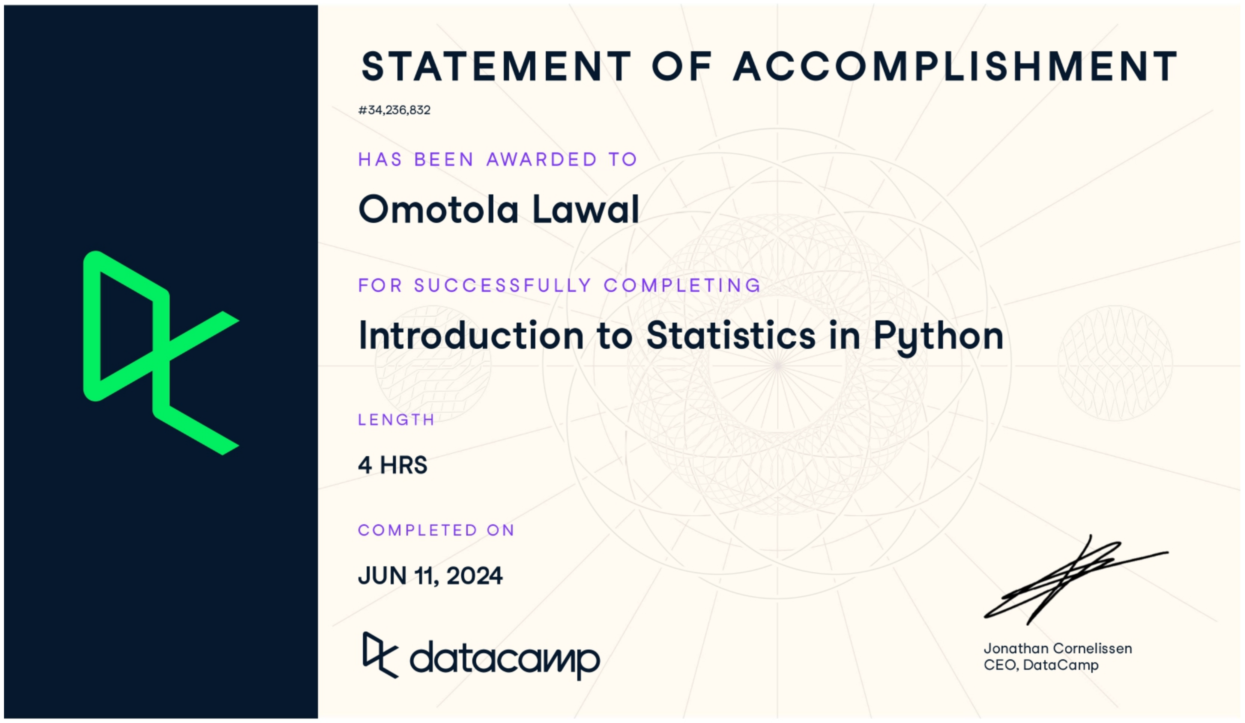
Introduction to Statistics in Python

Lawal's Note

2025-09-26

Table of contents



STATEMENT OF ACCOMPLISHMENT

#34,236,832

HAS BEEN AWARDED TO

Omotola Lawal

FOR SUCCESSFULLY COMPLETING

Introduction to Statistics in Python

LENGTH

4 HRS

COMPLETED ON

JUN 11, 2024

datacamp

[illegible]

Jonathan Cornelissen
CEO, DataCamp

Chapter 1

Chapter 1.1: Data type classification

In the course , you will learn about two main types of data: numeric and categorical data.

Numeric variables can be classified as either discrete or continuous, and categorical variables can be classified as either nominal or ordinal. These characteristics of a variable determine which ways of summarizing your data will work best.

Measure of Center

In this lesson, we'll begin to discuss summary statistics, some of which you may already be familiar with, like mean and median.

Mammal sleep data

In this lesson, we'll look at data about different mammals' sleep habits.

Histograms

Before we dive in, let's remind ourselves how histograms work. A histogram takes a bunch of data points and separates them into bins, or ranges of values. Here, there's a bin for 0 to 2 hours, 2 to 4 hours, and so on. The heights of the bars represent the number of data points that fall into that bin, so there's one mammal in the dataset that sleeps between 0 to 2 hours, and nine mammals that sleep two to four hours. Histograms are a great way to visually summarize the data, but we can use numerical summary statistics to summarize even further.

How long do mammals in this dataset typically sleep?

One way we could summarize the data is by answering the question, How long do mammals in this dataset typically sleep? To answer this, we need to figure out what the "typical" or "center" value of the data is. We'll discuss three different definitions, or measures, of center: mean, median, and mode.

Measures of center: mean

The mean, often called the average, is one of the most common ways of summarizing data. To calculate mean, we add up all the numbers of interest and divide by the total number of data points, which is 83 here. This gives us 10-point-43 hours of sleep. In Python, we can use numpy's mean function, passing it the variable of interest.

Measures of center: median

Another measure of center is the median. The median is the value where 50% of the data is lower than it, and 50% of the data is higher. We can calculate this by sorting all the data points and taking the middle one, which would be index 41 in this case. This gives us a median of 10-point-1 hours of sleep. In Python, we can use `np.median` to do the calculations for us.

Measures of center: mode

The mode is the most frequent value in the data. If we count how many occurrences there are of each `sleep_total` and sort in descending order, there are 4 mammals that sleep for 12.5 hours, so this is the mode. The mode of the vore variable, which indicates the animal's diet, is herbivore. We can also find the mode using the mode function from the statistics module. Mode is often used for categorical variables, since categorical variables can be unordered and often don't have an inherent numerical representation.

Adding an outlier

Now that we have lots of ways to measure center, how do we know which one to use? Let's look at an example. Here, we have all of the insectivores in the dataset. We get a mean sleep time of 16.5 hours and a median sleep time of 18.9 hours. Now let's say we've discovered a new mystery insectivore that never sleeps. If we take the mean and median again, we get different results. The mean went down by more than 3 hours, while the median changed by less than an hour. This is because the mean is much more sensitive to extreme values than the median.

Which measure to use?

Since the mean is more sensitive to extreme values, it works better for symmetrical data like this. Notice that the mean, in black, and median, in red, are quite close.

Skew

However, if the data is skewed, meaning it's not symmetrical, like this, median is usually better to use. In this histogram, the data is piled up on the right, with a tail on the left. Data that looks like this is called left-skewed data. When data is piled up on the left with a tail on the right, it's right-skewed.

Which measure 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.

Exercise 1.1

Only show final grouped result

```
result = be_and_usa.groupby('country')['consumption'].agg(['mean', 'median'])
result
```

Mean and median

In this chapter, you'll be working with the 2018 Food Carbon Footprint Index from nu3. 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.

1. Import `numpy` with the alias `np`.
2. 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`.
3. Calculate the mean and median of kilograms of food consumed per person per year for both countries.

```
import warnings
warnings.filterwarnings("ignore")

# Import numpy with alias np
import numpy as np
import pandas as pd

# Importing the dataset
food_consumption = pd.read_csv("datasets/food_consumption.csv")

# Filter for Belgium
be_consumption = food_consumption[food_consumption['country'] == 'Belgium']
# OR
be_consumption = food_consumption.set_index('country').loc[['Belgium']]
```

```
# Filter for USA
usa_consumption = food_consumption[food_consumption['country'] == "USA"]
# OR
usa_consumption = food_consumption.set_index('country').loc[['USA']]

# Calculate mean and median consumption in Belgium
print(f"Mean consumption in Belgium is: {np.mean(be_consumption['consumption']):.4f}")
print(f"Median consumption in Belgium is: {np.median(be_consumption['consumption']):.4f}")

# Calculate mean and median consumption in USA
print(f"Mean consumption in USA is: {np.mean(usa_consumption['consumption']):.4f}")
print(f"Median consumption in USA is: {np.median(usa_consumption['consumption']):.4f}")

# 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 cou

# Subset for Belgium and USA only
be_and_usa = food_consumption[(food_consumption['country'] == 'Belgium') | (food_consumption['cou

# Group by country, select consumption column, and compute mean and median
print("\n The mean and median of the kilograms of food consumed per person per year in Belgium an
print(be_and_usa.groupby('country')['consumption'].agg([np.mean, np.median]))
```

```
Mean consumption in Belgium is: 42.1327
Median consumption in Belgium is: 12.5900
Mean consumption in USA is: 44.6500
Median consumption in USA is: 14.5800

 The mean and median of the kilograms of food consumed per person per year in Belgium and USA usi
              mean  median
country
Belgium  42.132727   12.59
USA      44.650000   14.58
```

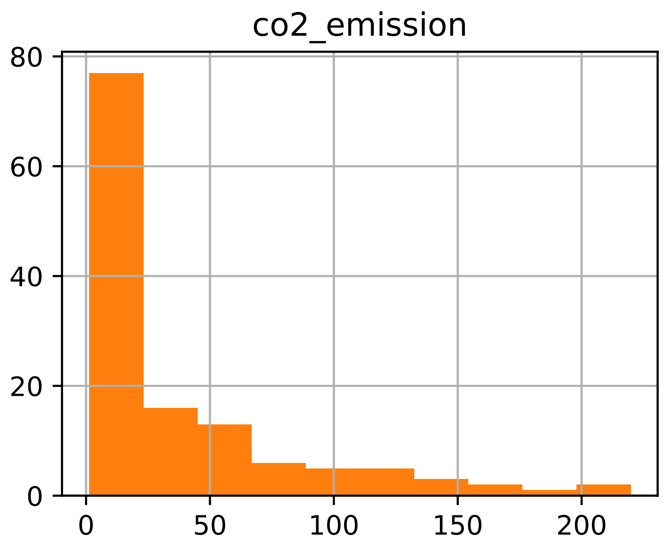
Exercise 1.2

1. Import `matplotlib.pyplot` with the alias `plt`.
2. Subset food_consumption to get the rows where `food_category` is 'rice'.
3. Create a histogram of `co2_emission` for rice and show the plot.
4. Use `.agg()` to calculate the mean and median of `co2_emission` for rice.

```
result = be_and_usa.groupby('country')['consumption'].agg(['mean', 'median'])
result
```

Mean and median

```
import warnings
warnings.filterwarnings("ignore")

# Import matplotlib.pyplot with the alias plt.
import matplotlib.pyplot as plt

# Subset food_consumption to get the rows where food_category is 'rice'.
rice_consumption = food_consumption[food_consumption['food_category'] == 'rice']

# Create a histogram of co2_emission for rice and show the plot.
rice_consumption.hist(column= 'co2_emission')
# or
rice_consumption['co2_emission'].hist()
plt.show()

# Use .agg() to calculate the mean and median of co2_emission for rice.
mean_rice, median_rice = mean_rice, median_rice = rice_consumption['co2_emission'].agg([np.mean,
print(f"Mean rice CO2 Emission: {mean_rice}")
print(f"Median rice CO2 Emission: {median_rice}")
```



```
Mean rice CO2 Emission: 37.59161538461538
Median rice CO2 Emission: 15.2
```

Chapter 1.2: Measure of Dispersion

Measures of spread

In this lesson, we'll talk about another set of summary statistics: 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

The first measure, variance, measures the average distance from each data point to the data's mean.

Calculating variance

To calculate the variance, we start by calculating the distance between each point and the mean, so we get one number for every data point. We then square each distance and then add them all together. Finally, we divide the sum of squared distances by the number of data points minus 1, giving us the variance. The higher the variance, the more spread out the data is. It's important to note that the units of variance are squared, so in this case, it's 19.8 hours squared. 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.

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 19.8 hours squared.

Mean absolute deviation

Mean absolute deviation takes the absolute value of the distances to the mean, and then takes the mean of those differences. While this is similar to standard deviation, it's not exactly the same. Standard deviation squares distances, so longer distances are penalized more than shorter ones, while mean absolute deviation penalizes each distance equally. One isn't better than the other, but SD is more common than MAD.

Quantiles

Before we discuss the next measure of spread, let's quickly talk about 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 0.5. 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. 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.

Boxplots use quartiles

The boxes in box plots represent quartiles. The bottom of the box is the first quartile, and the top of the box is the third quartile. The middle line is the second quartile, or the median.

Quantiles using np.linspace()

Here, we split the data in five equal pieces, but we can also use `np.linspace` as a shortcut, which takes in the starting number, the stopping number, and the number intervals. We can compute the same quantiles using 'np.linspace starting at zero, stopping at one, splitting into 5 different intervals.

Interquartile range (IQR)

The interquartile range, or IQR, is another measure of spread. It's the distance between the 25th and 75th percentile, which is also the height of the box in a boxplot. We can calculate it using the quantile function, or using the `iqr` function from `scipy.stats` to get 5.9 hours.

Outliers

Outliers are data points that are substantially different from the others. But how do we know what a substantial difference is ? A rule that's often used is that any data point less than the first quartile minus 1.5 times the IQR is an outlier, as well as any point greater than the third quartile plus 1.5 times the IQR.

Finding outliers

To find outliers, we'll start by calculating the IQR of the mammals' body weights. We can then calculate the lower and upper thresholds following the formulas from the previous slide. We can now subset the DataFrame to find mammals whose body weight is below or above the thresholds. There are eleven body weight outliers in this dataset, including the cow and the Asian elephant.

15. All in one go {.unnumbered}

Many of the summary statistics we've covered so far can all be calculated in just one line of code using the `.describe` method, so it's convenient to use when you want to get a general sense of your data.

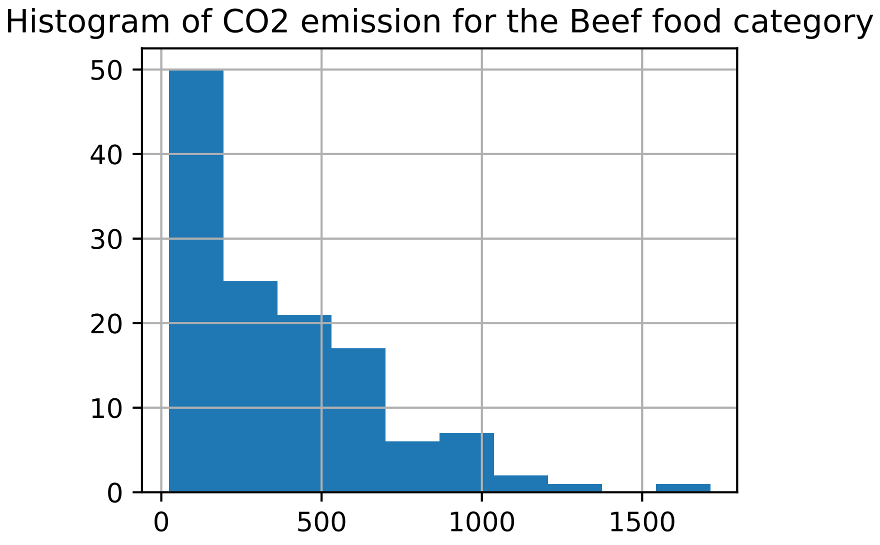
Exercise 1.2.1

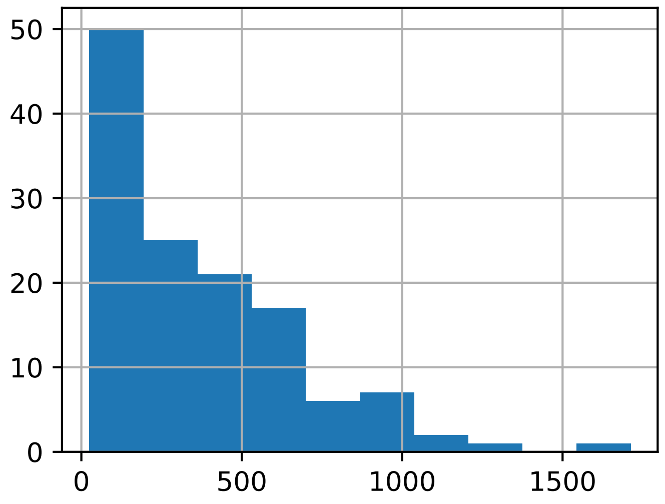
1. Calculate the quartiles of the `co2_emission` column of `food_consumption`.
2. Calculate the six quantiles that split up the data into 5 pieces (quintiles) of the `co2_emission` column of `food_consumption`.
3. Calculate the eleven quantiles of `co2_emission` that split up the data into ten pieces (deciles).
4. Calculate the variance and standard deviation of `co2_emission` for each food_category by grouping and aggregating.
5. Import matplotlib.pyplot with alias plt.
6. Create a histogram of `co2_emission` for the beef `food_category` and show the plot.
7. Create a histogram of `co2_emission` for the eggs `food_category` and show the plot.

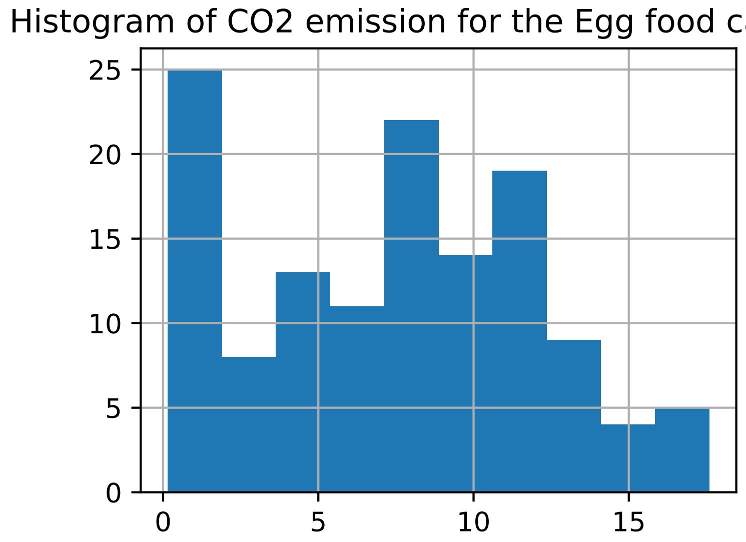
```python
import warnings
warnings.filterwarnings("ignore")

# Calculate the quartiles of the co2_emission column of food_consumption.
print("The quartiles of the co2_emission column of food_consumption: \n")
print(np.quantile(food_consumption['co2_emission'], np.linspace(0, 1, 5)))

# Calculate the six quantiles that split up the data into 5 pieces (quintiles) of the co2_emissio
print("The quintiles of the co2_emission column of food_consumption: \n")
print(np.quantile(food_consumption['co2_emission'], np.linspace(0, 1, 6)))

# Calculate the eleven quantiles of co2_emission that split up the data into ten pieces (deciles)
print("The 11 quantiles of the co2_emission column of food_consumption: \n")
print(np.quantile(food_consumption['co2_emission'], np.linspace(0, 1, 11)))

# Calculate the variance and standard deviation of co2_emission for each food_category by groupin
print("The variance and standard deviation of co2_emission for each food_category by grouping and
print(food_consumption.groupby('food_category')['co2_emission'].agg([np.var, np.std]))
```

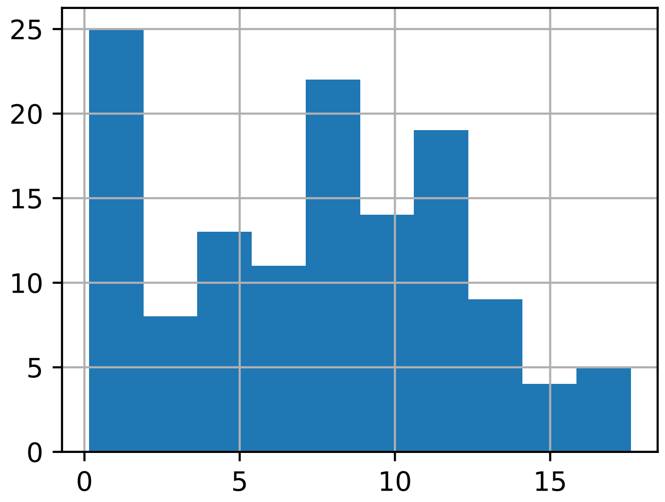
```
# Import matplotlib.pyplot with alias plt.
import matplotlib.pyplot as plt

# Create a histogram of co2_emission for the beef food_category and show the plot.
food_consumption[food_consumption['food_category'] == 'beef'].hist(column = 'co2_emission')
plt.title("Histogram of CO2 emission for the Beef food category")
plt.show()
# or
food_consumption[food_consumption['food_category'] == 'beef']['co2_emission'].hist()
plt.show()

# Create a histogram of co2_emission for the eggs food_category and show the plot.
food_consumption[food_consumption['food_category'] == 'eggs'].hist(column = 'co2_emission')
plt.title("Histogram of CO2 emission for the Egg food category")
plt.show()
# or
food_consumption[food_consumption['food_category'] == 'eggs']['co2_emission'].hist()
plt.show()
```

```
The quartiles of the co2_emission column of food_consumption:

[   0.        5.21     16.53     62.5975 1712.    ]
The quintiles of the co2_emission column of food_consumption:

[   0.       3.54    11.026   25.59    99.978 1712.   ]
The 11 quantiles of the co2_emission column of food_consumption:

[0.00000e+00 6.68000e-01 3.54000e+00 7.04000e+00 1.10260e+01 1.65300e+01
 2.55900e+01 4.42710e+01 9.99780e+01 2.03629e+02 1.71200e+03]
The variance and standard deviation of co2_emission for each food_category by grouping and aggreg

                        var         std
food_category
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
```


Histogram of CO2 emission for the Beef food category
0
10
20
30
40
50
0
500
1000
1500


0
10
20
30
40
50
0
500
1000
1500





Exercise 1.2.2

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. The outlier rule states that values are considered outliers if $x < Q_1 - 1.5 \times IQR$ or $x > Q_3 + 1.5 \times IQR$. In fact, this is how the lengths of the whiskers in a matplotlib box plot are calculated.

1. Calculate the total `co2_emission` per country by grouping by country and taking the sum of `co2_emission`. Store the resulting DataFrame as `emissions_by_country`.

2. Compute the first and third quartiles of `emissions_by_country` and store these as `q1` and `q3`.
3. Calculate the interquartile range of `emissions_by_country` and store it as `iqr`.
4. Calculate the lower and upper cutoffs for outliers of `emissions_by_country`, and store these as `lower` and `upper`.
5. 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 the total co2_emission per country by grouping by country and taking the sum of co2_e
emissions_by_country = food_consumption.groupby('country')['co2_emission'].sum()

print("The Total CO2 Emission by Country \n")
print(emissions_by_country)

# Compute the first and third quartiles of emissions_by_country and store these as q1 and q3.
q1 = np.quantile(emissions_by_country, 0.25)
q3 = np.quantile(emissions_by_country, 0.75)

# Calculate the interquartile range of emissions_by_country and store it as iqr.
iqr = q3 - q1

# Calculate the lower and upper cutoffs for outliers of emissions_by_country, and store these as
lower = q1 - 1.5 * iqr
upper = q3 + 1.5 * iqr

# Subset emissions_by_country to get countries with a total emission greater than the upper cutof
outliers = emissions_by_country[(emissions_by_country < lower) | (emissions_by_country > upper)]
print("The Outliers (countries with a total emission greater than the upper cutoff or a total emi
print(outliers)
```

```
The Total CO2 Emission by Country

country
Albania      1777.85
Algeria       707.88
Angola        412.99
Argentina    2172.40
Armenia      1109.93
               ...
Uruguay      1634.91
Venezuela    1104.10
Vietnam       641.51
Zambia        225.30
Zimbabwe      350.33
Name: co2_emission, Length: 130, dtype: float64
```

```
The Outliers (countries with a total emission greater than the upper cutoff or a total emission l
country
Argentina    2172.4
Name: co2_emission, dtype: float64
```

Chapter 2

Chapter 2.1: Random Numbers and Probability

What are the 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. This gives us one half, or a fifty percent chance of getting heads. Probability is always between zero and 100 percent. If the probability of something is zero, it's impossible, and if the probability of something is 100%, it will certainly happen.

Assigning salespeople

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%.

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. However, if we run the same thing again, we may get a different row since the sample method chooses randomly. If we want to show the team how we picked Brian, this won't work well.

Setting a random seed

To ensure we get the same results when we run the script in front of the team, we'll set the random seed using `np.random.seed`. The seed is a number that Python's random number generator uses as a starting point, so if we orient it with a seed number, it will generate the same random value each time. The number itself doesn't matter. We could use 5, 139, or 3 million. The only thing that matters is that we use the same seed the next time we run the script. Now, we, or one of the sales-team members, can run this code over and over and get Brian every time.

A second meeting

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%.

Sampling twice in Python

To recreate this in Python, we can pass 2 into the sample method, which will give us 2 rows of the DataFrame.

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%.

Sampling with/without replacement in Python

To sample with replacement, set the `replace` argument to `True`, so names can appear more than once. If there were 5 meetings, all at different times, it's possible to pick some rows multiple times since we're replacing them each time.

Independent events

Let's quickly talk about independence. Two events are independent if the probability of the second event isn't affected by the outcome of the first event. For example, if we're sampling with replacement, the probability that Claire is picked second is 25%, no matter who gets picked first. In general, when sampling with replacement, each pick is independent.

Dependent events

Similarly, events are considered dependent when the outcome of the first changes the probability of the second. If we sample without replacement, the probability that Claire is picked second depends on who gets picked first. If Claire is picked first, there's 0% probability that Claire will be picked second. If someone else is picked first, there's a 33% probability Claire will be picked second. In general, when sampling without replacement, each pick is dependent.

Exercise 2.1.1: Calculating probabilities

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.

1. Count the number of deals Amir worked on for each product type and store in counts.
2. 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`.

```python
# Importing the dataset
amir_deals = pd.read_csv("datasets/amir_deals.csv")

# Count the number of deals Amir worked on for each product type and store in counts.
counts = amir_deals['product'].value_counts()
# or
counts = amir_deals.value_counts('product')
print(f"The number of deals Amir worked on for each product type: {counts}")

# Calculate the probability of selecting a deal for the different product types by dividing the c
probs = amir_deals['product'].value_counts()/178
print(f"The probability of selecting a deal for the different product types: {probs}")
```

```
The number of deals Amir worked on for each product type: product
Product B    62
Product D    40
Product A    23
Product C    15
Product F    11
Product H     8
Product I     7
Product E     5
Product N     3
Product G     2
Product J     2
```

```
Name: count, dtype: int64
The probability of selecting a deal for the different product types: product
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
Name: count, dtype: float64
```

Exercise 2.1.2 : Sampling deals

In the previous exercise Section , 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.

1. Import the necessary packages.
2. Set the random seed to 24.
3. Take a sample of 5 deals without replacement and store them as `sample_without_replacement`.
4. Take a sample of 5 deals with replacement and save as `sample_with_replacement`.

```python
import pandas as pd
import numpy as np

# Set the random seed to 24.
np.random.seed(24)

# Take a sample of 5 deals without replacement and store them as sample_without_replacement.
sample_without_replacement = amir_deals.sample(5)
print("Sample of 5 deals without replacement \n")
print(sample_without_replacement)

# Take a sample of 5 deals with replacement and save as sample_with_replacement.
sample_with_replacement = amir_deals.sample(5, replace = True)
print("Sample of 5 deals with replacement \n")
print(sample_with_replacement)
```

```
Sample of 5 deals without replacement

     Unnamed: 0    product   client status   amount  num_users
127         128  Product B  Current    Won  2070.25          7
148         149  Product D  Current    Won  3485.48         52
77           78  Product B  Current    Won  6252.30         27
104         105  Product D  Current    Won  4110.98         39
166         167  Product C      New   Lost  3779.86         11
Sample of 5 deals with replacement

     Unnamed: 0    product   client status   amount  num_users
133         134  Product D  Current    Won  5992.86         98
101         102  Product H  Current    Won  5116.34         63
110         111  Product B  Current    Won   696.88         44
49           50  Product B  Current    Won  3488.36         79
56           57  Product D  Current    Won  6820.84         42
```

What type of sampling is better to use for this situation? If you sample with replacement, you might end up calling the same customer twice.

Chapter 2.2: Discrete distributions

In this lesson, we'll take a deeper dive into probability and begin looking at probability distributions.

Rolling the dice

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.

Choosing salespeople

This is similar to the scenario from earlier, except we had names instead of numbers. Just like rolling a die, each outcome, or name, had an equal chance of being chosen.

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.

Visualizing a probability distribution

We can visualize this using a barplot, where each bar represents an outcome, and each bar's height represents the probability of that outcome.

Probability = area

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? To figure this out, we'll take the area of each bar representing an outcome of 2 or less. 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.

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.

Visualizing uneven probabilities

When we visualize these new probabilities, the bars are no longer even.

Adding areas

With this die, what's the probability of getting something less than or equal to 2? There's a one-sixth probability of getting 1, and zero probability of getting 2, which sums to one-sixth.

Discrete probability distributions

The probability distributions you've seen so far are both discrete probability distributions, since they represent situations with discrete outcomes. Recall from chapter 1, Section that discrete variables can be thought of as counted variables. In the case of a die, we're counting dots, so we can't roll a 1.5 or 4.3. When all outcomes have the same probability, like a fair die, this is a special distribution called a discrete uniform distribution.

Sampling from discrete distributions

Just like we sampled names from a box, we can do the same thing with probability distributions like the ones we've seen. Here's a DataFrame called die that represents a fair die, and its expected value is 3.5. We'll sample from it 10 times to simulate 10 rolls. Notice that we sample with replacement so that we're sampling from the same distribution every time.

Visualizing a sample

We can visualize the outcomes of the ten rolls using a histogram, defining the bins we want using `np.linspace`.

Sample distribution vs. theoretical distribution

Notice that we have different numbers of 1's, 2's, 3's, and so on since the sample was random, even though on each roll we had the same probability of rolling each number. The mean of our sample is 3.0, which isn't super close to the 3.5 we were expecting.

A bigger sample

If we roll the die 100 times, the distribution of the rolls looks a bit more even, and the mean is closer to 3.5.

An even bigger sample

If we roll 1000 times, it looks even more like the theoretical probability distribution and the mean closely matches 3.5.

Law of large numbers

This is called the law of large numbers, which is the idea that as the size of your sample increases, the sample mean will approach the theoretical mean.

Exercise 2.2: 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` DataFrame.

1. Create a histogram of the `group_size` column of `restaurant_groups`, setting bins to [2, 3, 4, 5, 6]. Remember to show the plot.
2. 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`.
3. Reset the index of `size_dist`.
4. Rename the columns of `size_dist` to `group_size` and `prob`.
5. Calculate the expected value of the `size_dist`, which represents the expected group size, by multiplying the `group_size` by the `prob` and taking the `sum`.
6. 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.
7. Sum the probabilities of `groups_4_or_more`.

```
# Create a histogram of the group_size column of restaurant_groups, setting bins to [2, 3, 4, 5,
restaurant_groups['group_size'].hist(bins = [2, 3, 4, 5, 6] )
plt.show()

# Count the number of each group_size in restaurant_groups, then divide by the number of rows in
# Reset the index of size_dist.
# Rename the columns of size_dist to group_size and prob.
size_dist = restaurant_groups['group_size'].value_counts() / restaurant_groups.shape[0]
# Reset index and rename columns
size_dist = size_dist.reset_index()
size_dist.columns = ['group_size', 'prob']

# Calculate the expected value of the size_dist, which represents the expected group size, by mul
expected_value = (size_dist['group_size'] * size_dist['prob']).sum()
# Or
expected_value = np.sum(size_dist['group_size'] * size_dist['prob'])
print(expected_value)

# Calculate the probability of randomly picking a group of 4 or more people by subsetting for gro
groups_4_or_more = size_dist[size_dist['group_size'] >= 4]

# Sum the probabilities of groups_4_or_more
prob_4_or_more = np.sum(groups_4_or_more['prob'])
print(prob_4_or_more)
```

Special note

- You learned about the basics of probability distributions, focusing on discrete distributions, and how they apply to real-world scenarios. Specifically, you explored:
- Probability Distributions: Understanding that a probability distribution describes the likelihood of each possible outcome in a scenario, like rolling a six-sided die where each outcome has an equal chance.
- Expected Value: Learning to calculate the expected value of a distribution as the mean, demonstrated by multiplying each outcome's value by its probability and summing these products. For a fair die, the expected value is 3.5.
- Visualizing Distributions: How to visualize probability distributions with bar plots, where each bar's height represents the outcome's probability, and histograms for sample outcomes.
- Discrete Uniform Distribution: Identifying when all outcomes have the same probability, such as with a fair die, this represents a discrete uniform distribution.
- Sampling and the Law of Large Numbers: Through examples, you saw how sampling from a distribution (like rolling a die multiple times) and calculating the sample mean can illustrate the law of large numbers. The larger the sample, the closer the sample mean will be to the theoretical mean.

```python
# Example of calculating expected value for a fair die
expected_value = sum([i * (1/6) for i in range(1, 7)])
```

Chapter 2.3 Continuous distributions

We can use discrete distributions to model situations that involve discrete or countable variables, but how can we model continuous variables?

Waiting for the bus

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.

Continuous uniform distribution

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.

Continuous uniform distribution

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%.

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%.

"Greater than" probabilities

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.

Combining multiple uniform.cdf() calls

How do we calculate the probability of waiting 4 to 7 minutes using Python? We can start with the probability of waiting less than 7 minutes, then subtract the probability of waiting less than 4 minutes. This gives us 25%.

Total area = 1

To calculate the probability of waiting between 0 and 12 minutes, we multiply 12 by 1/12, which is 1, or 100%. This makes sense since we're certain we'll wait anywhere from 0 to 12 minutes.

Generating random numbers according to uniform distribution

To generate random numbers according to the uniform distribution, we can use `uniform.rvs`, which takes in the minimum value, maximum value, followed by the number of random values we want to generate. Here, we generate 10 random values between 0 and 5.

Other continuous distributions

Continuous distributions can take forms other than uniform where some values have a higher probability than others. No matter the shape of the distribution, the area beneath it must always equal 1.

Other special types of distributions

This will also be true of other distributions you'll learn about later on in the course, like the normal distribution or exponential distribution, which can be used to model many real-life situations.

Exercise 2.3.1

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.

1. 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`.
2. Calculate the probability that Amir has to wait more than 5 minutes, and store in a variable called `prob_greater_than_5`.
3. Calculate the probability that Amir has to wait between 10 and 20 minutes, and store in a variable called `prob_between_10_and_20`.

```python
min_time = 0
max_time = 30

from scipy.stats import uniform
prob_less_than_5 = uniform.cdf(5, 0, 30)
print(f"The probability that Amir has to wait less than 5 minutes: {prob_less_than_5}")

# Calculate the probability that Amir has to wait more than 5 minutes, and store in a variable ca

prob_greater_than_5 = 1 - uniform.cdf(5, 0, 30)
print(f"The probability that Amir has to wait more than 5 minutes: {prob_greater_than_5}")
```

```
# Calculate the probability that Amir has to wait between 10 and 20 minutes, and store in a varia
prob_between_10_and_20 = uniform.cdf(20, 0, 30) - uniform.cdf(10, 0, 30)
print(f"The probability that Amir has to wait between 10 and 20 minutes: {prob_between_10_and_20}
```

```
The probability that Amir has to wait less than 5 minutes: 0.16666666666666666
The probability that Amir has to wait more than 5 minutes: 0.8333333333333334
The probability that Amir has to wait between 10 and 20 minutes: 0.3333333333333333
```

Exercise 2.3.2

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.

1. Set the random seed to 334.
2. Import `uniform` from `scipy.stats`.
3. Generate 1000 wait times from the continuous uniform distribution that models Amir's wait time. Save this as `wait_times`.
4. Create a histogram of the simulated wait times and show the plot.

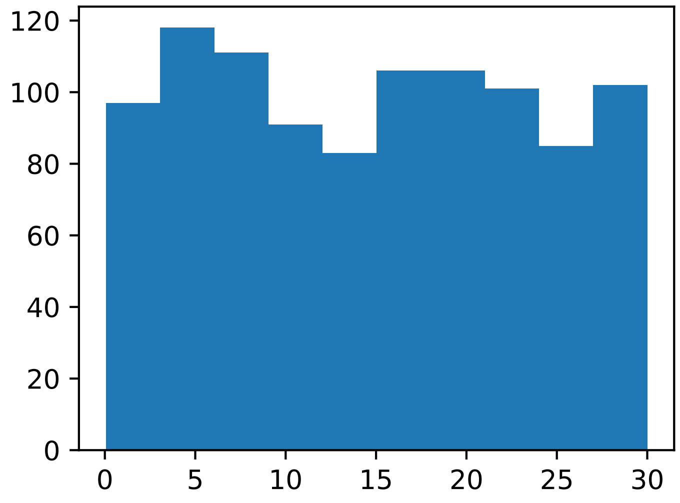
```
# Set the random seed to 334.
np.random.seed(334)

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

# Generate 1000 wait times from the continuous uniform distribution that models Amir's wait time.
wait_times = uniform.rvs(0, 30, size = 1000)
print("The Wait times Distribution \n")
print(wait_times)

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

```
The Wait times Distribution

[ 7.144097    0.97455866  3.72802787  5.11644319  8.70602482 24.69140099
 23.98012075  3.19592668 25.1985306  17.89048629 24.68695356 18.27160808
 22.85829011  0.12929581 15.67789664  2.3262095  15.79151771 21.90473557
 18.25817257 15.78657023 28.74328434  9.38971275 17.0010565   0.95433991
 15.96917606  2.16395679  8.903302   19.24982156  6.52414731 14.10185677
```

```
27.86973455 14.38121016 28.59233824 13.17550521 22.96382405  2.52904371
 6.2163781   8.40217036  4.48079914 28.16365298 19.73302715  3.63959282
 1.74002295  1.5324219  26.97096908 29.37490861  4.71379092  6.44990488
 6.81999564 22.81302458 12.41563042 11.14591581  8.08043184 29.60574053
 3.24571518 19.66709277 13.38939302 29.56433707 24.84697258  6.249684
15.07668579  5.27474477 27.27430815  2.42084219 27.88317515  0.81231424
 3.48564273 19.80738592  6.11128124 19.10323692  9.12156821 28.31696467
20.80158047 17.0840986  26.59969632 28.38502685 20.75398728 11.73610958
20.55950389 18.20349931  4.68857179 17.67638417 29.99091109 18.67756789
11.16391438  3.53028943 14.93882748 24.89203249 17.47310051 20.8740314
16.6070177  19.19564265  8.96414904  5.28451257  1.05350993 21.77737031
23.61684528 22.72809506 24.8322021  14.36218169  8.6091251  25.13656432
11.45898163 19.92575672 25.12266034  7.63273768  5.0240165  20.92435348
13.23933607  0.6401269  15.6813087  13.98669465  5.00961969 26.55407346
15.6478544   5.36392254 22.2807534  22.95452053 12.09657902 15.33347354
29.36367996 27.32761079 16.57775534 13.28054948 17.09693671 27.45768651
 2.59501602  9.41004409  9.14688319 19.72368555  0.8399502   1.36535784
 8.24745591 10.2650512  29.95498509 20.64562623  5.93986334 11.17818273
26.32817252  8.07638083  6.6348515  16.29376148 29.99092275 25.71567927
11.55546768 21.91960495 11.23343293 12.37874514 15.0825809  28.02838931
22.8535862   8.8911682  27.94808637 18.59880054 19.31673054  4.78414772
24.68245155  4.61858339  4.34185585  9.86285797 22.64196176 20.70638537
 2.07610021 16.5432359   0.83932634 26.55708037 11.96557209  8.07255521
10.80342781 23.29099616 22.86574237 14.34997804  5.88657148 13.93691899
 3.85622609 15.39470306 21.25310818 18.36870054  8.06383855 27.35806466
 8.24616015 21.22703251 17.37235409  4.83719581 17.84253763  3.31270513
28.05979944 28.42309918  8.73094692 25.71299775  2.22366188  4.30187426
21.89436597 29.49072021 15.30664125 22.3802789   0.08102944 14.78896917
16.40399119  7.68390804  2.10437223  6.68794817 10.19108918 22.31258787
 5.8189225  27.60781938 19.51339944 21.33124269  1.6294243  15.51174843
13.47212795 16.24027513 22.10266715  5.52406499  7.22056009  7.36829948
18.91081125 19.80927487 12.80401324  2.70031211 26.86102574 22.01951095
17.17665123 18.81827876  3.67837051 15.46293386 29.51597515 15.89595847
17.41855781 25.55857784 25.65737763 16.4151132   5.00323267 11.63197386
17.66808546 25.466024   15.18486451  0.85940046 27.00984537 14.42468181
27.4812224   9.47231537  2.05988889 17.30621667 24.95053845  5.57417231
16.0357139  13.26420998 29.46815429  1.16904392  4.53575356 10.65537707
11.26270564  6.0214732  27.64202445 10.29940068 28.54836132 22.22931556
17.55762098 28.31703818  3.20547158  6.51317017  9.60103128 11.55504752
 1.10769737 25.19601111 26.41265999  3.17669809 29.03461951 23.78452941
22.35394515 18.03280452 11.23378656 25.18138124 23.9597556  10.58566727
 2.56203807 16.32353266  8.34705721 21.65497161 27.82670318  3.24006097
 9.48180202  1.48689026 12.89086955  0.84704402 21.10201173  9.14793573
27.17361576 18.45069642 14.49691685  3.98370201  0.44134824  5.05259221
18.47816542  8.11334339 19.95136747  7.1754582  20.10018729  4.97982617
```

```
 8.81527565  5.76132824 15.72125645  4.76049255  4.76730185  4.54755451
11.99610604 22.19271804  7.0054561   0.9554475  12.72622458  5.10178612
 3.46974455  7.75842307 13.13578589  3.8451399  11.02848319  5.77758805
16.19423275 23.15484726 20.1239129   8.50840771 10.91441361  7.11704929
20.25016912 28.69395694 20.42004446 24.35824921 24.56208344 18.70863082
 1.55698104 27.49671811  2.57368945 23.03303579  2.38875615 13.98131955
21.8838685   8.44024292 25.05019036 21.95993093  8.76034493 22.85418651
26.07063198  5.46361622 25.58172855 15.67874872 17.81956733 26.81841668
13.19033546 23.19910687 23.62718492 13.15663104 23.41668296  1.87489683
25.2484807   8.4845543   4.92042842 22.73961665 20.2466653  19.58723259
 2.03270822 22.9895573   7.3372969  17.54273192 21.09771191 16.96192787
 3.48948107 16.15590988 17.70440831  1.60420151 19.81559878 25.80086106
15.68959678 18.83009183 23.19198615 12.66930187  7.00118096  6.64600958
19.87400439 19.55842619 25.54278522 14.27653959 12.0734572  22.78237983
17.15165279 24.08463516 28.59624819 28.16152099  5.93891699 16.94105605
20.80665928  9.99890108  8.79730012  7.05110922  3.8104523   5.62610024
 5.58639014 18.78260685  9.95891585 20.62479473 28.77797774 25.81469814
 2.25609181 13.65905187 21.1528068   8.01290913  7.73848867 26.88193447
21.44953343 10.12615075 19.75731318  4.27872352 20.30849315 26.77882578
 1.351449   20.8075843   0.46173517 19.80997905  9.7624355  17.37924972
 1.43062491 22.20157066 25.1824487  28.8357933  16.09702531 24.73842767
 0.2308042   6.13048075 29.614561    8.33629603 16.12795224 29.21240454
13.45025092 28.92901674 28.68113999 18.76015795  8.85400148  0.72177506
18.5678127   1.5946721   6.51114298  0.14954964  8.40759558 15.3456776
18.16257912 21.48146244 19.35734786  6.69307887 11.14786018  5.19542535
20.11979092 15.36643858  1.75976538  1.88235419 14.67047828 17.87206607
23.89728101 11.53568193 19.55519337 12.9925055   3.75560435  8.84921298
10.41509647 10.05015649  2.08203941 13.95507535 18.69687686  3.65394569
16.73715719 25.9668531  19.90170356 10.66486523 27.98606183 19.33610382
27.37475735 24.1411498  28.39648506 10.56397635 17.67065935 19.81446104
 5.08556252 21.61589418  4.91334443 10.25571404 13.86547777 28.5015485
19.52083861 22.05488912  9.17148377  1.11732389 18.30445657 13.80853
 2.00270834 26.65687785  8.33324258 16.37203881 24.89846207  8.62610704
29.82242843 16.81338141 17.33734435  1.0612399  29.2449401   6.66842773
14.65276829 11.46127532  5.94042969  8.51718703 19.16192131 27.77484802
26.80478414 18.46459441 14.64021886 10.41881806  5.05273386 22.93123257
12.32037453 29.07333784 19.8933068  19.33393445 23.75624561 27.41576612
 4.89400728 27.97022569  5.46431449 26.39553105 23.10148492  5.59326106
 5.43438053 12.80881651 22.58136604  6.81522341  1.29891598  8.92824717
 0.06222649  0.31415758  1.49504525 14.46068646 17.65853438  7.4496374
17.22446855 21.84287476 21.60599421 28.86047834 28.24253893  7.10948381
26.95417677 22.69920835 14.85402627  8.3681587  29.58708806 13.78486659
17.35864925 26.46297136 21.22739031 28.53046947  8.93805023  7.02914326
11.80133052 18.99940059 14.08892962  3.85612353 13.56279119 20.12274586
16.74078625 29.53044614  5.18316362 11.53812907 27.43654492  0.42476924
```

```
29.87062735  5.50874294  3.8568494   6.41204685 20.30702925  3.32308882
 3.94770677  6.87059867  3.93014852 21.98204107  0.86507581 20.56982783
10.47517828  5.51681714 28.14377379 29.45286026 11.06512511 16.61676719
 1.98579434  5.54260989  1.07592481  2.31764882  8.88290412 21.4881271
22.64224754  7.27994039  9.37643238 10.83908073  0.69663121 21.23717882
 9.66027816 16.97956524 14.81638878 16.5534871   9.75601043  5.93087217
28.36797115 21.09462207 13.03173376  0.7983171   3.50516639 19.00649926
 5.82081691  5.81833671 20.61983383 28.71056507  6.91237663 12.4619051
12.30761644 19.84040954  4.64447622  0.64917761 19.86567877  5.99418851
 1.35794716 27.48067897 29.73644518 28.6065078   8.5299145   4.12152041
 0.5607774  18.57081877 22.17373158  7.58966957 18.2784296  10.70211771
11.13177466 29.30678999  1.2423198  14.35713294 17.35500663 17.14866592
10.3789623  24.17742423 14.94162775  2.59992154 11.85508506 27.4947809
22.37927752 16.63597764  2.18654009 27.58998237 24.9960839  18.97091648
19.87097998 29.71143066 18.15641786  8.02594757 12.34873726  6.39950324
26.71285573  5.72013105 27.25611587 11.11118651 13.14563054 24.3161132
 9.12227066  6.09164062  7.61512695 15.27894726  1.55172543 16.11462964
20.01927489  0.36023495 22.18975204 10.68017381 26.67207213 18.49390616
14.12316947 12.30157185 24.46937392  2.94349773  2.71721257 19.89328934
11.60564868 20.99716418  6.83938153  3.15126472 21.18057782 23.04084254
29.52319489 13.41468779  8.36540008  1.17474372 16.52320975 16.29079038
23.68058086  9.73990339 21.21125025 11.64009223  8.27581985 25.34508281
27.14020237 20.99498767 26.36379634 22.12460919 14.38182874 12.83843751
 2.4921115   3.30654254 15.38775096  9.47486316 16.53719202 12.18243936
 6.94896378  5.99117553 16.06222869  5.82268307 10.69273715  8.58678605
27.77414396  6.29565658 22.85039389 22.68722007  0.43084526 19.12501901
23.05647004  6.23804663 16.49611786 10.854354   12.4620608  26.2548841
19.76816626  6.54492209 28.71514653  7.80202629 16.96751571 11.15377563
22.80092952  5.09403519 24.00001246  8.21769557 13.53560859  2.94076629
 4.02757203  8.96659938  6.86912639 12.0326642  27.15751025 15.4782868
 7.22781653 17.19415714 10.47116023 21.49491811  5.4747806  26.51862933
26.16289208 29.74722369 26.5735013  16.7792301  15.31299374  5.97510684
14.539767    5.34494067  8.90650139 13.70810391  0.63137974  9.78624106
22.98898991 21.52163702  5.36747243  2.08048168  7.42243152 25.8369847
25.40242175 13.07497422 28.66966768 24.04491316 21.29637332  7.42492739
 2.11604443 29.49179095  4.27645773  8.61358603 17.16555207 26.59803794
13.78944191 15.62067191  9.21670849 12.14562662 10.52617782 25.98579879
14.39787573  8.08763319 16.78552727 22.79597812  8.7695133  17.27072338
24.43151203 16.88485001 24.44534146 16.21756467 13.61941002 19.87394843
12.63576776 14.50185412 26.83264391  3.00081583 16.48057545  2.45212633
18.83439311 13.19865842  9.40766691 15.60923545  1.63933761 23.76828936
10.15252566  0.40447879 24.59292978  1.3675066   9.71814934 12.16659686
23.04867235 20.03414656  8.5168254  19.59414161  6.77825259  8.8070403
 5.87152425  2.5970591   4.93086037 21.00689143 10.53794997  4.50059669
 4.80161712 19.86587841 13.82231035 25.92370876 16.76861995 19.88191792
 5.11770677 29.00105159 25.65446515  6.84179709 22.57233685  8.15577205
24.13735729  4.03149435  4.13231282 16.12464321 22.48533776 20.282976
 5.74012718 22.43342651  2.60477234  3.21766129  5.65283575 28.84370986
23.65634284 21.7798963  16.85637734  0.61478999  4.53882038  4.65444933
 9.04857098 19.63428333 15.00766938 17.76592972 21.46082362 25.77459268
19.85548771  8.20584755  8.47650143 21.69630957  3.13097349 10.20447772
29.66564838 23.56946713 26.2778947  23.81853285 10.81655255  6.05639951
 2.85441413 19.56517693 23.71136826 20.36624573  6.59131157 23.74895849
22.42574624 10.31297875  6.60447634  5.37750477 29.10394928 12.33961909
14.61065264 27.83726115  5.16829978 19.01401557  5.2640168  10.30361904
24.32169445  3.18704573 25.06890427 26.01541021 13.94792952 21.25266832
25.93658805 16.80375076  1.50816509  2.64600638 13.47500562 15.08548187
14.90796765 20.88436957 22.53457433 15.84915157  8.86204958 27.73245412
11.22385246 15.78749796 15.91197569 22.37113884  3.19226887  4.65976517
 9.30923397 10.77526643  3.97151627 29.28059273 29.27602822 20.71976894
18.60895651  4.29611269 23.93608872 22.58290622  4.47426371  7.2489746
15.22674952 27.63320022  8.78304562 28.40607909 17.1539386   2.61718331
10.65596075 29.8980006  29.91966885 27.34174997  0.61629001 15.17716754
 9.90825324 16.72368794 12.96779625 10.46723235  1.79071474  4.91140186
 0.71891742 27.83882355 26.95599526 19.63276655 25.02837667  6.70120501
27.8039181   3.93032514 29.20218039 20.45922096 18.39870488  6.64103042
15.9427296  29.26956198 29.75236465  6.24029179 10.9032813  25.74945237
19.34538144 16.31296664 10.93219849 10.70922385 21.19432171 10.39189311
 1.8610141  24.11741202 25.59864155  0.68627027 15.7876837   1.10010957
 3.47094738 27.61646738 10.07577678 19.84021078 27.29452887  8.52034156
 5.181769   12.92311547 14.25423041 26.70151037 27.44545754 24.06119139
14.00076717  8.56031135 25.99043117 20.11722212]
```



Chapter 2.4 The binomial distribution

It's time to further expand your toolbox of distributions. In this lesson, you'll learn about the binomial distribution.

Coin flipping

We'll start by flipping a coin, which has two possible outcomes, heads or tails, each with a probability of 50%.

Binary outcomes

This is just one example of a binary outcome, or an outcome with two possible values. We could also represent these outcomes as a 1 and a 0, a success or a failure, and a win or a loss.

A single flip

In Python, we can simulate this by importing `binom` from `scipy.stats` and using the `binom.rvs` function, which takes in the number of coins we want to flip, the probability of heads or success, and an argument called size, which is number of trials. size is a named argument, so we'll need to explicitly specify that the third argument corresponds to size, or we'll get incorrect results. This call will return a 1, which we'll count as a head, or a 0, which we'll count as tails. We can use `binom.rvs` 1, 0.5, size equals 1 to flip 1 coin, with a 50% probability of heads, 1 time.

One flip many times

To perform eight coin flips, we can change the size argument to 8, which will flip 1 coin with a 50% chance of heads 8 times. This gives us a set of 8 ones and zeros.

Many flips one time

If we swap the first and last arguments, we flip eight coins one time. This gives us one number, which is the total number of heads or successes.

Many flips many times

Similarly, we can pass 3 as the first argument, and set size equal to 10 to flip 3 coins. This returns 10 numbers, each representing the total number of heads from each set of flips.

Other probabilities

We could also have a coin that's heavier on one side than the other, so the probability of getting heads is only 25%. To simulate flips with this coin, we'll adjust the second argument of `binom.rvs` to 0.25. The result has lower numbers, since getting multiple heads isn't as likely with the new coin.

Binomial distribution

The binomial distribution describes the probability of the number of successes in a sequence of independent trials. In other words, it can tell us the probability of getting some number of heads in a sequence of coin flips. Note that this is a discrete distribution since we're working with a countable outcome. The binomial distribution can be described using two parameters, `n` and `p`. `n` represents the total number of trials being performed, and `p` is the probability of success. `n` and `p` are also the third and second arguments of `binom.rvs`. Here's what the distribution looks like for 10 coins. We have the biggest chance of getting 5 heads total, and a much smaller chance of getting 0 heads or 10 heads.

What's the probability of 7 heads?

To get the probability of getting 7 heads out of 10 coins, we can use `binom.pmf`. The first argument is the number of heads or successes. The second argument is the number of trials, n, and the third is the probability of success, p. If we flip 10 coins, there's about a 12% chance that exactly 7 of them will be heads.

What's the probability of 7 or fewer heads?

`binom.cdf` gives the probability of getting a number of successes less than or equal to the first argument. The probability of getting 7 or fewer heads out of 10 coins is about 95%.

What's the probability of more than 7 heads?

We can take 1 minus the probability of getting 7 or fewer heads to get the probability of a number of successes greater than the first argument.

Expected value

The expected value of the binomial distribution can be calculated by multiplying `n` times `p`. The expected number of heads we'll get from flipping 10 coins is 10 times 0.5, which is 5.

Independence

It's important to remember that in order for the binomial distribution to apply, each trial must be independent, so the outcome of one trial shouldn't have an effect on the next. For example, if we're picking randomly from these cards with zeros and ones, we have a 50-50 chance of getting a 0 or a 1. But since we're sampling without replacement, the probabilities for the second trial are different due to the outcome of the first trial. Since these trials aren't independent, we can't calculate accurate probabilities for this situation using the binomial distribution.

Exercise 2.4.1: Simulating sales deals

Assume that Amir usually works on 3 deals per week, and overall, he wins 30% of deals he works on. Each deal has a binary outcome: it's either lost, or won, so you can model his sales deals with a binomial distribution. In this exercise, you'll help Amir simulate a year's worth of his deals so he can better understand his performance.

1. Import binom from scipy.stats and set the random seed to 10.
2. Simulate 1 deal worked on by Amir, who wins 30% of the deals he works on.
3. Simulate a typical week of Amir's deals, or one week of 3 deals.
4. Simulate a year's worth of Amir's deals, or 52 weeks of 3 deals each, and store in `deals`.
5. Print the mean number of deals he won per week.

```python
# Import binom from scipy.stats and set the random seed to 10.
from scipy.stats import binom
np.random.seed(10)

# Simulate 1 deal worked on by Amir, who wins 30% of the deals he works on.
print(f"Probability of 30% won: {binom.rvs(1, 0.3, size= 1)}")

# Simulate a typical week of Amir's deals, or one week of 3 deals.
print(f"Probability of 1 week of 3 deals: {binom.rvs(3, 0.3, size = 1)}")

# Simulate a year's worth of Amir's deals, or 52 weeks of 3 deals each, and store in deals.
# Print the mean number of deals he won per week.
deals = binom.rvs(3, 0.3, size = 52)
print(f"The mean number of deals he won per week: {np.mean(deals)}")
```

```
Probability of 30% won: [1]
Probability of 1 week of 3 deals: [0]
The mean number of deals he won per week: 0.8461538461538461
```

Exercise 2.4.2: Calculating binomial probabilities

Just as in the last exercise, Section , assume that Amir wins 30% of deals. He wants to get an idea of how likely he is to close a certain number of deals each week. In this exercise, you'll calculate what the chances are of him closing different numbers of deals using the binomial distribution.

1. What's the probability that Amir closes 1 or fewer deals in a week? Save this as `prob_less_than_or_equal_1`.
2. What's the probability that Amir closes more than 1 deal? Save this as `prob_greater_than_1`.

```
prob_3 = binom.pmf(3, 3, 0.3)
print(f"The likely he is to close a certain number of deals each week: {prob_3}")

# What's the probability that Amir closes 1 or fewer deals in a week? Save this as prob_less_than
prob_less_than_or_equal_1 = binom.cdf(1, 3, 0.3)
print(f"The probability that Amir closes 1 or fewer deals in a week: {prob_less_than_or_equal_1}"

# What's the probability that Amir closes more than 1 deal? Save this as prob_greater_than_1.
prob_greater_than_1 = 1 - binom.cdf(1, 3, 0.3)
print(f"Tthe probability that Amir closes more than 1 deal: {prob_greater_than_1}")
```

```
The likely he is to close a certain number of deals each week: 0.027
The probability that Amir closes 1 or fewer deals in a week: 0.784
Tthe probability that Amir closes more than 1 deal: 0.21599999999999997
```

Exercise 2.4.3: How many sales will be won?

Now Amir wants to know how many deals he can expect to close each week if his win rate changes. Luckily, you can use your binomial distribution knowledge to help him calculate the expected value in different situations. Recall from the lesson that the expected value of a binomial distribution can be calculated by `n x p`.

1. Calculate the expected number of sales out of the 3 he works on that Amir will win each week if he maintains his 30% win rate.
2. Calculate the expected number of sales out of the 3 he works on that he'll win if his win rate drops to 25%.
3. Calculate the expected number of sales out of the 3 he works on that he'll win if his win rate rises to 35%.

```
# Calculate the expected number of sales out of the 3 he works on that Amir will win each week if
won_30pct = 3 * 0.3
print(f"Number of sales out of 3 works at 30% win rate: {won_30pct}")

# Calculate the expected number of sales out of the 3 he works on that he'll win if his win rate
won_25pct = 3 * 0.25
```

```
print(f"Number of sales out of 3 works at 25% win rate: {won_25pct}")

# Calculate the expected number of sales out of the 3 he works on that he'll win if his win rate
won_35pct = 3 * 0.35
print(f"Number of sales out of 3 works at 35% win rate: {won_35pct}")
```

```
Number of sales out of 3 works at 30% win rate: 0.8999999999999999
Number of sales out of 3 works at 25% win rate: 0.75
Number of sales out of 3 works at 35% win rate: 1.0499999999999998
```

Key Points:

You learned about the binomial distribution, a fundamental concept in probability that models events with two possible outcomes, such as flipping a coin. Key points included:

- Understanding binary outcomes, which can be success/failure, win/loss, or heads/tails, and how these can be represented numerically (1 or 0).
- Using the `binom.rvs` function from `scipy.stats` to simulate random variables following a binomial distribution. This function requires specifying the number of trials (`n`), the probability of success (`p`), and the size, which determines how many times the experiment is run.
- The difference between simulating a single trial multiple times and multiple trials in one go was illustrated with coin flips.
- Adjusting the probability of success (`p`) to model biased outcomes, like a weighted coin, and observing how it affects the results.
- Calculating probabilities with the binomial distribution using binom.pmf for the probability of a specific number of successes, and binom.cdf for the probability of up to a certain number of successes.
- The expected value of a binomial distribution, which is the average number of successes over many trials, can be calculated with `n * p`.
- For example, to calculate the expected number of sales Amir will win each week with different win rates, you used the formula for the expected value in a binomial distribution:

```
# Expected number won with 30% win rate

won_30pct = 3 * 0.3
print(won_30pct)


# Expected number won with 25% win rate
```

```
won_25pct = 3 * 0.25
print(won_25pct)

# Expected number won with 35% win rate
won_35pct = 3 * 0.35
print(won_35pct)
```

- This lesson emphasized the importance of understanding and applying the binomial distribution to model real-world scenarios with binary outcomes, enhancing your ability to analyze and predict the probability of events.

Chapter 2.5: The normal distribution

The next probability distribution we'll discuss is the 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.

What is the normal distribution?

The normal distribution looks like this. Its shape is commonly referred to as a "bell curve". The normal distribution has a few important properties.

Symmetrical

First, it's symmetrical, so the left side is a mirror image of the right.

Area = 1

Second, just like any continuous distribution, the area beneath the curve is 1.

Curve never hits 0

Second, the probability never hits 0, even if it looks like it does at the tail ends. Only 0.006% of its area is contained beyond the edges of this graph.

Described by mean and standard deviation

The normal distribution is described by its mean and standard deviation. Here is a normal distribution with a mean of 20 and standard deviation of 3, and here is a normal distribution with a mean of 0 and a standard deviation of 1. When a normal distribution has mean 0 and a standard deviation of 1, it's a special distribution called the standard normal distribution.

Areas under the normal distribution

For the normal distribution, 68% of the area is within 1 standard deviation of the mean. 95% of the area falls within 2 standard deviations of the mean, and 99.7% of the area falls within three standard deviations. This is sometimes called the 68-95-99.7 rule.

Lots of histograms look normal

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.

Approximating data with the normal distribution

Since this height data closely resembles the normal distribution, we can take the area under a normal distribution with mean 161 and standard deviation 7 to approximate what percent of women fall into different height ranges.

What percent of women are shorter than 154 cm?

For example, what percent of women are shorter than 154 centimeters? We can answer this using `norm.cdf` from `scipy.stats`, which takes the area of the normal distribution less than some number. We pass in the number of interest, 154, followed by the mean and standard deviation of the normal distribution we're using. This gives us about 16% of women are shorter than 154 centimeters.

What percent of women are taller than 154 cm?

To find the percent of women taller than 154 centimeters, we can take 1 minus the area on the left of 154, which equals the area to the right of 154.

What percent of women are 154-157 cm?

To get the percent of women between 154 and 157 centimeters tall we can take the area below 157 and subtract the area below 154, which leaves us the area between 154 and 157.

What height are 90% of women shorter than?

We can also calculate percentages from heights using `norm.ppf`. To figure out what height 90% of women are shorter than, we pass 0.9 into `norm.ppf` along with the same mean and standard deviation we've been working with. This tells us that 90% of women are shorter than 170 centimeters tall.

What height are 90% of women taller than?

We can figure out the height 90% of women are taller than, since this is also the height that 10% of women are shorter than. We can take 1 minus 0.9 to get 0.1, which we'll use as the first argument of `norm.ppf`.

Generating random numbers

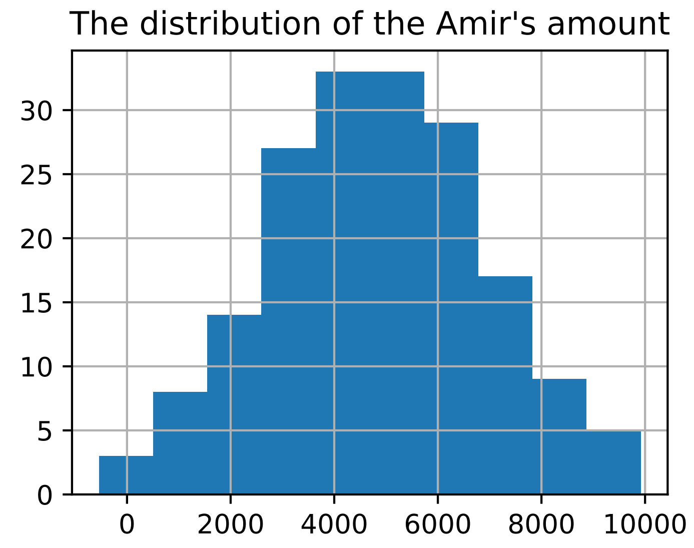
Just like with other distributions, we can generate random numbers from a normal distribution using `norm.rvs`, passing in the distribution's mean and standard deviation, as well as the sample size we want.

Exercise 2.5.1

1. Create a histogram with 10 bins to visualize the distribution of the amount. Show the plot.
2. What's the probability of Amir closing a deal worth less than $7500?
3. What's the probability of Amir closing a deal worth more than $1000?
4. What's the probability of Amir closing a deal worth between $3000 and $7000?
5. What amount will 25% of Amir's sales be less than?

```python
from scipy.stats import norm

# Create a histogram with 10 bins to visualize the distribution of the amount. Show the plot.
amir_deals['amount'].hist( bins=10)
plt.title("The distribution of the Amir's amount")
plt.show()

# What's the probability of Amir closing a deal worth less than $7500?
prob_less_7500 = norm.cdf(7500, 5000, 2000)
print(f"The probability of Amir closing a deal worth less than $7500 is: {prob_less_7500}")
```

```
# What's the probability of Amir closing a deal worth more than $1000?
prob_over_1000 = 1 - norm.cdf(1000, 5000, 2000)
print(f"The probability of Amir closing a deal worth more than $1000 IS: {prob_over_1000}")

# What's the probability of Amir closing a deal worth between $3000 and $7000?
prob_3000_to_7000 = norm.cdf(7000, 5000, 2000) - norm.cdf(3000, 5000, 2000)
print(f"The probability of Amir closing a deal worth between $3000 and $7000 is: {prob_3000_to_70

# What amount will 25% of Amir's sales be less than?
pct_25 = norm.ppf(0.25, 5000, 2000)
print(f"The amount of 25% of Amir's sales be will less than: {pct_25}")
```



```
The probability of Amir closing a deal worth less than $7500 is: 0.8943502263331446
The probability of Amir closing a deal worth more than $1000 IS: 0.9772498680518208
The probability of Amir closing a deal worth between $3000 and $7000 is: 0.6826894921370859
The amount of 25% of Amir's sales be will less than: 3651.0204996078364
```

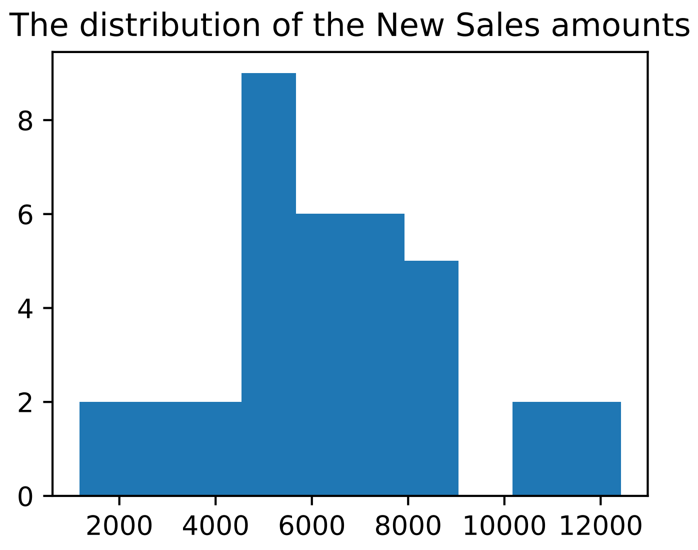
Exercise 2.5.2: Simulating sales under new market conditions

The company's financial analyst is predicting that next quarter, the worth of each sale will increase by 20% and the volatility, or standard deviation, of each sale's worth will increase by 30%. To see what Amir's sales might look like next quarter under these new market conditions, you'll simulate new sales amounts using the normal distribution and store these in the `new_sales` DataFrame, which has already been created for you.

1. Currently, Amir's average sale amount is $5000. Calculate what his new average amount will be if it increases by 20% and store this in `new_mean`.

2. Amir's current standard deviation is $2000. Calculate what his new standard deviation will be if it increases by 30% and store this in `new_sd`.
3. Create a variable called new_sales, which contains 36 simulated amounts from a normal distribution with a mean of `new_mean` and a standard deviation of `new_sd`.
4. Plot the distribution of the `new_sales` amounts using a histogram and show the plot.

```
# Currently, Amir's average sale amount is $5000. Calculate what his new average amount will be i
new_mean = (0.2 * 5000) + 5000

# Amir's current standard deviation is $2000. Calculate what his new standard deviation will be i
new_sd = (0.3 * 2000) + 2000

# Create a variable called new_sales, which contains 36 simulated amounts from a normal distribut
new_sales = norm.rvs(new_mean, new_sd, size = 36)

# Plot the distribution of the new_sales amounts using a histogram and show the plot.
plt.hist(new_sales)
plt.title("The distribution of the New Sales amounts")
plt.show()
```



Chapter 3: The central limit theorem

Now that you're familiar with the normal distribution, it's time to learn about what makes it so important.

Rolling the dice 5 times

Let's go back to our dice rolling example. We have a Series of the numbers 1 to 6 called die. To simulate rolling the die 5 times, we'll call `die.sample`. We pass in the Series we want to sample from, the size of the sample, and set `replace` to `True`. This gives us the results of 5 rolls. Now, we'll take the mean of the 5 rolls, which gives us 2. If we roll another 5 times and take the mean, we get a different mean. If we do it again, we get another mean.

Rolling the dice 5 times 10 times

Let's repeat this 10 times: we'll roll 5 times and take the mean. To do this, we'll use a `for loop`. We start by creating an empty list called `sample_means` to hold our means. We loop from 0 to 9 so that the process is repeated 10 times. Inside the loop, we roll 5 times and append the sample's mean to the `sample_means` list. This gives us a list of 10 different sample means. Let's plot these sample means.

Sampling distributions

A distribution of a summary statistic like this is called a sampling distribution. This distribution, specifically, is a sampling distribution of the sample mean.

100 sample means

Now let's do this 100 times. If we look at the new sampling distribution, its shape somewhat resembles the normal distribution, even though the distribution of the die is uniform.

1000 sample means

Let's take 1000 means. This sampling distribution more closely resembles the normal distribution.

Central limit theorem (CLT)

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.

Standard deviation and the CLT

The central limit theorem, or CLT, applies to other summary statistics as well. If we take the standard deviation of 5 rolls 1000 times, the sample standard deviations are distributed normally, centered around 1.9, which is the distribution's standard deviation.

Proportions and the CLT

Another statistic that the CLT applies to is proportion. Let's sample from the sales team 10 times with replacement and see how many draws have Claire as the outcome. In this case, 10% of draws were Claire. If we draw again, there are 40% Claires.

Sampling distribution of proportion

If we repeat this 1000 times and plot the distribution of the sample proportions, it resembles a normal distribution centered around 0.25, since Claire's name was on 25% of the tickets.

Mean of sampling distribution

Since these sampling distributions are normal, we can take their mean to get an estimate of a distribution's mean, standard deviation, or proportion. If we take the mean of our sample means from earlier, we get 3.48. That's pretty close to the expected value, which is 3.5! Similarly, the mean of the sample proportions of Claires isn't far off from 0.25. In these examples, we know what the underlying distributions look like, but if we don't, this can be a useful method for estimating characteristics of an underlying distribution. The central limit theorem also comes in handy when you have a huge population and don't have the time or resources to collect data on everyone. Instead, you can collect several smaller samples and create a sampling distribution to estimate what the mean or standard deviation is.

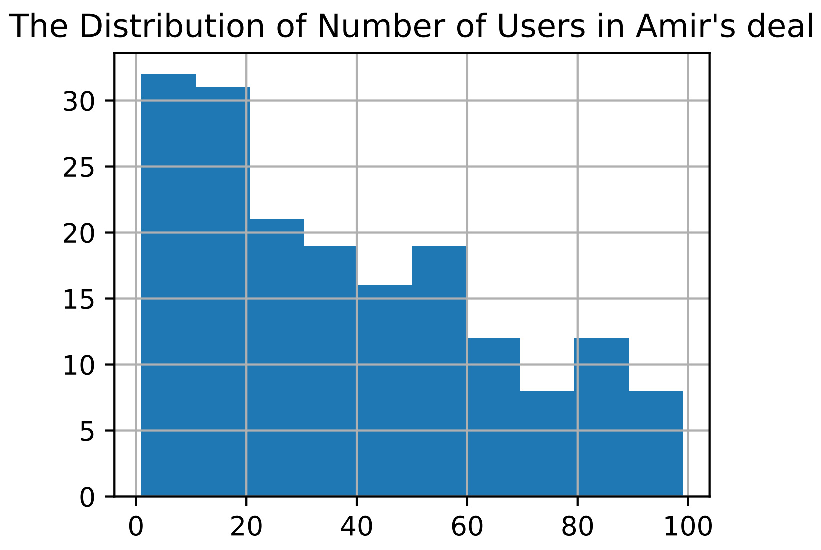
Exercise 3.1

1. Create a histogram of the `num_users` column of `amir_deals` and show the plot.
2. Set the seed to 104.
3. Take a sample of size 20 with replacement from the `num_users` column of `amir_deals`, and take the mean. Store the mean in `samp_20`.
4. Take mean of `samp_20`.
5. 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.
6. Convert `sample_means` into a `pd.Series`, create a histogram of the `sample_means`, and show the plot.
7. Take 30 samples (with replacement) of size 20 from `all_deals['num_users']` and take the mean of each sample. Store the sample means in `sample_means_1`.

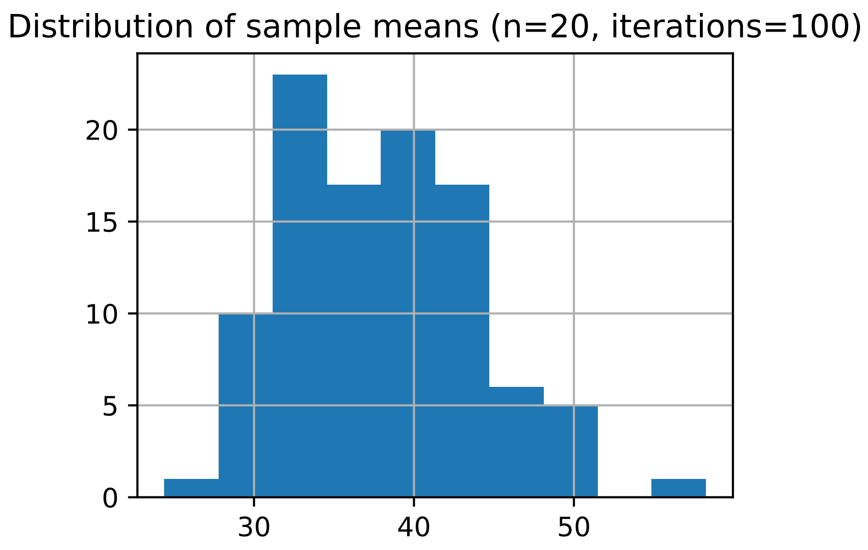
8. Print mean of `sample_means_1`.
9. Print the mean of the `num_users` column of `amir_deals`.

```
# Create a histogram of the num_users column of amir_deals and show the plot.
amir_deals['num_users'].hist()
plt.title("The Distribution of Number of Users in Amir's deal")
plt.show()

# Set the seed to 104.
np.random.seed(104)

# Take a sample of size 20 with replacement from the num_users column of amir_deals, and take the
samp_20 = amir_deals['num_users'].sample(20, replace=True)

# Take mean of samp_20
print(f"The Mean of the 20 samples from Number of Users is: {np.mean(samp_20)}")

# Repeat this 100 times using a for loop and store as sample_means. This will take 100 different
# Set seed to 104
np.random.seed(104)

# Sample 20 num_users with replacement from amir_deals and take mean
samp_20 = amir_deals['num_users'].sample(20, replace=True)
np.mean(samp_20)

sample_means = []
# Loop 100 times
for i in range(100):
  # Take sample of 20 num_users
  samp_20 = amir_deals['num_users'].sample(20, replace=True)
  # Calculate mean of samp_20
  samp_20_mean = np.mean(samp_20)
  # Append samp_20_mean to sample_means
  sample_means.append(samp_20_mean)

print(f"Distribution of sample means (n=20, iterations=100): {sample_means}")

# Convert sample_means into a pd.Series, create a histogram of the sample_means, and show the plo
sample_means_series = pd.Series(sample_means)
sample_means_series.hist()
plt.title("Distribution of sample means (n=20, iterations=100)")
# Show plot
plt.show()

# Set the random seed to 321.
```

```
np.random.seed(321)

# Take 30 samples (with replacement) of size 20 from all_deals['num_users'] and take the mean of
# sample_means_1 = []
# Loop 30 times to take 30 means
# for i in range(30):
  # Take sample of size 20 from num_users col of all_deals with replacement
  # cur_sample = all_deals['num_users'].sample(20, replace = True)

# Take mean of cur_sample
#  cur_mean = np.mean(cur_sample)
  # Append cur_mean to sample_means
#  sample_means_1.append(cur_mean)

# Print mean of sample_means_1
# print(np.mean(sample_means_1))

# Print the mean of the num_users column of amir_deals.
print(f"Amir's average number of users: {amir_deals['num_users'].mean()}")
```



```
The Mean of the 20 samples from Number of Users is: 32.0
Distribution of sample means (n=20, iterations=100): [np.float64(31.35), np.float64(45.05), np.fl
```



```
Amir's average number of users: 37.651685393258425
```

```
Expected output:
Overall average number of users: 38.31333333333332
Amir's average number of users: 37.651685393258425
```

Conclusion:

We can see that Amir's average number of users is very close to the overall average, so it looks like he's meeting expectations. Make sure to note this in his performance review!

Chapter 4: The Poisson distribution

In this lesson, we'll talk about another probability distribution called the Poisson distribution.

Poisson processes

Before we talk about probability, let's define a Poisson process. A Poisson process is a process where events appear to happen at a certain rate, but completely at random. For example, the number of animals adopted from an animal shelter each week is a Poisson process - we may know that on average there are 8 adoptions per week, but this number can differ randomly. Other examples would be the number of people arriving at a restaurant each hour, or the number of earthquakes per year in California. The time unit like, hours, weeks, or years, is irrelevant as long as it's consistent.

Poisson distribution

The Poisson distribution describes the probability of some number of events happening over a fixed period of time. We can use the Poisson distribution to calculate the probability of at least 5 animals getting adopted in a week, the probability of 12 people arriving in a restaurant in an hour, or the probability of fewer than 20 earthquakes in California in a year.

Lambda (λ)

The Poisson distribution is described by a value called lambda, which represents the average number of events per time period. In the animal shelter example, this would be the average number of adoptions per week, which is 8. This value is also the expected value of the distribution! The Poisson distribution with λ equals 8 looks like this. Notice that it's a discrete distribution since we're counting events, and 7 and 8 are the most likely number of adoptions to happen in a week.

Lambda is the distribution's peak

Lambda changes the shape of the distribution, so a Poisson distribution with λ equals 1, in blue, looks quite different than a Poisson distribution with lambda equals 8, in green, but no matter what, the distribution's peak is always at its lambda value.

Probability of a single value

Given that the average number of adoptions per week is 8, what's the probability of 5 adoptions in a week? Just like the other probability distributions, we can import `poisson` from `scipy.stats`. We'll use the `poisson.pmf` function, passing 5 as the first argument and 8 as the second argument to indicate the distribution's mean. This gives us about 9%.

Probability of less than or equal to

To get the probability that 5 or fewer adoptions will happen in a week, use the `poisson.cdf` function, passing in the same numbers. This gives us about 20%.

Probability of greater than

Just like other probability functions you've learned about so far, take 1 minus the "less than or equal to 5" probability to get the probability of more than 5 adoptions. There's an 81% chance that more than 5 adoptions will occur. If the average number of adoptions rises to 10 per week, there will be a 93% chance that more than 5 adoptions will occur.

Sampling from a Poisson distribution

Just like other distributions, we can take samples from Poisson distributions using poisson-dot-rvs. Here, we'll simulate 10 different weeks at the animal shelter. In one week, there are 14 adoptions, but only 6 in another.

The CLT still applies!

Just like other distributions, the sampling distribution of sample means of a Poisson distribution looks normal with a large number of samples.

Exercise 4.1: Tracking lead responses

Your company uses sales software to keep track of new sales leads. It organizes them into a queue so that anyone can follow up on one when they have a bit of free time. Since the number of lead responses is a countable outcome over a period of time, this scenario corresponds to a Poisson distribution. On average, Amir responds to 4 leads each day. In this exercise, you'll calculate probabilities of Amir responding to different numbers of leads.

1. Import poisson from `scipy.stats` and calculate the probability that Amir responds to 5 leads in a day, given that he responds to an average of 4.
2. Amir's coworker responds to an average of 5.5 leads per day. What is the probability that she answers 5 leads in a day?
3. What's the probability that Amir responds to 2 or fewer leads in a day?
4. What's the probability that Amir responds to more than 10 leads in a day?

```python
# Import poisson from scipy.stats and calculate the probability that Amir responds to 5 leads in
from scipy.stats import poisson
# Probability of 5 responses
prob_5 = poisson.pmf(5, 4)
print(f"The probability that Amir responds to 5 leads in a day: {prob_5}")
# 0.1562934518505317 (15.6%)

# Amir's coworker responds to an average of 5.5 leads per day. What is the probability that she a
prob_coworker = poisson.pmf(5, 5.5)
print(f"The probability Amir's coworker responds to an average of 5.5 leads per day: {prob_cowork
# 0.17140068409793663 (17.1%)

# What's the probability that Amir responds to 2 or fewer leads in a day?
prob_2_or_less = poisson.cdf(2, 4)
print(f"The probability that Amir responds to 2 or fewer leads in a day: {prob_2_or_less}")
# 0.23810330555354436 (23.8%)

# What's the probability that Amir responds to more than 10 leads in a day?
```

```
prob_over_10 = 1 - poisson.cdf(10, 4)
print(f"The probability that Amir responds to more than 10 leads in a day: {prob_over_10}")
# 0.0028397661205137315 (0.28397661%)
```

```
The probability that Amir responds to 5 leads in a day: 0.1562934518505317
The probability Amir's coworker responds to an average of 5.5 leads per day: 0.17140068409793663
The probability that Amir responds to 2 or fewer leads in a day: 0.23810330555354436
The probability that Amir responds to more than 10 leads in a day: 0.0028397661205137315
```

Chapter 4.1: More probability distributions

In this lesson, we'll discuss a few other probability distributions.

Exponential distribution

The first distribution is the exponential distribution, which represents the probability of a certain time passing between Poisson events. We can use the exponential distribution to predict, for example, the probability of more than 1 day between adoptions, the probability of fewer than 10 minutes between restaurant arrivals, and the probability of 6-8 months passing between earthquakes. Just like the Poisson distribution, the time unit doesn't matter as long as it's consistent. The exponential distribution uses the same λ value, which represents the rate, that the Poisson distribution does. Note that lambda and rate mean the same value in this context. It's also continuous, unlike the Poisson distribution, since it represents time.

Customer service requests

For example, let's say that one customer service ticket is created every 2 minutes. We can rephrase this so it's in terms of a time interval of one minute, so half of a ticket is created each minute. We'll use 0.5 as the λ value. The exponential distribution with a rate of one half looks like this.

Lambda in exponential distribution

The rate affects the shape of the distribution and how steeply it declines.

Expected value of exponential distribution

Recall that lambda is the expected value of the Poisson distribution, which measures frequency in terms of rate or number of events. In our customer service ticket example, this means that the expected number of requests per minute is 0.5. The exponential distribution measures frequency in terms of time between events. The expected value of the exponential distribution can be calculated by taking 1 divided by lambda. In our example, the expected time between requests is 1 over one half, which is 2, so there is an average of 2 minutes between requests.

How long until a new request is created?

Similar to other continuous distributions, we can use `expon.cdf` to calculate probabilities. The probability of waiting less than 1 minute for a new request is calculated using `expon.cdf`, passing in 1 followed by a 2, which gives us about an 40% chance. Note that we're passing in 2, not the lambda value which is 0.5. The probability of waiting more than 4 minutes can be found using 1 minus `expon.cdf` of 4, 2, giving a 13% chance. Finally, the probability of waiting between 1 and 4 minutes can be found by taking `expon.cdf` of 4 and subtracting `expon.cdf` of 1. There's a 50% chance you'll wait between 1 and 4 minutes.

(Student's) t-distribution

The next distribution is the t-distribution, which is also sometimes called Student's t-distribution. Its shape is similar to the normal distribution, but not quite the same. If we compare the normal distribution, in blue, with the t-distribution with one degree of freedom, in orange, the t-distribution's tails are thicker. This means that in a t-distribution, observations are more likely to fall further from the mean.

Degrees of freedom

The t-distribution has a parameter called degrees of freedom, which affects the thickness of the distribution's tails. Lower degrees of freedom results in thicker tails and a higher standard deviation. As the number of degrees of freedom increases, the distribution looks more and more like the normal distribution.

Log-normal distribution

The last distribution we'll discuss is the log-normal distribution. Variables that follow a log-normal distribution have a logarithm that is normally distributed. This results in distributions that are skewed, unlike the normal distribution. There are lots of real-world examples that follow this distribution, such as the length of chess games, blood pressure in adults, and the number of hospitalizations in the 2003 SARS outbreak.

Exercise 4.2: Modeling time between leads

To further evaluate Amir's performance, you want to know how much time it takes him to respond to a lead after he opens it. On average, he responds to 1 request every 2.5 hours. In this exercise, you'll calculate probabilities of different amounts of time passing between Amir receiving a lead and sending a response.

1. Import `expon` from `scipy.stats`. What's the probability it takes Amir less than an hour to respond to a lead?
2. What's the probability it takes Amir more than 4 hours to respond to a lead?

3. What's the probability it takes Amir 3-4 hours to respond to a lead?

```
# Import expon from scipy.stats. What's the probability it takes Amir less than an hour to respon
from scipy.stats import expon
# Print probability response takes < 1 hour
print(f"The probability it takes Amir less than an hour to respond to a lead {expon.cdf(1, scale=
# 0.3296799539643607 (32.97%)

# What's the probability it takes Amir more than 4 hours to respond to a lead?
print(f"The probability it takes Amir more than 4 hours to respond to a lead: {1 - expon.cdf(4, s
# 0.20189651799465536 (20.2%)

# What's the probability it takes Amir 3-4 hours to respond to a lead?
print(f"The probability it takes Amir 3-4 hours to respond to a lead: {expon.cdf(4, scale = 2.5)
# 0.09929769391754684 (9.93%)
```

```
The probability it takes Amir less than an hour to respond to a lead 0.3296799539643607
The probability it takes Amir more than 4 hours to respond to a lead: 0.20189651799465536
The probability it takes Amir 3-4 hours to respond to a lead: 0.09929769391754684
```

Chapter 5: Correlation

Welcome to the final chapter of the course, where we'll talk about correlation and experimental design.

Relationships between two variables

Before we dive in, let's talk about relationships between numeric variables. We can visualize these kinds of relationships with scatter plots - in this scatterplot, we can see the relationship between the total amount of sleep mammals get and the amount of REM sleep they get. The variable on the x-axis is called the explanatory or independent variable, and the variable on the y-axis is called the response or dependent variable.

Correlation coefficient

We can also examine relationships between two numeric variables using a number called the correlation coefficient. This is a number between -1 and 1, where the magnitude corresponds to the strength of the relationship between the variables, and the sign, positive or negative, corresponds to the direction of the relationship.

Magnitude = strength of relationship

Here's a scatterplot of 2 variables, x and y, that have a correlation coefficient of 0.99. Since the data points are closely clustered around a line, we can describe this as a **near-perfect or very strong relationship**. If we know what x is, we'll have a pretty good idea of what the value of y could be. Here, `x` and `y` have a correlation coefficient of 0.75, and the data points are a bit more spread out. In this plot, x and y have a correlation of 0.56 and are therefore **moderately correlated**. A correlation coefficient around 0.2 would be considered a **weak relationship**. When the correlation coefficient is close to 0, x and y have no relationship and the scatterplot looks completely random. This means that knowing the value of x doesn't tell us anything about the value of y.

Sign = direction

The sign of the correlation coefficient corresponds to the direction of the relationship. A positive correlation coefficient indicates that as x increases, y also increases. A negative correlation coefficient indicates that as x increases, y decreases.

Visualizing relationships

To visualize relationships between two variables, we can use a scatterplot. We'll use `seaborn`, which is a plotting package built on top of `matplotlib`. We import `seaborn` as `sns`, which is the alias commonly used for `seaborn`. We create a scatterplot using `sns.scatterplot`, passing it the name of the variable for the x-axis, the name of the variable for the y-axis, as well as the `msleep` DataFrame to the data argument. Finally, we call `plt.show`.

Adding a trendline

We can add a linear trendline to the scatterplot using seaborn's `lmplot()` function. It takes the same arguments as `sns.scatterplot`, but we'll set `ci` to `None` so that there aren't any confidence interval margins around the line. Trendlines like this can be helpful to more easily see a relationship between two variables.

Computing correlation

To calculate the correlation coefficient between two Series, we can use the `.corr` method. If we want the correlation between the `sleep_total` and `sleep_rem` columns of `msleep`, we can take the `sleep_total` column and call `.corr` on it, passing in the other Series we're interested in. Note that it doesn't matter which Series the method is invoked on and which is passed in since the correlation between x and y is the same thing as the correlation between y and x.

Many ways to calculate correlation

There's more than one way to calculate correlation, but the method we've been using in this video is called the Pearson product-moment correlation, which is also written as `r`. This is the most commonly used measure of correlation. Mathematically, It's calculated using this formula, where $\bar{x}$ and $\bar{y}$ are the means of x and y, and σ_x and σ_y are the standard deviations of x and y. The formula itself isn't important to memorize, but know that there are variations of this formula that measure correlation a bit differently, such as *Kendall's tau* (τ) and *Spearman's rho* (ρ), but those are beyond the scope of this course.

Exercise 5.1: Relationships between variables

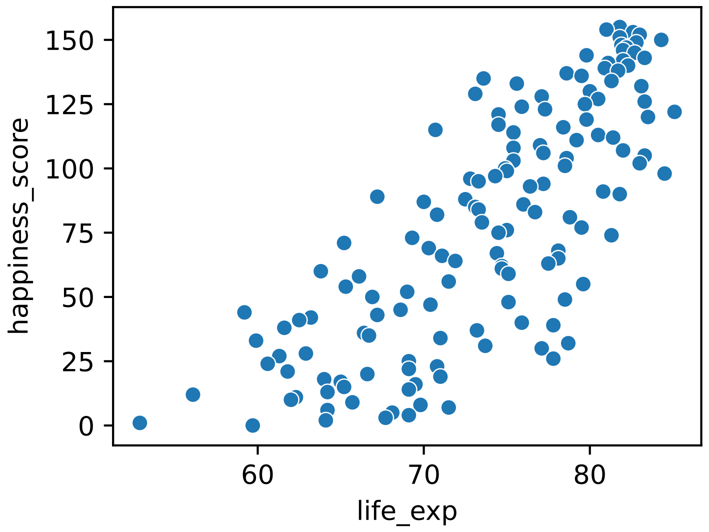
In this chapter, you'll be working with a dataset `world_happiness` containing results from the 2019 World Happiness Report. The report scores various countries based on how happy people in that country are. It also ranks each country on various societal aspects such as social support, freedom, corruption,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

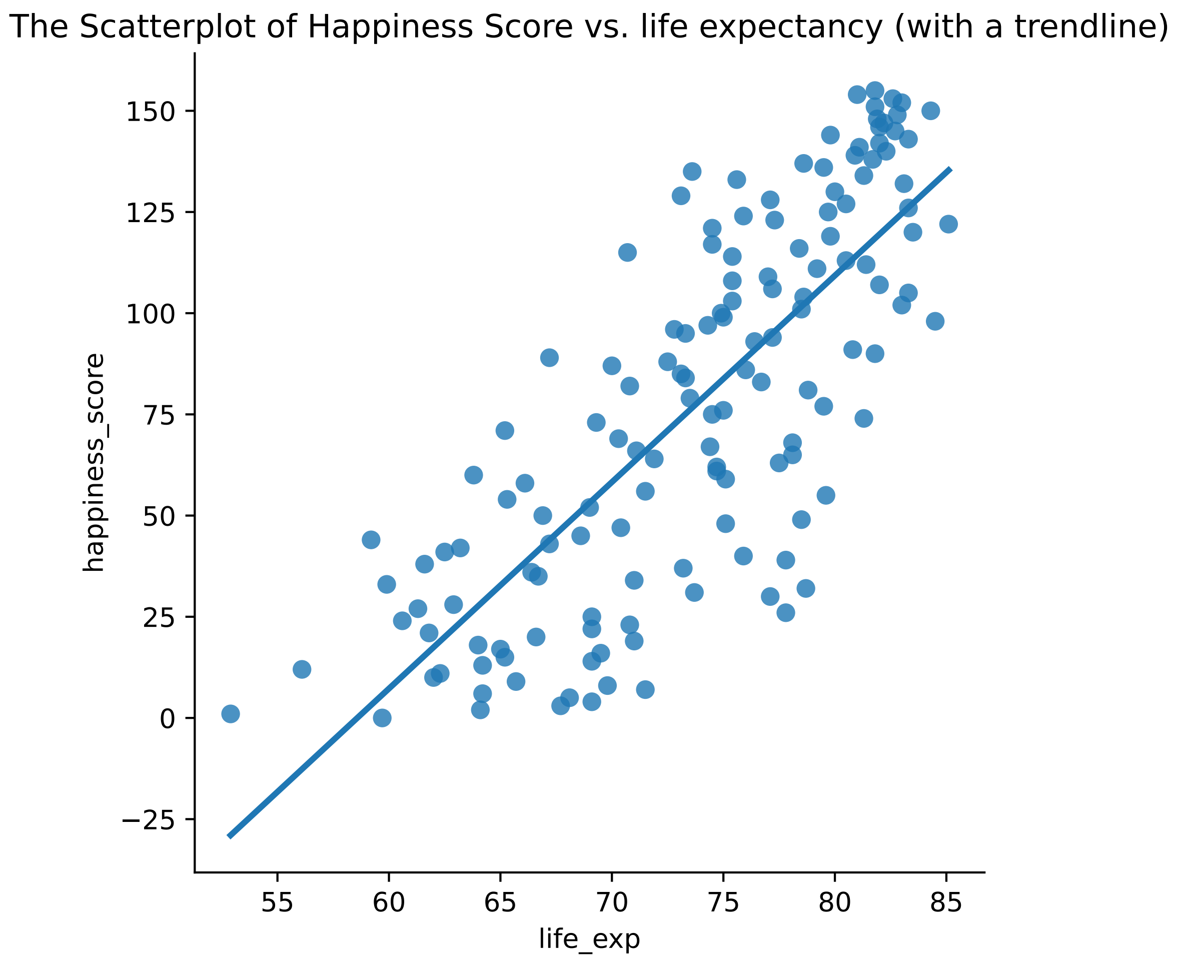
1. Create a scatterplot of `happiness_score` vs. `life_exp` (without a trendline) using `seaborn`.
2. Create a scatterplot of `happiness_score` vs. `life_exp` with a linear trendline using `seaborn`, setting `ci` to None.
3. Based on the scatterplot, which is most likely the correlation between `life_exp` and `happiness_score` ?

```python
# Import the seaborn package
import seaborn as sns

# Import dataset
world_happiness = pd.read_csv("datasets/world_happiness.csv")

# Create a scatterplot of happiness_score vs. life_exp (without a trendline) using seaborn.
sns.scatterplot(x='life_exp', y='happiness_score', data=world_happiness)
plt.title("The Scatterplot of Happiness Score vs. life expectancy (without a trendline)")
# Show plot
plt.show()

# Create a scatterplot of happiness_score vs. life_exp with a linear trendline using seaborn, set
sns.lmplot(x='life_exp', y='happiness_score', data=world_happiness, ci=None)
plt.title("The Scatterplot of Happiness Score vs. life expectancy (with a trendline)")
# Show plot
plt.show()

# Based on the scatterplot, which is most likely the correlation between life_exp and happiness_s
corr_happy_life = world_happiness['happiness_score'].corr(world_happiness['life_exp'])
```

```
print("The correlation between life_exp and happiness_score: {corr_happy_life}")
# 0.7802249053272062
```

The Scatterplot of Happiness Score vs. life expectancy (without a trendline)





```
The correlation between life_exp and happiness_score: {corr_happy_life}
```

Chapter 6: Correlation caveats

While correlation is a useful way to quantify relationships, there are some caveats.

Non-linear relationships

Consider a data. There is clearly a relationship between x and y, but when we calculate the correlation, we get 0.18.

Non-linear relationships

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.

Correlation only accounts for linear relationships

Just like any summary statistic, correlation shouldn't be used blindly, and you should always visualize your data when possible.

Mammal sleep data

Let's return to the mammal sleep data.

Body weight vs. awake time

Here's a scatterplot of each mammal's body weight versus the time they spend awake each day. The relationship between these variables is definitely not a linear one. The correlation between body weight and awake time is only about 0.3, which is a weak linear relationship.

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.

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.

Other transformations

In addition to the log transformation, there are lots of other transformations that can be used to make a relationship more linear, like taking the square root or reciprocal of a variable. The choice of transformation will depend on the data and how skewed it is. These can be applied in different combinations to x and y, for example, you could apply a log transformation to both x and y, or a square root transformation to x and a reciprocal transformation to y.

Why use a transformation?

So why use a transformation? Certain statistical methods rely on variables having a linear relationship, like calculating a correlation coefficient. Linear regression is another statistical technique that requires variables to be related in a linear manner, which you can learn all about in this course.

Correlation does not imply causation

Let's talk about one more important caveat of correlation that you may have heard about before: correlation does not imply causation. This means that if x and y are correlated, x doesn't necessarily cause y. For example, here's a scatterplot of the per capita margarine consumption in the US each year and the divorce rate in the state of Maine. The correlation between these two variables is 0.99, which is nearly perfect. However, this doesn't mean that consuming more margarine will cause more divorces. This kind of correlation is often called a spurious correlation.

Confounding

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. However, there is a third, hidden variable at play, which is smoking. Smoking is known to be associated with coffee consumption. 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. Another example of this is the relationship between holidays and retail sales. While it might be that people buy more around holidays as a way of celebrating, it's hard to tell how much of the increased sales is due to holidays, and how much is due to the special deals and promotions that often run around holidays. Here, special deals confound the relationship between holidays and sales.

Note

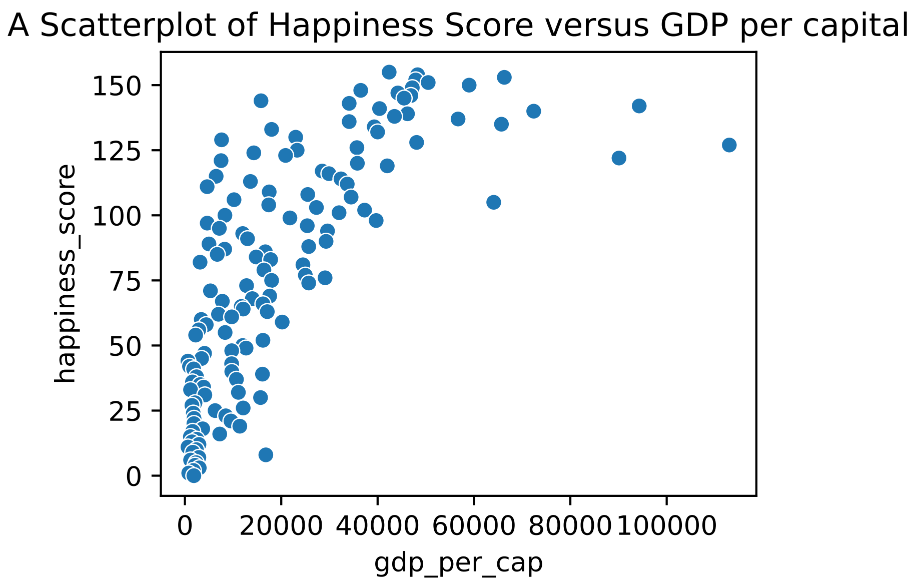
The correlation coefficient can't account for any relationships that aren't linear, regardless of strength.

Exercise 5.1

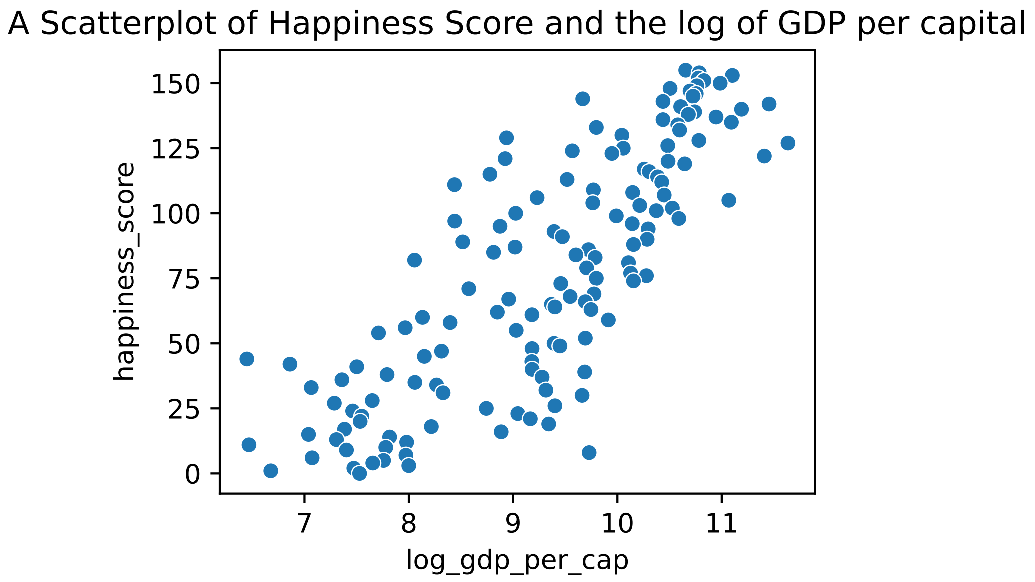
1. Create a scatterplot of `happiness_score` versus `gdp_per_cap` and calculate the correlation between them.
2. Add a new column to `world_happiness` called `log_gdp_per_cap` that contains the `log of gdp_per_cap`.
3. Create a seaborn scatterplot of `happiness_score` versus `log_gdp_per_cap`.

4. Calculate the correlation between `log_gdp_per_cap` and `happiness_score`.

```
# Create a scatterplot of happiness_score versus gdp_per_cap and calculate the correlation betwee
sns.scatterplot(x='gdp_per_cap', y='happiness_score', data=world_happiness)
plt.title("A Scatterplot of Happiness Score versus GDP per capital")
plt.show()
# Calculate correlation
cor = world_happiness['happiness_score'].corr(world_happiness['gdp_per_cap'])
print(f"The correlation between happiness_score and gdp_per_cap is: {cor}")

# Add a new column to world_happiness called log_gdp_per_cap that contains the log of gdp_per_cap
world_happiness['log_gdp_per_cap'] = np.log(world_happiness['gdp_per_cap'])

# Create a seaborn scatterplot of happiness_score versus log_gdp_per_cap.
sns.scatterplot(x='log_gdp_per_cap', y='happiness_score', data=world_happiness)
plt.title("A Scatterplot of Happiness Score and the log of GDP per capital")
plt.show()

# Calculate the correlation between log_gdp_per_cap and happiness_score.
cor_1 = world_happiness['happiness_score'].corr(world_happiness['log_gdp_per_cap'])
print(f"The correlation between happiness_score and log of gdp_per_cap is: {cor_1}")
```



```
The correlation between happiness_score and gdp_per_cap is: 0.7279733012222976
```



```
The correlation between happiness_score and log of gdp_per_cap is: 0.804314600491829
```

Chapter 6: Design of experiments

Often, data is created as a result of a study that aims to answer a specific question. However, data needs to be analyzed and interpreted differently depending on how the data was generated and how the study was designed.

Vocabulary

Experiments generally aim to answer a question in the form, "What is the effect of the treatment on the response?" In this setting, treatment refers to the explanatory or independent variable, and response refers to the response or dependent variable. For example, what is the effect of an advertisement on the number of products purchased? In this case, the treatment is an advertisement, and the response is the number of products purchased.

Controlled experiments

In a controlled experiment, participants are randomly assigned to either the treatment group or the control group, where the treatment group receives the treatment and the control group does not. A great example of this is an A/B test. In our example, the treatment group will see an advertisement, and the control group will not. Other than this difference, the groups should be comparable so that we can determine if seeing an advertisement causes people to buy more. If the groups aren't comparable, this could lead to confounding, or bias. If the average age of participants in the treatment group is 25 and the average age of participants in the control group is 50, age could be a potential confounder if younger people are more likely to purchase more, and this will make the experiment biased towards the treatment.

The gold standard of experiments will use...

The gold standard, or ideal experiment, will eliminate as much bias as possible by using certain tools. The first tool to help eliminate bias in controlled experiments is to use a randomized controlled trial. In a randomized controlled trial, participants are randomly assigned to the treatment or control group and their assignment isn't based on anything other than chance. Random assignment like this helps ensure that the groups are comparable. The second way is to use a placebo, which is something that resembles the treatment, but has no effect. This way, participants don't know if they're in the treatment or control group. This ensures that the effect of the treatment is due to the treatment itself, not the idea of getting the treatment. This is common in clinical trials that test the effectiveness of a drug. The control group will still be given a pill, but it's a sugar pill that has minimal effects on the response. In a double-blind experiment, the person administering the treatment or running the experiment also doesn't know whether they're administering the actual treatment or the placebo. This protects against bias in the response as well as the analysis of the results. These different tools all boil down to the same principle: if there are fewer opportunities for bias to creep into your experiment, the more reliably you can conclude whether the treatment affects the response.

Observational studies

The other kind of study we'll discuss is the **observational study**. In an observational study, participants are not randomly assigned to groups. Instead, participants assign themselves, usually based on pre-existing characteristics. This is useful for answering questions that aren't conducive to a controlled experiment. If you want to study the effect of smoking on cancer, you can't force people to start smoking. Similarly, if you want to study how past purchasing behavior affects whether someone will buy a product, you can't force people to have certain past purchasing behavior. Because assignment isn't random, there's no way to guarantee that the groups will be comparable in every aspect, so observational studies can't establish causation, only association. The effects of the treatment may be confounded by factors that got certain people into the control group and certain people into the treatment group. However, there are ways to control for confounders, which can help strengthen the reliability of conclusions about association.

Longitudinal vs. cross-sectional studies

The final important distinction to make is between **longitudinal and cross-sectional studies**. In a **longitudinal study**, the same participants are followed over a period of time to examine the effect of treatment on the response. In a **cross-sectional study**, data is collected from a single snapshot in time. If you wanted to investigate the effect of age on height, a **cross-sectional study** would measure the heights of people of different ages and compare them. However, the results will be confounded by birth year and lifestyle since it's possible that each generation is getting taller.

In a longitudinal study,the same people would have their heights recorded at different points in their lives, so the confounding is eliminated. It's important to note that **longitudinal studies** are more expensive, and take longer to perform, while **cross-sectional studies** are cheaper, faster, and more convenient.