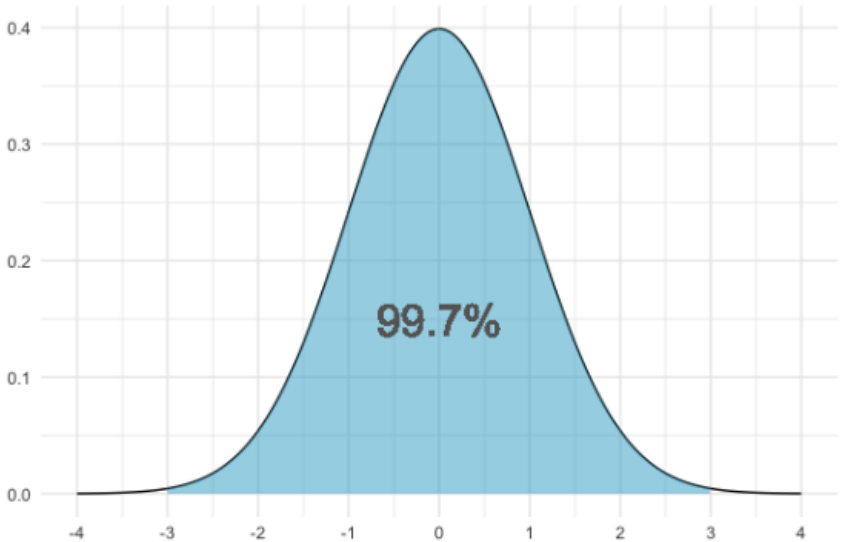
68% falls within 1 standard deviation



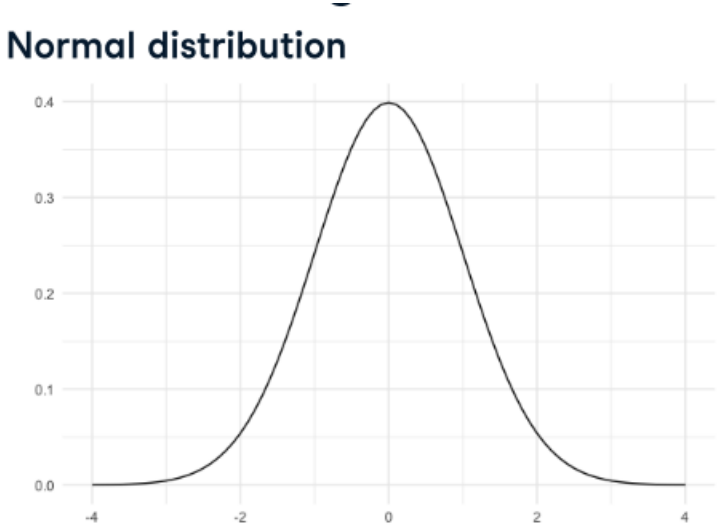
95% falls within 2 standard deviations

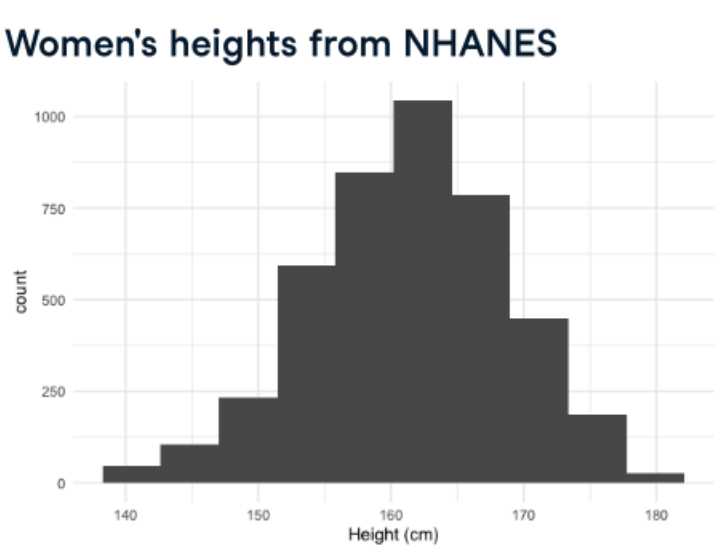


99.7% falls within 3 standard deviations



This is sometimes called the 68-95-99-point-7 rule.

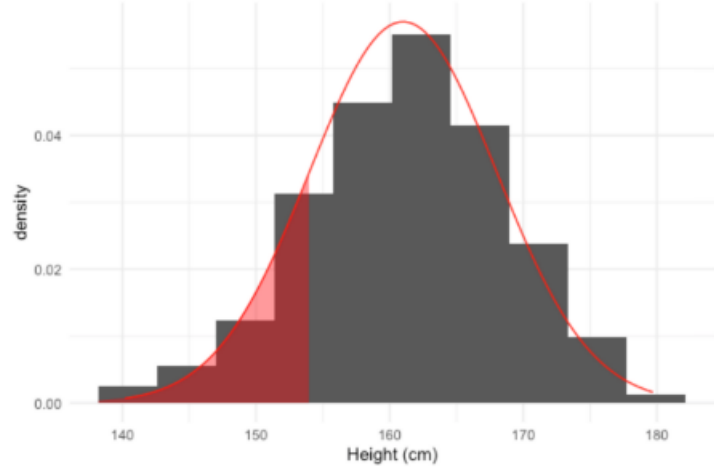




Mean: 161 cm *Standard deviation: 7 cm*

There's lots of real-world data shaped like the normal distribution. For example, here is a histogram of the heights of women that participated in the National Health and Nutrition Examination Survey. The mean height is around 161 centimeters, and the standard deviation is about 7 centimeters.

What percent of women are shorter than 154 cm?

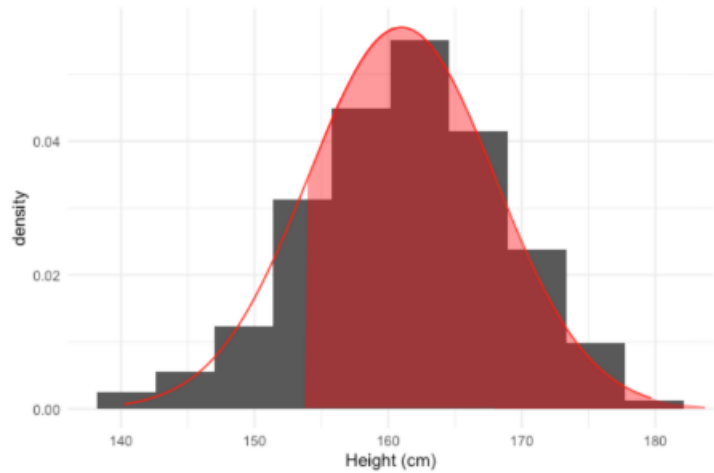



```
from scipy.stats import norm
norm.cdf(154, 161, 7)
```

```
0.158655
```

16% of women in the survey are shorter than 154 cm

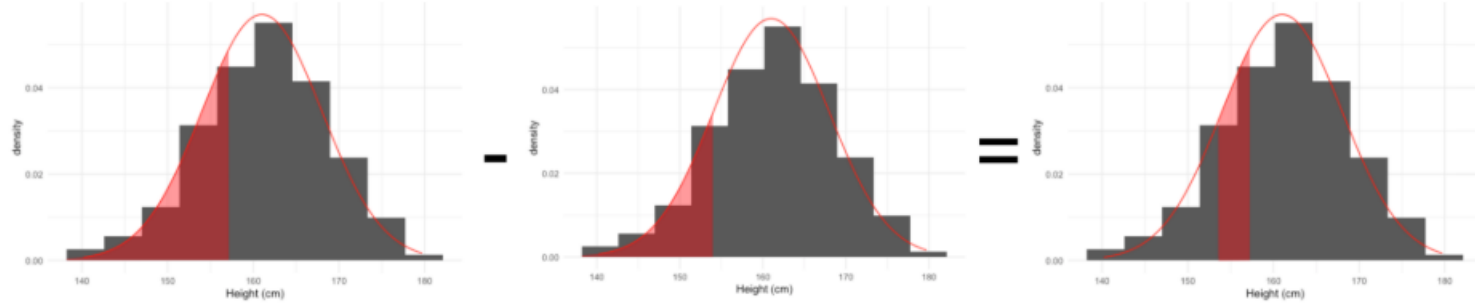
What percent of women are taller than 154 cm?




```
from scipy.stats import norm
1 - norm.cdf(154, 161, 7)
```

```
0.841345
```

What percent of women are 154-157 cm?




```
norm.cdf(157, 161, 7) - norm.cdf(154, 161, 7)
```

```
0.1252
```

Generating random numbers

```
# Generate 10 random heights
norm.rvs(161, 7, size=10)
```

```
array([155.5758223 , 155.13133235, 160.06377097, 168.33345778,
       165.92273375, 163.32677057, 165.13280753, 146.36133538,
       149.07845021, 160.5790856 ])
```

The central limit theorem

Rolling the dice 5 times

```
die = pd.Series([1, 2, 3, 4, 5, 6])
# Roll 5 times
samp_5 = die.sample(5, replace=True)
print(samp_5)
```

```
array([3, 1, 4, 1, 1])
```

```
np.mean(samp_5)
```

```
2.0
```

```
# Roll 5 times and take mean
samp_5 = die.sample(5, replace=True)
np.mean(samp_5)
```

```
4.4
```

```
samp_5 = die.sample(5, replace=True)
np.mean(samp_5)
```

```
3.8
```

Rolling the dice 5 times 10 times

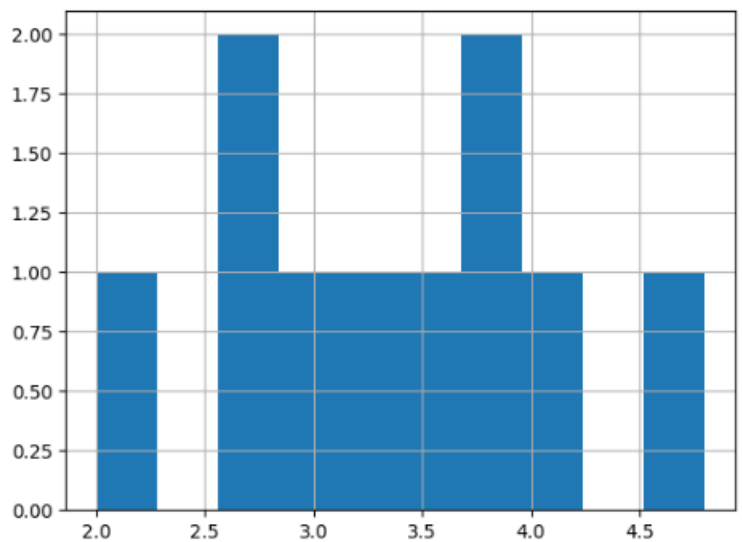
Repeat 10 times:

- Roll 5 times
- Take the mean

```
sample_means = []
for i in range(10):
  samp_5 = die.sample(5, replace=True)
  sample_means.append(np.mean(samp_5))
print(sample_means)
```

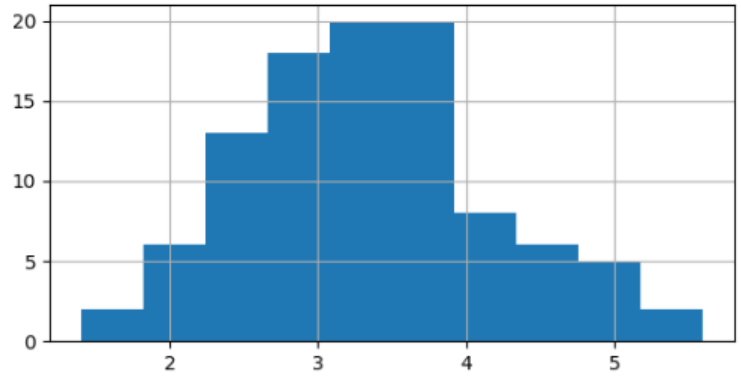
```
[3.8, 4.0, 3.8, 3.6, 3.2, 4.8, 2.6,
3.0, 2.6, 2.0]
```

Sampling distribution of the sample mean



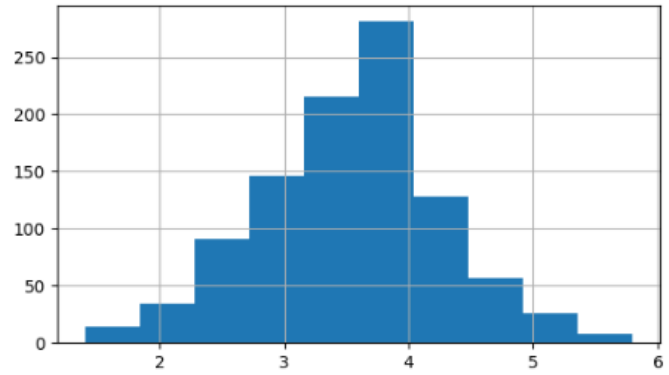
100 sample means

```
sample_means = []
for i in range(100):
    sample_means.append(np.mean(die.sample(5, replace=True)))
```



1000 sample means

```
sample_means = []
for i in range(1000):
    sample_means.append(np.mean(die.sample(5, replace=True)))
```



This phenomenon is known as the central limit theorem, which states that a sampling distribution will approach a normal distribution as the number of trials increases. In our example, the sampling distribution became closer to the normal distribution as we took more and more sample means. It's important to note that the central limit theorem only applies when samples are taken randomly and are independent, for example, randomly picking sales deals with replacement.

Central Limit Theorem in action

- The central limit theorem states that a sampling distribution of a sample statistic approaches the normal distribution as you take more samples, no matter the original distribution being sampled from.

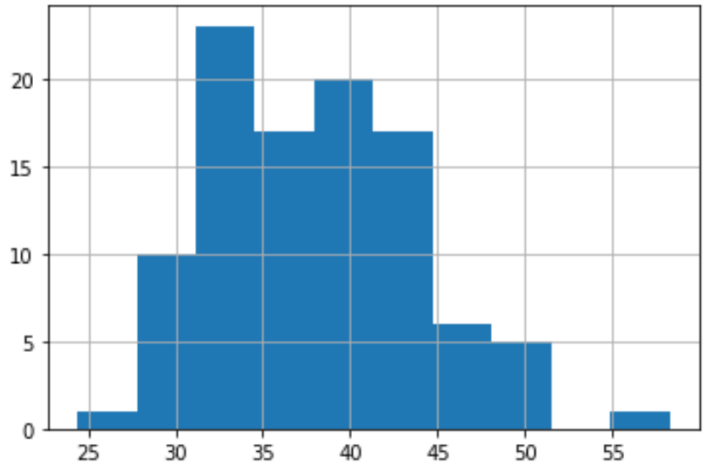
- In this exercise, you'll focus on the sample mean and see the central limit theorem in action while examining the num_users column of amir_deals more closely, which contains the number of people who intend to use the product Amir is selling.

- Create a histogram of the num_users column of amir_deals and show the plot.

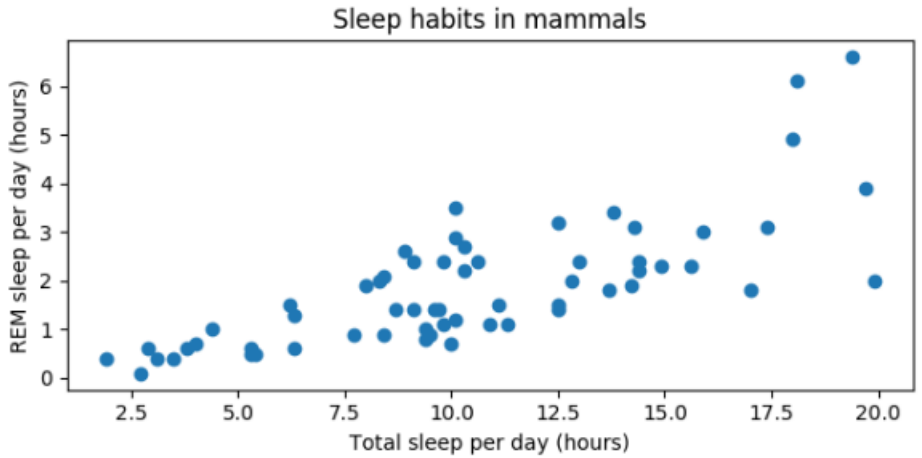
- Set the seed to 104.
- Take a sample of size 20 with replacement from the num_users column of amir_deals, and take the mean.

- Repeat this 100 times using a for loop and store as sample_means. This will take 100 different samples and calculate the mean of each.

- Convert sample_means into a pd.Series, create a histogram of the sample_means, and show the plot.



Correlation



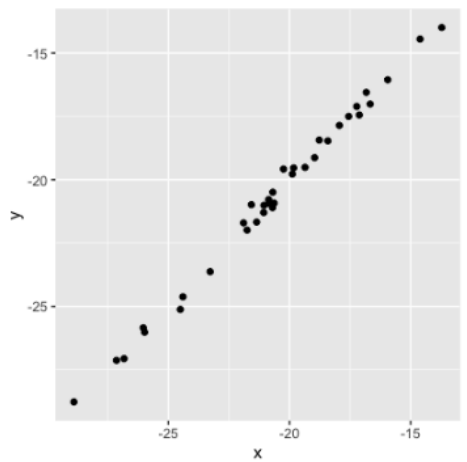
Correlation coefficient

- Quantifies the linear relationship between two variables
- Number between -1 and 1
- Magnitude corresponds to strength of relationship
- Sign (+ or -) corresponds to direction of relationship

- x = explanatory/independent variable
- y = response/dependent variable

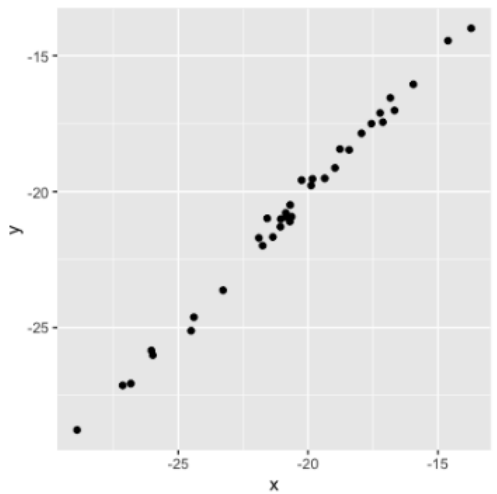
Magnitude = strength of relationship

0.99 (very strong relationship)

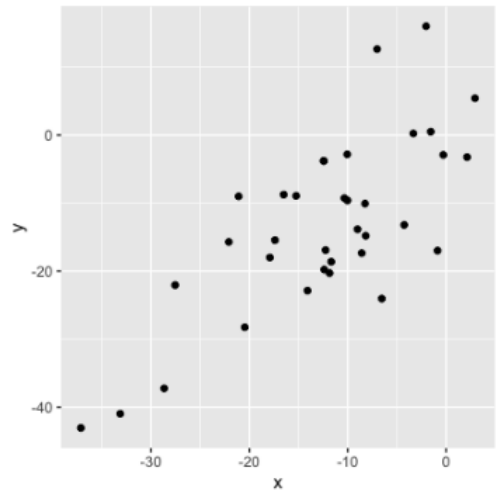


Magnitude = strength of relationship

0.99 (very strong relationship)

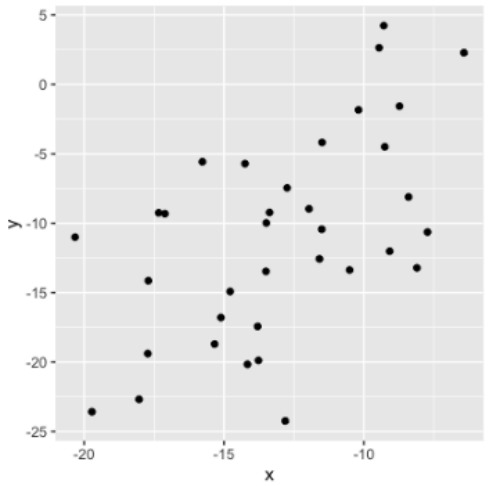


0.75 (strong relationship)



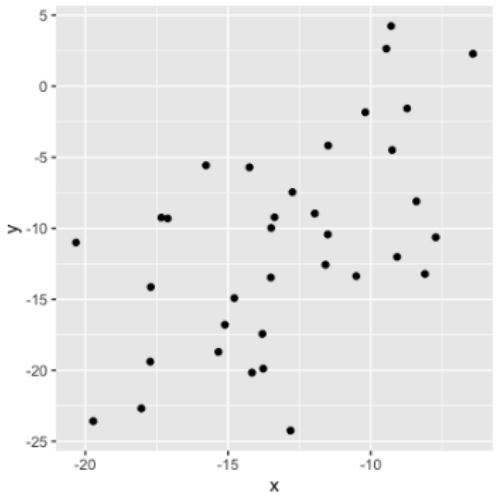
Magnitude = strength of relationship

0.56 (moderate relationship)

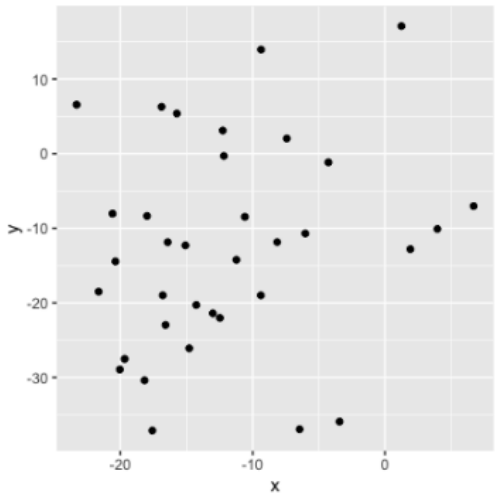


Magnitude = strength of relationship

0.56 (moderate relationship)

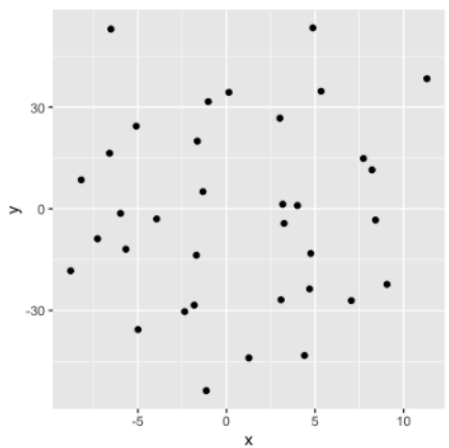


0.21 (weak relationship)



Magnitude = strength of relationship

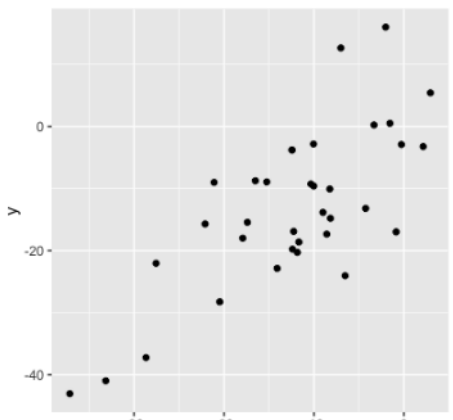
0.04 (no relationship)



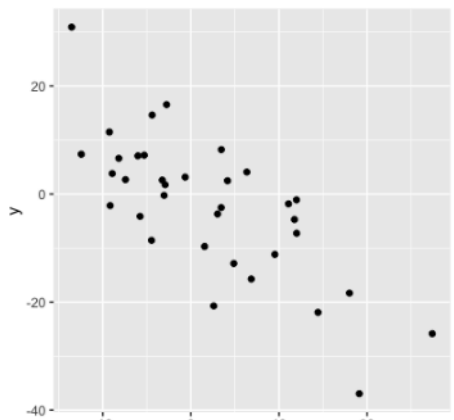
- Knowing the value of `x` doesn't tell us anything about `y`

Sign = direction

0.75: as `x` increases, `y` increases

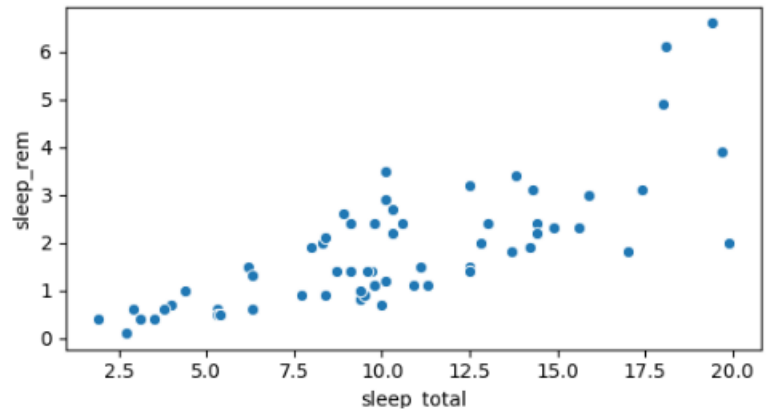


-0.75: as `x` increases, `y` decreases



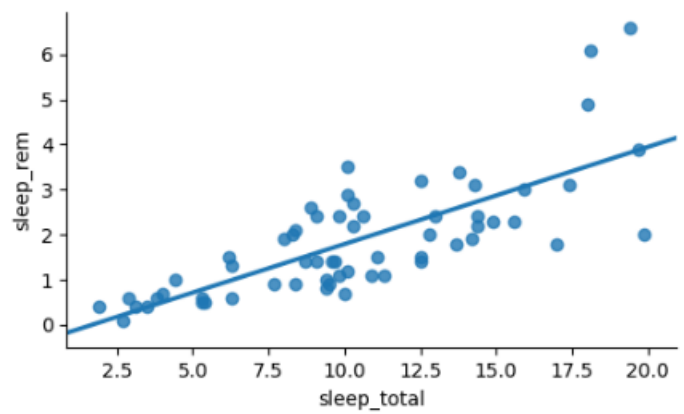
Visualizing relationships

```
import seaborn as sns
sns.scatterplot(x="sleep_total", y="sleep_rem", data=msleep)
plt.show()
```



Adding a trendline

```
import seaborn as sns
sns.lmplot(x="sleep_total", y="sleep_rem", data=msleep, ci=None)
plt.show()
```



Computing correlation

```
msleep['sleep_total'].corr(msleep['sleep_rem'])
```

```
0.751755
```

```
msleep['sleep_rem'].corr(msleep['sleep_total'])
```

```
0.751755
```

- Relationships between variables
- You'll be working with a dataset world_happiness containing results from the 2019 World Happiness Report. The report scores various countries based on how happy people in that country are. It also ranks each country on various societal aspects such as social support, freedom, corruption, and others. The dataset also includes the GDP per capita and life expectancy for each country.

- In this exercise, you'll examine the relationship between a country's life expectancy (life_exp) and happiness score (happiness_score) both visually and quantitatively.
- world_happiness is loaded from world_happiness.csv.

- Create a scatterplot of happiness_score vs. life_exp (without a trendline) using seaborn.
- Show the plot.

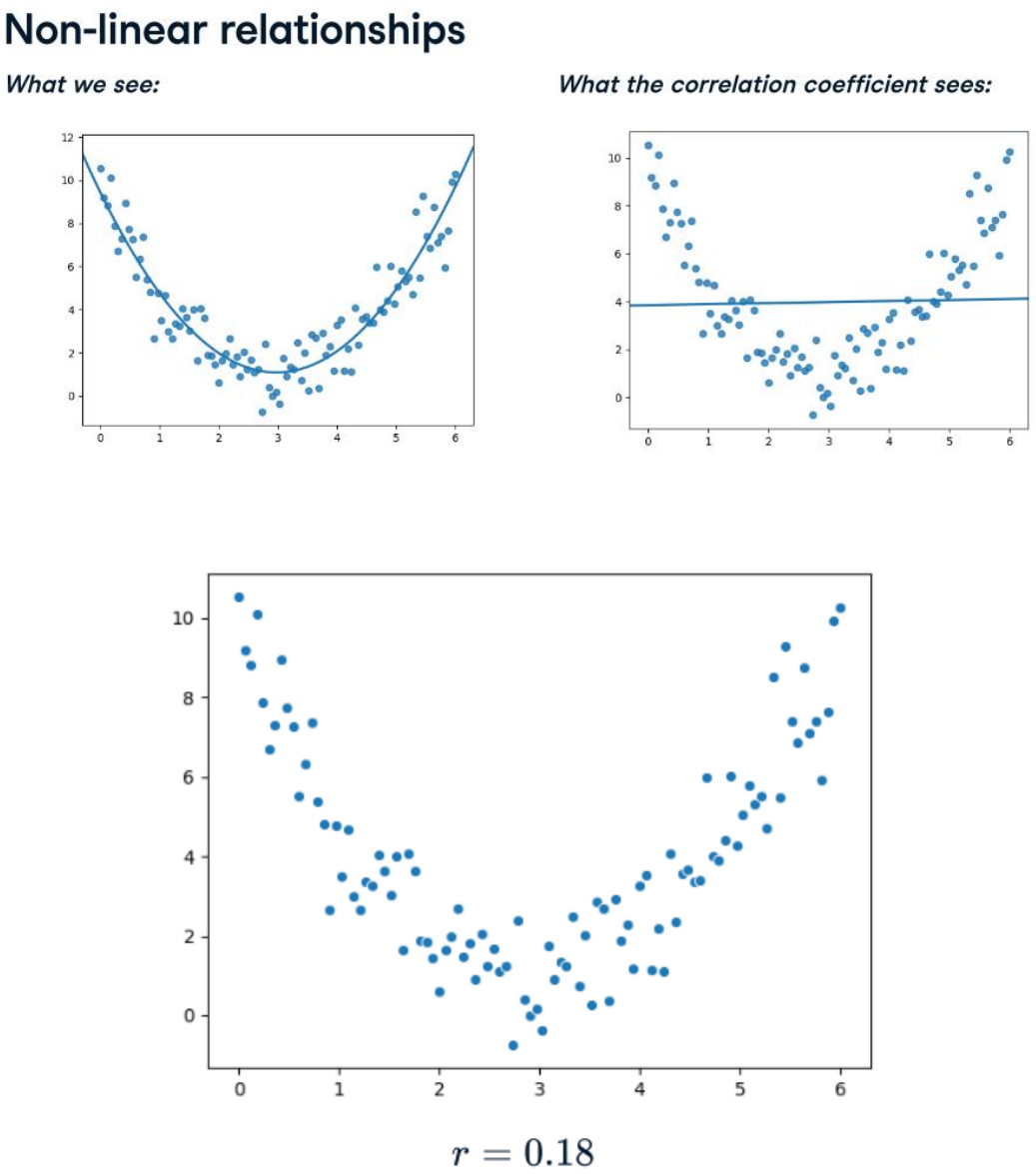
- Create a scatterplot of happiness_score vs. life_exp with a linear trendline using seaborn, setting ci to None.
- Show the plot.

- Based on the scatterplot shown, which is most likely be the correlation between the two values?
  - 0.3, -0.3, 0.8, -0.8

- Calculate the correlation between life_exp and happiness_score. Save this as cor.

Correlation caveats

- Consider the graph. There is clearly a relationship between x and y, but when we calculate the correlation, we get 0.18.
- This is because the relationship between the two variables is a quadratic relationship, not a linear relationship. The correlation coefficient measures the strength of linear relationships, and linear relationships only.

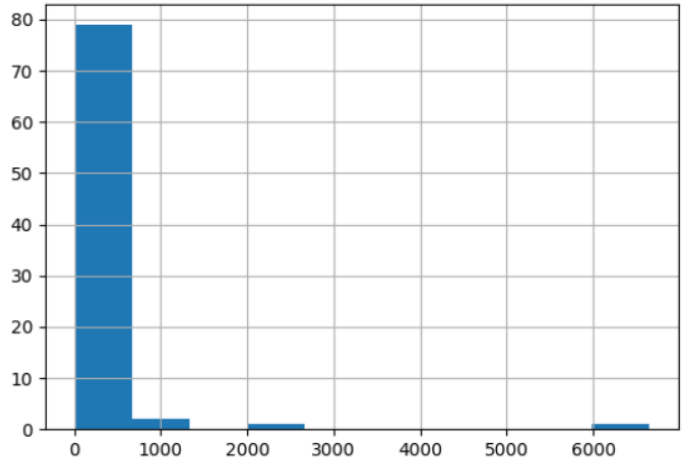


Mammal sleep data

```
print(msleep)
```

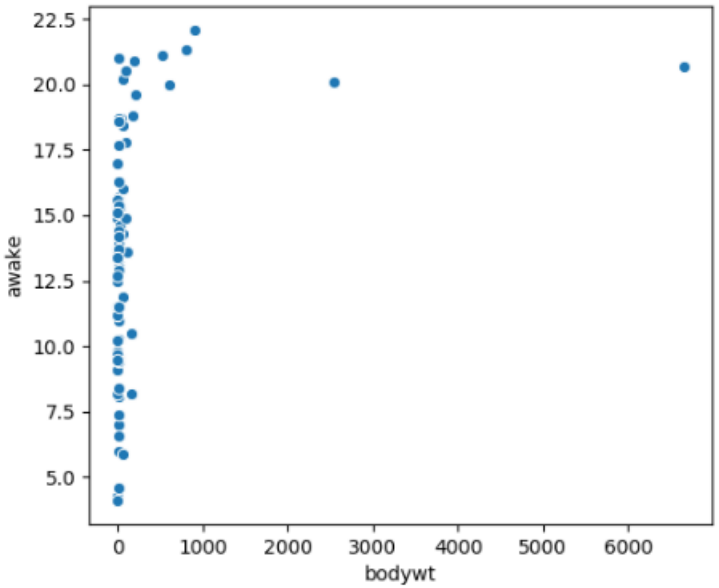
	name	genus	vore	order	...	sleep_cycle	awake	brainwt	bodywt
1	Cheetah	Acinonyx	carni	Carnivora	...	NaN	11.9	NaN	50.000
2	Owl monkey	Aotus	omni	Primates	...	NaN	7.0	0.01550	0.480
3	Mountain beaver	Aplodontia	herbi	Rodentia	...	NaN	9.6	NaN	1.350
4	Greater short-ta...	Blarina	omni	Soricomorpha	...	0.133333	9.1	0.00029	0.019
5	Cow	Bos	herbi	Artiodactyla	...	0.666667	20.0	0.42300	600.000
..	...	...	...	...	...	...	...	...	...
79	Tree shrew	Tupaia	omni	Scandentia	...	0.233333	15.1	0.00250	0.104
80	Bottle-nosed do...	Tursiops	carni	Cetacea	...	NaN	18.8	NaN	173.330
81	Genet	Genetta	carni	Carnivora	...	NaN	17.7	0.01750	2.000
82	Arctic fox	Vulpes	carni	Carnivora	...	NaN	11.5	0.04450	3.380
83	Red fox	Vulpes	carni	Carnivora	...	0.350000	14.2	0.05040	4.230

Distribution of body weight



If we take a closer look at the distribution for bodywt, it's highly skewed. There are lots of lower weights and a few weights that are much higher than the rest.

Body weight vs. awake time




```
msleep['bodywt'].corr(msleep['awake'])
```

```
0.3119801
```

Log transformation

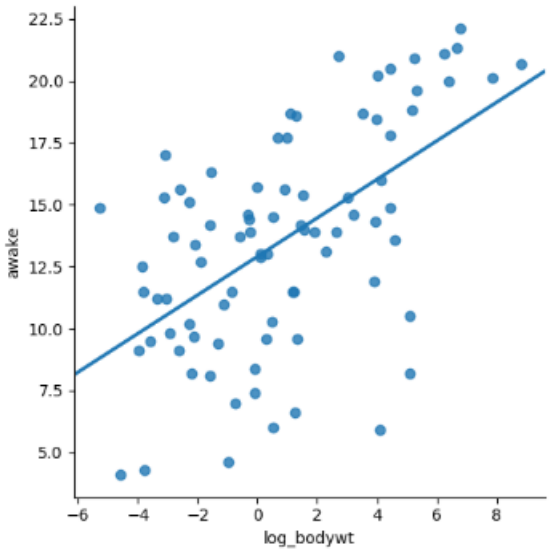
- When data is highly skewed like this, we can apply a log transformation.
- We'll create a new column called log_bodywt which holds the log of each body weight. We can do this using np.log. If we plot the log of bodyweight versus awake time, the relationship looks much more linear than the one between regular bodyweight and awake time. The correlation between the log of bodyweight and awake time is about 0.57, which is much higher than the 0.3 we had before.

Log transformation

```
msleep['log_bodywt'] = np.log(msleep['bodywt'])

sns.lmplot(x='log_bodywt',
           y='awake',
           data=msleep,
           ci=None)
plt.show()
```

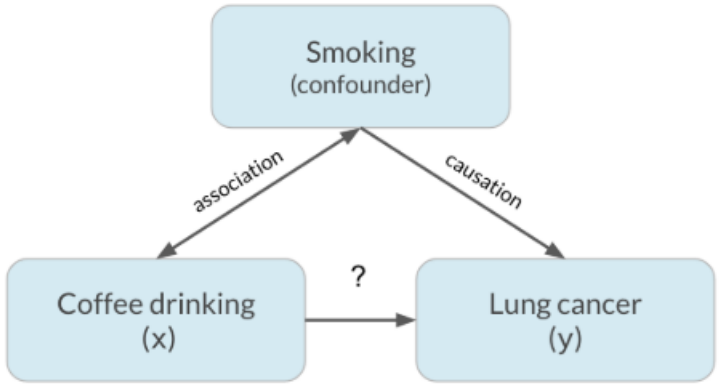
```
msleep['log_bodywt'].corr(msleep['awake'])
```

```
0.5687943
```



A phenomenon called confounding can lead to spurious correlations. Let's say we want to know if drinking coffee causes lung cancer. Looking at the data, we find that coffee drinking and lung cancer are correlated, which may lead us to think that drinking more coffee will give you lung cancer.

Confounding



However, there is a third, hidden variable at play, which is smoking. It is also known that smoking causes lung cancer.

In reality, it turns out that coffee does not cause lung cancer and is only associated with it, but it appeared causal due to the third variable, smoking. This third variable is called a confounder, or lurking variable. This means that the relationship of interest between coffee and lung cancer is a spurious correlation.

Practice

- Transforming variables
- When variables have skewed distributions, they often require a transformation in order to form a linear relationship with another variable so that correlation can be computed. In this exercise, you'll perform a transformation yourself.
- Work with world_happiness DataFrame
- Complete the following tasks.
  - Create a scatterplot of happiness_score versus gdp_per_cap and calculate the correlation between them.
  - Add a new column to world_happiness called log_gdp_per_cap that contains the log of gdp_per_cap.
  - Create a seaborn scatterplot of happiness_score versus log_gdp_per_cap.
  - Calculate the correlation between log_gdp_per_cap and happiness_score.

Thank you for your attention..