COURSE 19 | SAMPLING AND POINT IN PYTHON

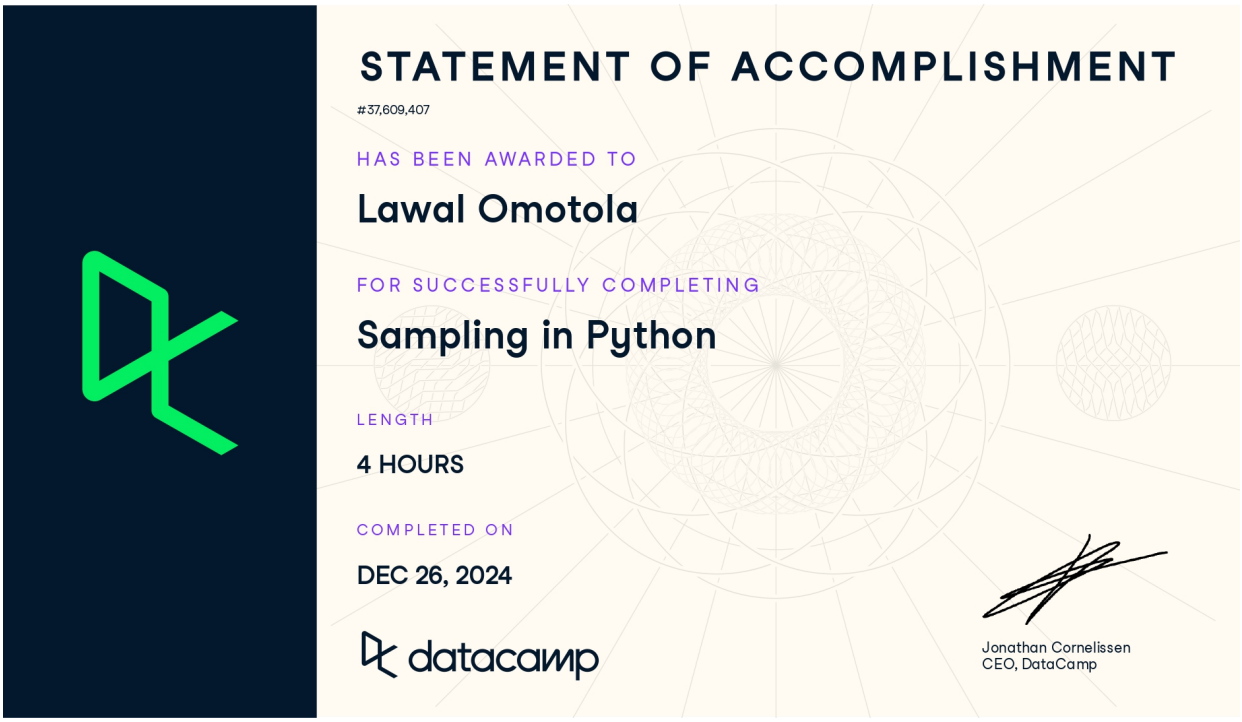
Lawal's Note

2024-12-26

Table of contents

0.1 Chapter 1: Introduction to Sampling 2
0.1.1 Chapter 1.1: Sampling and point estimates 3
0.1.2 Exercise 1.1.1 4
0.1.3 Exercise 1.1.2 7
0.1.4 Chapter 1.2: Convenience sampling 8
0.1.5 Exercise 1.2.1 10
0.1.6 Question 11
0.1.7 Exercise 1.2.2 12
0.1.8 Question 13
0.1.9 Chapter 1.3: Pseudo-random number generation 14
0.1.10 Exercise 1.3.1 15
0.1.11 Exercise 1.3.2 17
0.2 CHAPTER 2: Sampling Methods 18
0.2.1 Chapter 2.1: Simple random and systematic sampling 19
0.2.2 Exercise 2.1.1 20
0.2.3 Exercise 2.1.2 23
0.2.4 Exercise 2.1.3 25
0.2.5 Chapter 2.2: Stratified and weighted random sampling 28
0.2.6 Exercise 2.2.2 32
0.2.7 Exercise 2.2.3 36
0.2.8 Chapter 2.3: Cluster sampling 39
0.2.9 Exercise 2.3.1 40
0.2.10 Chapter 2.4: Comparing sampling methods 49
0.2.11 Exercise 2.4.1 51
0.2.12 Exercise 2.4.4 54
0.3 CHAPTER 3: Sampling Distributions 56
0.3.1 Chapter 3.1: Relative error of point estimates 56
0.3.2 Exercise 3.1.1 57

0.3.3 Chapter 3.2: Creating a sampling distribution . . . 58
0.3.4 Exercise 3.2.1 . . . 59
0.3.5 Chapter 3.3: Approximate sampling distributions . . . 61
0.3.6 Exercise 3.3.1 . . . 62
0.3.7 Exercise 3.3.2 . . . 64
0.3.8 Chapter 3.4: Standard errors and the Central Limit Theorem . . . 66
0.3.9 Exercise 3.4.1 . . . 67
0.3.10 Exercise 3.4.2 . . . 69
0.4 CHAPTER 4: Bootstrap Distributions . . . 71
0.4.1 Chapter 4.1: Introduction to bootstrapping . . . 71
0.4.2 Exercise 4.1.1 . . . 73
0.4.3 Chapter 4.2: Comparing sampling and bootstrap distributions . . . 75
0.4.4 Exercise 4.2.1 . . . 77
0.4.5 Exercise 4.2.2 . . . 78
0.4.6 Exercise 4.2.3 . . . 80
0.4.7 Chapter 4.3: Confidence intervals . . . 81
0.4.8 Exercise 4.3.1 . . . 83
0.5 Reference . . . 85



0.1 Chapter 1: Introduction to Sampling

Learn what sampling is and why it is so powerful. You'll also learn about the problems caused by convenience sampling and the differences between true randomness and pseudo-randomness.

0.1.1 Chapter 1.1: Sampling and point estimates

Hi! Welcome to the course! I'm James, and I'll be your host as we delve into the world of sampling data with Python. To start, let's look at what sampling is and why it might be useful.

Estimating the population of France

Let's consider the problem of counting how many people live in France. The standard approach is to take a census. This means contacting every household and asking how many people live there. There are lots of people in France. Since there are millions of people in France, this is a really expensive process. Even with modern data collection technology, most countries will only conduct a census every five or ten years due to the cost.

Sampling households

In 1786, Pierre-Simon Laplace realized you could estimate the population with less effort. Rather than asking every household who lived there, he asked a small number of households and used statistics to estimate the number of people in the whole population. This technique of working with a subset of the whole population is called sampling.

Population vs. sample

Two definitions are important for this course. The population is the complete set of data that we are interested in. The previous example involved the literal population of France, but in statistics, it doesn't have to refer to people. One thing to bear in mind is that there is usually no equivalent of the census, so typically, we won't know what the whole population is like - more on this in a moment. The sample is the subset of data that we are working with.

Coffee rating dataset

Picture a dataset of professional ratings of coffees. Each row corresponds to one coffee, and there are thirteen hundred and thirty-eight rows in the dataset. The coffee is given a score from zero to one hundred, which is stored in the total_cup_points column. Other columns contain contextual information like the variety and country of origin and scores between zero and ten for attributes of the coffee such as aroma and body. These scores are averaged across all the reviewers for that particular coffee. It doesn't contain every coffee in the world, so we don't know exactly what the population of coffees is. However, there are enough here that we can think of it as our population of interest.

Points vs. flavor: population

Let's consider the relationship between cup points and flavor by selecting those two columns. This dataset contains all thirteen hundred and thirty-eight rows from the original dataset.

Points vs. flavor: 10 row sample

The pandas `.sample` method returns a random subset of rows. Setting n to ten means ten random rows are returned. By default, rows from the original dataset can't appear in the sample dataset multiple times, so we are guaranteed to have ten unique rows in our sample.

Python sampling for Series

The `.sample` method also works on pandas Series. Here, using square-bracket subsetting retrieves the `total_cup_points` column as a Series, and the `n` argument specifies how many random values to return.

Population parameters & point estimates

A population parameter is a calculation made on the population dataset. We aren't limited to counting values either; here, we calculate the mean of the cup points using NumPy. By contrast, a point estimate, or sample statistic, is a calculation based on the sample dataset. Here, the mean of the total cup points is calculated on the sample. Notice that the means are very similar but not identical.

Point estimates with pandas

Working with pandas can be easier than working with NumPy. These mean calculations can be performed using the `.mean` pandas method.

0.1.2 Exercise 1.1.1

Simple sampling with pandas

Throughout this chapter, you'll be exploring song data from Spotify. Each row of this population dataset represents a song, and there are over 40,000 rows. Columns include the song name, the artists who performed it, the release year, and attributes of the song like its duration, tempo, and danceability. You'll start by looking at the durations.

Your first task is to sample the Spotify dataset and compare the mean duration of the population with the sample.

Instructions

1. Sample 1000 rows from `spotify`, assigning to `spotify_sample`.
2. Calculate the mean duration in minutes from `spotify` using pandas.
3. Calculate the mean duration in minutes from `spotify_sample` using pandas.

```python
# Importing pandas
import pandas as pd

# Importing the course arrays
spotify = pd.read_feather("datasets/spotify_2000_2020.feather")


# Sample 1000 rows from spotify_population
spotify_sample = spotify.sample(n=1000)

# Print the sample
print(spotify_sample)

# Calculate the mean duration in mins from spotify_population
mean_dur_pop = spotify['duration_minutes'].mean()

# Calculate the mean duration in mins from spotify_sample
mean_dur_samp = spotify_sample['duration_minutes'].mean()

# Print the means
print(mean_dur_pop)
print(mean_dur_samp)
```

```
       acousticness                                     artists  \
21088      0.108000                          ['Keith Urban']
37017      0.686000                        ['Crecer German']
24030      0.030100                          ['Keith Urban']
6412       0.795000                     ['Rufus Wainwright']
26227      0.017000                               ['Lloyd']
...             ...                                     ...
633        0.362000                         ['Joshua Radin']
41418      0.000479                       ['Imagine Dragons']
17777      0.273000  ['Josef "J7" Lord', 'Christopher H. Knight']
2599       0.021800                     ["Lil' Flip", 'Lea']
23197      0.027800                        ['Kenny Chesney']

       danceability  duration_ms  duration_minutes  energy  explicit  \
21088         0.624     323040.0          5.384000   0.840       0.0
37017         0.785     202573.0          3.376217   0.381       0.0
24030         0.657     273587.0          4.559783   0.785       0.0
6412          0.460     284467.0          4.741117   0.378       0.0
26227         0.527     240107.0          4.001783   0.796       1.0
...             ...          ...               ...     ...       ...
633           0.545     149013.0          2.483550   0.255       0.0
41418         0.634     210933.0          3.515550   0.662       0.0
```

```
17777         0.751     142581.0          2.376350   0.805       1.0
2599          0.845     225187.0          3.753117   0.346       0.0
23197         0.684     235733.0          3.928883   0.840       0.0

                          id  instrumentalness  key  liveness  loudness  \
21088  0b9djfiuDIMw1zKH6gV74g          0.000546  4.0    0.1440    -5.768
37017  0VYFakqQG8p6yHrqq9TAoY          0.000000  6.0    0.1070    -5.991
24030  3PY88239tYBnAv5LQoU2oY          0.000005  3.0    0.0927    -8.428
6412   586EonpqbFo5GRPRy8IKqf          0.000000  4.0    0.0905    -8.950
26227  0npGoOENjn7vVvIMmvWekQ          0.000000  1.0    0.0798    -5.111
...                       ...               ...  ...       ...       ...
633    2ZaYFNn1YQuLSVdHhanr4Q          0.000018  8.0    0.0841   -14.844
41418  2bzitsPcImYC6DZWvvLCQi          0.001420  6.0    0.1110    -7.543
17777  1apH42Wa9c0Iy2X7VZOZr9          0.000000  9.0    0.0766    -7.444
2599   3FaUH7ZMjW1hv9Jx6MIAIf          0.000000  0.0    0.1350    -9.381
23197  3tHPjLBakLS48aumhFpJMt          0.000000  9.0    0.3290    -5.753

       mode                                                 name  popularity  \
21088   1.0                                 Somebody Like You        67.0
37017   0.0                                 Quién Te Entiende        64.0
24030   1.0                                   Long Hot Summer        59.0
6412    1.0                     Cigarettes And Chocolate Milk        40.0
26227   0.0                                      Lay It Down        57.0
...     ...                                              ...         ...
633     1.0                                        Only You        55.0
41418   1.0  Zero - From the Original Motion Picture "Ralph...        70.0
17777   0.0               Who Do You Voodoo (From Dead Island)        48.0
2599    1.0                                Sunshine (feat. Lea)        47.0
23197   1.0                                             Young        55.0

      release_date  speechiness    tempo  valence    year
21088   2002-01-01       0.0337  111.020    0.656  2002.0
37017   2016-10-07       0.0348  121.885    0.872  2016.0
24030   2010-01-01       0.0370  127.987    0.769  2010.0
6412          2001       0.0271   97.891    0.166  2001.0
26227         2011       0.0620  155.990    0.534  2011.0
...            ...          ...      ...      ...     ...
633           2006       0.0311   98.088    0.831  2006.0
41418   2018-11-09       0.0325   90.011    0.244  2018.0
17777   2011-09-13       0.3030   93.045    0.507  2011.0
2599    2004-03-30       0.1060   93.989    0.819  2004.0
23197   2002-04-02       0.0433  125.801    0.682  2002.0

[1000 rows x 20 columns]
3.8521519140900073
```

```
3.9347189666666664
```

> **Note**
>
> *Notice that the mean song duration in the sample is similar, but not identical to the mean song duration in the whole population.*

0.1.3 Exercise 1.1.2

Simple sampling and calculating with NumPy

You can also use numpy to calculate parameters or statistics from a list or pandas Series.

You'll be turning it up to eleven and looking at the loudness property of each song.

Instructions

1. Create a pandas Series, `loudness_pop`, by subsetting the `loudness` column from `spotify`.
   - Sample `loudness_pop` to get 100 random values, assigning to `loudness_samp`.
2. Calculate the mean of `loudness_pop` using `numpy`.
3. Calculate the mean of `loudness_samp` using `numpy`.

```
# Importing pandas
import pandas as pd
import numpy as np

# Importing the course arrays
spotify = pd.read_feather("datasets/spotify_2000_2020.feather")

# Create a pandas Series from the loudness column of spotify_population
loudness_pop = spotify['loudness']

# Sample 100 values of loudness_pop
loudness_samp = loudness_pop.sample(n=100)

print(loudness_samp)

# Calculate the mean of loudness_pop
mean_loudness_pop = np.mean(loudness_pop)

# Calculate the mean of loudness_samp
mean_loudness_samp = np.mean(loudness_samp)
```

```
print(mean_loudness_pop)
print(mean_loudness_samp)
```

```
19881    -3.844
16465    -4.749
13177    -6.521
8441     -9.341
22506    -4.081
          ...
4917    -12.201
15219    -5.029
9224     -4.501
30503    -4.289
9181     -4.631
Name: loudness, Length: 100, dtype: float64
-7.366856851353947
-7.067409999999999
```

> **Note**
>
> *Again, notice that the calculated value (the mean) is close but not identical in each case.*

0.1.4 Chapter 1.2: Convenience sampling

The point estimates you calculated in the previous exercises were very close to the population parameters that they were based on, but is this always the case?

The Literary Digest election prediction

In 1936, a newspaper called The Literary Digest ran an extensive poll to try to predict the next US presidential election. They phoned ten million voters and had over two million responses. About one-point-three million people said they would vote for Landon, and just under one million people said they would vote for Roosevelt. That is, Landon was predicted to get fifty-seven percent of the vote, and Roosevelt was predicted to get forty-three percent of the vote. Since the sample size was so large, it was presumed that this poll would be very accurate. However, in the election, Roosevelt won by a landslide with sixty-two percent of the vote. So what went wrong? Well, in 1936, telephones were a luxury, so the only people who had been contacted by The Literary Digest were relatively rich. The sample of voters was not representative of the whole population of voters, and so the poll suffered from sample bias. The data was collected by the easiest method, in this case, telephoning people. This is called convenience sampling and is often prone to sample bias. Before sampling, we need to think about our data collection process to avoid biased results.

Finding the mean age of French people

Let's look at another example. While on vacation at Disneyland Paris, you start wondering about the mean age of French people. To get an answer, you ask ten people stood nearby about their ages. Their mean age is twenty-four-point-six years old. Do you think this will be a good estimate of the mean age of all French citizens?

How accurate was the survey?

On the left, you can see mean ages taken from the French census. Notice that the population has been gradually getting older as birth rates decrease and life expectancy increases. In 2015, the mean age was over forty, so our estimate of twenty-four-point-six is way off. The problem is that the family-friendly fun at Disneyland means that the sample ages weren't representative of the general population. There are generally more eight-year-olds than eighty-year-olds riding rollercoasters.

Convenience sampling coffee ratings

Let's return to the coffee ratings dataset and look at the mean cup points population parameter. The mean is about eighty-two. One form of convenience sampling would be to take the first ten rows, rather than the random rows we saw in the previous video. We can take the first 10 rows with the pandas `head` method. The mean cup points from this sample is higher at eighty-nine. The discrepancy suggests that coffees with higher cup points appear near the start of the dataset. Again, the convenience sample isn't representative of the whole population.

Visualizing selection bias

Histograms are a great way to visualize the selection bias. We can create a histogram of the total cup points from the population, which contains values ranging from around 59 to around 91. The `np.arange` function can be used to create bins of width 2 from 59 to 91. Recall that the stop value in `np.arange` is exclusive, so we specify 93, not 91. Here's the same code to generate a histogram for the convenience sample.

Distribution of a population and of a convenience sample

Comparing the two histograms, it is clear that the distribution of the sample is not the same as the population: all of the sample values are on the right-hand side of the plot.

Visualizing selection bias for a random sample

This time, we'll compare the `total_cup_points` distribution of the population with a random sample of 10 coffees.

Distribution of a population and of a simple random sample

Notice how the shape of the distributions is more closely aligned when random sampling is used.

0.1.5 Exercise 1.2.1

Are findings from the sample generalizable?

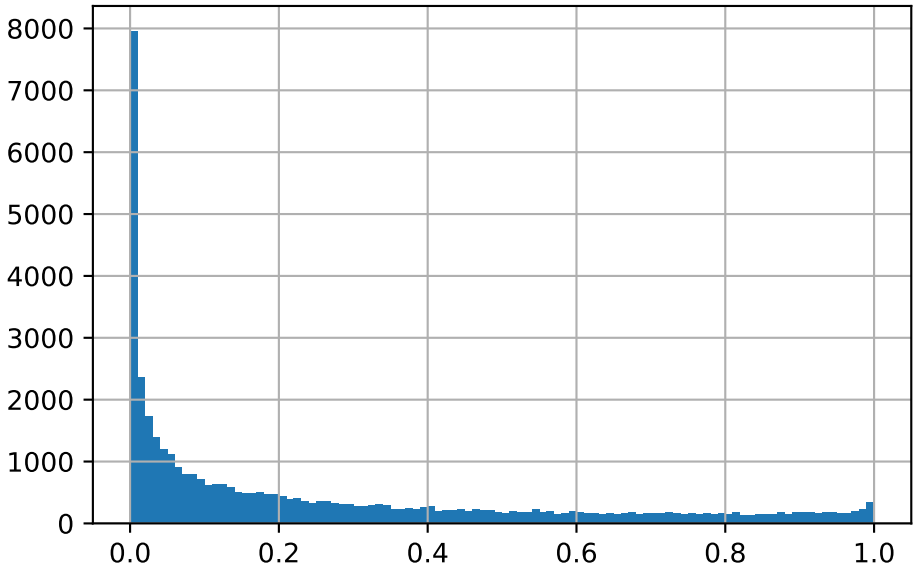
You just saw how convenience sampling—collecting data using the easiest method—can result in samples that aren't representative of the population. Equivalently, this means findings from the sample are not generalizable to the population. Visualizing the distributions of the population and the sample can help determine whether or not the sample is representative of the population.

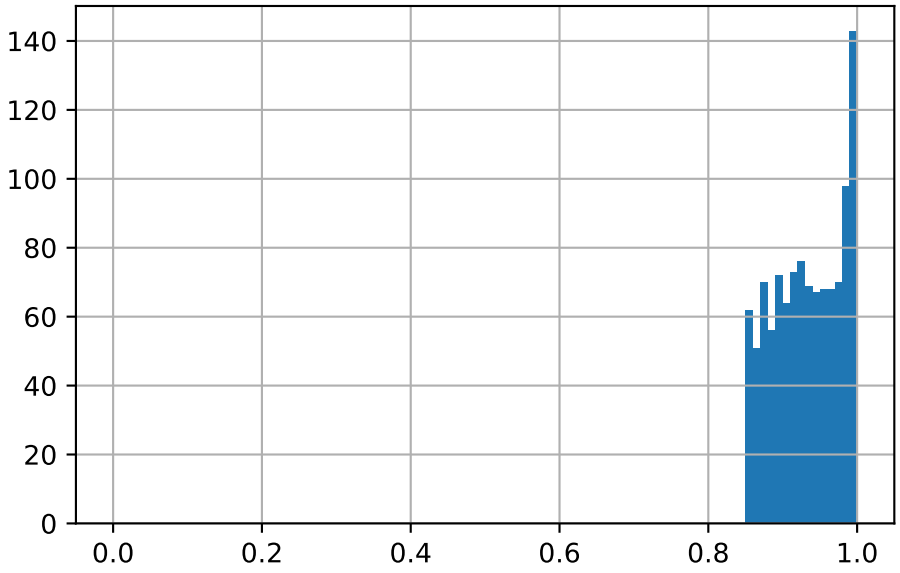
The Spotify dataset contains an `acousticness` column, which is a confidence measure from zero to one of whether the track was made with instruments that aren't plugged in. You'll compare the `acousticness` distribution of the total population of songs with a sample of those songs.

Instructions

1. Plot a histogram of the `acousticness` from `spotify` with bins of width `0.01` from `0` to `1` using pandas `.hist()`.
2. Update the histogram code to use the `spotify_mysterious_sample` dataset.

```
# Importing pandas
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Importing the course arrays
spotify = pd.read_feather("datasets/spotify_2000_2020.feather")

# Visualize the distribution of acousticness with a histogram
spotify['acousticness'].hist(bins=np.arange(0,1.01,0.01))
plt.show()

# Generate a convenience sample where acousticness is consistently higher
spotify_high_acousticness = spotify[(spotify['acousticness'] >= 0.85) & (spotify['acousticness']

# Sample 1107 entries from the high acousticness subset
spotify_mysterious_sample = spotify_high_acousticness.sample(n=1107)

# Update the histogram to use spotify_mysterious_sample
spotify_mysterious_sample['acousticness'].hist(bins=np.arange(0, 1.01, 0.01))
plt.show()
```





0.1.6 Question

Compare the two histograms you drew. Are the `acousticness` values in the sample generalizable to the general population?

No. The acousticness samples are consistently higher than those in the general population.

The acousticness values in the sample are all greater than 0.85, whereas they range from 0 to 1 in the whole population.

0.1.7 Exercise 1.2.2

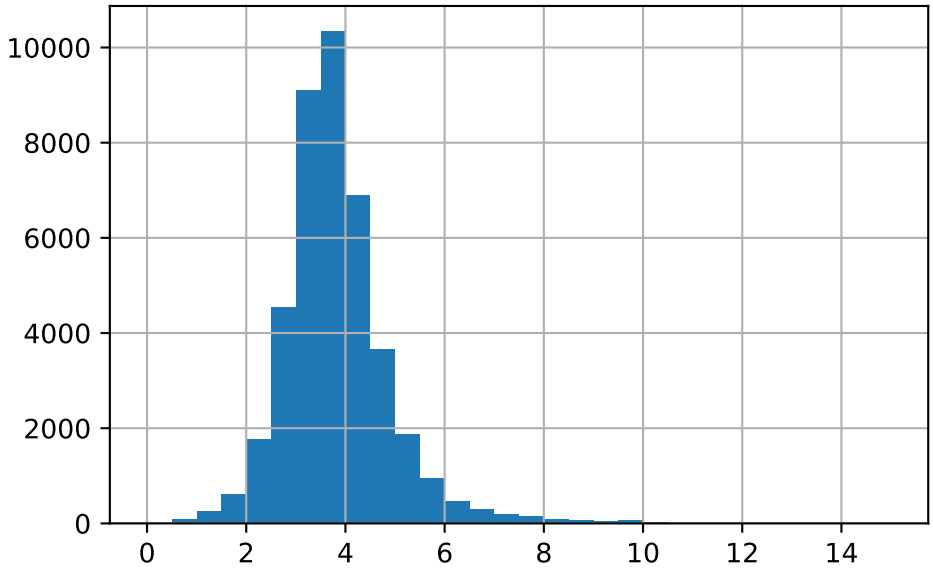
Are these findings generalizable?

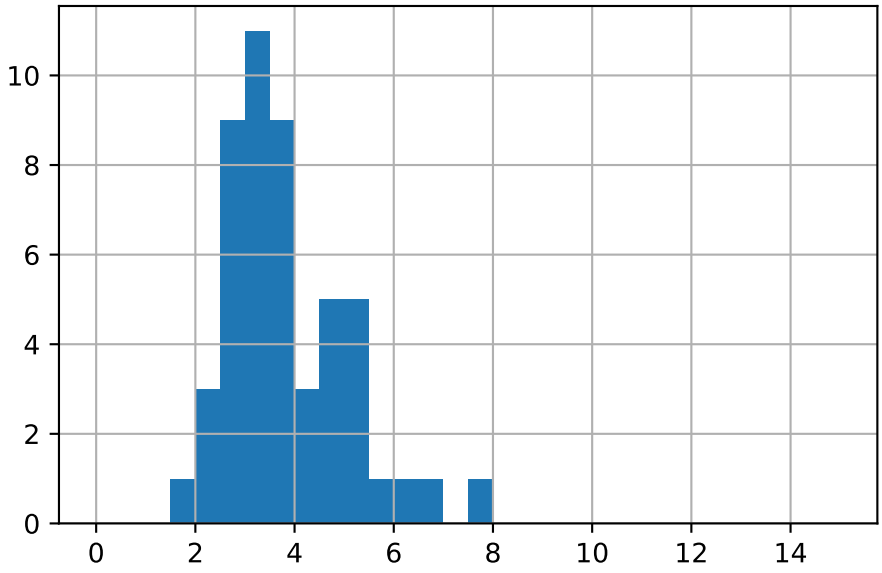
Let's look at another sample to see if it is representative of the population. This time, you'll look at the `duration_minutes` column of the `Spotify` dataset, which contains the length of the song in minutes.

Instructions

- Plot a histogram of `duration_minutes` from `spotify` with bins of width `0.5` from `0` to `15` using pandas `.hist()`.
- Update the histogram code to use the `spotify_mysterious_sample2` dataset.

```
# Importing pandas
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Importing the course arrays
spotify = pd.read_feather("datasets/spotify_2000_2020.feather")

# Generate a convenience sample where duration_minutes is within the specified range
spotify_duration_range = spotify[(spotify['duration_minutes'] >= 0.8079999999) & (spotify['durati

# Sample 50 entries from the spotify_mysterious_sample2 dataset
spotify_mysterious_sample2 = spotify_duration_range.sample(n=50)

# Visualize the distribution of duration_minutes in the population with a histogram
spotify['duration_minutes'].hist(bins=np.arange(0,15.5,0.5))
plt.show()

# Visualize the distribution of duration_minutes as a histogram
spotify_mysterious_sample2['duration_minutes'].hist(bins=np.arange(0, 15.5, 0.5))
plt.show()
```





0.1.8 Question

Compare the two histograms you drew. Are the duration values in the sample generalizable to the general population?

0.1.8.1 Answer

Yes. The sample selected is likely a random sample of all songs in the population.

The duration values in the sample show a similar distribution to those in the whole population, so the results are generalizable.

0.1.9 Chapter 1.3: Pseudo-random number generation

You previously saw how to use a random sample to get results similar to those in the population. But how does a computer actually do this random sampling?

What does random mean?

There are several meanings of random in English. This definition from Oxford Languages is the most interesting for us. If we want to choose data points at random from a population, we shouldn't be able to predict which data points would be selected ahead of time in some systematic way.

True random numbers

To generate truly random numbers, we typically have to use a physical process like flipping coins or rolling dice. The Hotbits service generates numbers from radioactive decay, and RANDOM.ORG generates numbers from atmospheric noise, which are radio signals generated by lightning. Unfortunately, these processes are fairly slow and expensive for generating random numbers.

https://www.fourmilab.ch/hotbits

https://www.random.org

Pseudo-random number generation

For most use cases, pseudo-random number generation is better since it is cheap and fast. Pseudo-random means that although each value appears to be random, it is actually calculated from the previous random number. Since you have to start the calculations somewhere, the first random number is calculated from what is known as a seed value. The word random is in quotes to emphasize that this process isn't really random. If we start from a particular seed value, all future numbers will be the same.

Pseudo-random number generation example

For example, suppose we have a function to generate pseudo-random values called `calc_next_random`. To begin, we pick a seed number, in this case, one. `calc_next_random` does some calculations and returns three. We then feed three into `calc_next_random`, and it does the same set of calculations and returns two. And if we can keep feeding in the last number, it will return something apparently random. Although the process is deterministic, the trick to a random number generator is to make it look like the values are random.

Random number generating functions

NumPy has many functions for generating random numbers from statistical distributions. To use each of these, make sure to prepend each function name with `numpy.random` or `np.random`. Some of them, like `.uniform` and `.normal`, may be familiar. Others have more niche applications.

Visualizing random numbers

Let's generate some pseudo-random numbers. The first arguments to each random number function specify distribution parameters. The size argument specifies how many numbers to generate, in this case, five thousand. We've chosen the beta distribution, and its parameters are named a and b. These random numbers come from a continuous distribution, so a great way to visualize them is with a histogram. Here, because the numbers were generated from the beta distribution, all the values are between zero and one.

Random numbers seeds

To set a random seed with NumPy, we use the `.random.seed` method. `Random.seed` takes an integer for the seed number, which can be any number you like. `.normal` generates pseudo-random numbers from the normal distribution. The loc and scale arguments set the mean and standard deviation of the distribution, and the size argument determines how many random numbers from that distribution will be returned. If we call `.normal` a second time, we get two different random numbers. If we reset the seed by calling `random.seed` with the same seed number, then call `.normal` again, we get the same numbers as before. This makes our code reproducible.

Using a different seed

Now let's try a different seed. This time, calling `.normal` generates different numbers.

0.1.10 Exercise 1.3.1

Generating random numbers

You've used `.sample()` to generate pseudo-random numbers from a set of values in a DataFrame. A related task is to generate random numbers that follow a statistical distribution, like the uniform distribution or the normal distribution.

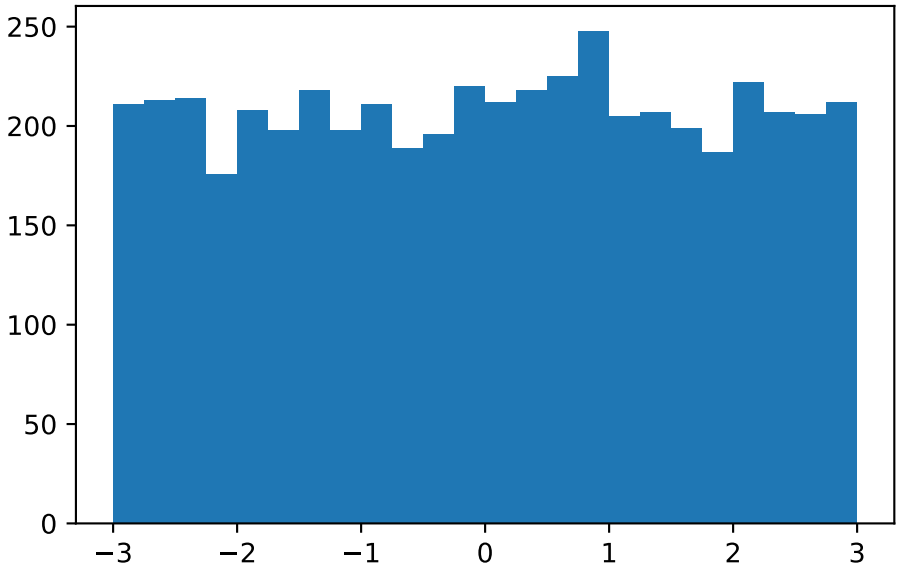
Each random number generation function has distribution-specific arguments and an argument for specifying the number of random numbers to generate.

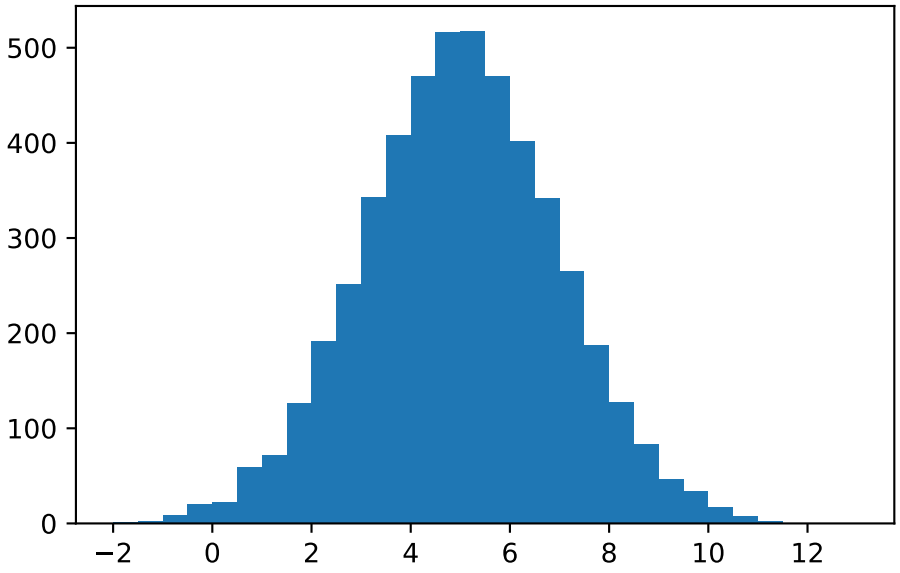
Instructions

1. Generate 5000 numbers from a uniform distribution, setting the parameters `low` to `-3` and `high` to `3`.
2. Generate 5000 numbers from a normal distribution, setting the parameters `loc` to `5` and `scale` to `2`.
3. Plot a histogram of `uniforms` with `bins` of width of `0.25` from `-3` to `3` using `plt.hist()`.
4. Plot a histogram of `normals` with `bins` of width of `0.5` from `-2` to `13` using `plt.hist()`.

```
# Importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Generate random numbers from a Uniform(-3, 3)
uniforms = np.random.uniform(low=-3, high=3, size=5000)

# Print uniforms
print(uniforms)

# Generate random numbers from a Normal(5, 2)
normals = np.random.normal(loc=5, scale = 2, size= 5000)

# Print normals
print(normals)

# Plot a histogram of uniform values, binwidth 0.25
plt.hist(uniforms, bins=np.arange(-3,3.25,0.25))
plt.show()

# Plot a histogram of normal values, binwidth 0.5
plt.hist(normals, bins = np.arange(-2, 13.5, 0.5))
plt.show()
```

```
[ 0.92847775  1.1912387  -1.39016428 ... -0.27583228  1.09736343
 -0.73869707]
[6.43786331 8.54350627 6.63613505 ... 5.70630282 4.97970263 4.26095338]
```





0.1.11 Exercise 1.3.2

Understanding random seeds

While random numbers are important for many analyses, they create a problem: the results you get can vary slightly. This can cause awkward conversations with your boss when your script for calculating the sales forecast gives different answers each time.

Setting the seed for `numpy`'s random number generator helps avoid such problems by making the random number generation reproducible.

Question 1

Which statement about x and y is true?

```
import numpy as np
np.random.seed(123)
x = np.random.normal(size=5)
y = np.random.normal(size=5)
```

The values of x are different from those of y

Question 2

Which statement about x and y is true?

```
import numpy as np
np.random.seed(123)
x = np.random.normal(size=5)
np.random.seed(123)
y = np.random.normal(size=5)
```

x and y have identical values.

Question 3

Which statement about x and y is true?

```
import numpy as np
np.random.seed(123)
x = np.random.normal(size=5)
np.random.seed(456)
y = np.random.normal(size=5)
```

The values of x are different from those of y.

0.2 CHAPTER 2: Sampling Methods

It's time to get hands-on and perform the four random sampling methods in Python: simple, systematic, stratified, and cluster.

0.2.1 Chapter 2.1: Simple random and systematic sampling

There are several methods of sampling from a population. In this video, we'll look at simple random sampling and systematic random sampling.

Simple random sampling

Simple random sampling works like a raffle or lottery. We start with our population of raffle tickets or lottery balls and randomly pick them out one at a time.

Simple random sampling of coffees

In our coffee ratings dataset, instead of raffle tickets or lottery balls, the population consists of coffee varieties. To perform simple random sampling, we take some at random, one at a time. Each coffee has the same chance as any other of being picked. When using this technique, sometimes we might end up with two coffees that were next to each other in the dataset, and sometimes we might end up with large areas of the dataset that were not selected from at all.

Simple random sampling with pandas

We've already seen how to do simple random sampling with pandas. We call `.sample` and set `n` to the size of the sample. We can also set the seed using the random_state argument to generate reproducible results, just like we did pseudo-random number generation. Previously, by not setting `random_state` when sampling, our code would generate a different random sample each time it was run.

Systematic sampling

Another sampling method is known as systematic sampling. This samples the population at regular intervals. Here, looking from top to bottom and left to right within each row, every fifth coffee is sampled.

Systematic sampling - defining the interval

Systematic sampling with pandas is slightly trickier than simple random sampling. The tricky part is determining how big the interval between each row should be for a given sample size. Suppose we want a sample size of five coffees. The population size is the number of rows in the whole dataset, and in this case, it's one thousand three hundred and thirty-eight. The interval is the population size divided by the sample size, but because we want the answer to be an integer, we perform integer division with two forward slashes. This is like standard division but discards any fractional part. One-three-three-eight divided by five is actually two hundred and sixty-seven-point-six, and discarding the fractional part leaves two hundred and sixty-seven. Thus, to get a systematic sample of five coffees, we will select every two hundred sixty-seventh coffee in the dataset.

Systematic sampling - selecting the rows

To select every two hundred and sixty-seventh row, we call dot-iloc on coffee_ratings and pass double-colons and the interval, which is 267 in this case. Double-colon interval tells pandas to select every two hundred and sixty-seventh row from zero to the end of the DataFrame.

The trouble with systematic sampling

There is a problem with systematic sampling, though. Suppose we are interested in statistics about the aftertaste attribute of the coffees. To examine this, first, we use `reset_index` to create a column of index values in our DataFrame that we can plot. Plotting aftertaste against index shows a pattern. Earlier rows generally have higher aftertaste scores than later rows. This introduces bias into the statistics that we calculate. In general, it is only safe to use systematic sampling if a plot like this has no pattern; that is, it just looks like noise.

Making systematic sampling safe

To ensure that systematic sampling is safe, we can randomize the row order before sampling. dot-sample has an argument named frac that lets us specify the proportion of the dataset to return in the sample, rather than the absolute number of rows that n specifies. Setting frac to one randomly samples the whole dataset. In effect, this randomly shuffles the rows. Next, the indices need to be reset so that they go in order from zero again. Specifying drop equals True clears the previous row indexes, and chaining to another `reset_index` call creates a column containing these new indexes. Redrawing the plot with the shuffled dataset shows no pattern between aftertaste and index. This is great, but note that once we've shuffled the rows, systematic sampling is essentially the same as simple random sampling.

0.2.2 Exercise 2.1.1

Simple random sampling

The simplest method of sampling a population is the one you've seen already. It is known as *simple random sampling* (sometimes abbreviated to "SRS"), and involves picking rows at random, one at a time, where each row has the same chance of being picked as any other.

In this chapter, you'll apply sampling methods to a synthetic (fictional) employee attrition dataset from IBM, where "attrition" in this context means leaving the company.

Instructions

- Sample 70 rows from `attrition` using simple random sampling, setting the random seed to 18900217.
- Print the sample dataset, `attrition_samp`. What do you notice about the indices?

```
# Importing pandas
import pandas as pd

# Importing the course arrays
attrition = pd.read_feather("datasets/attrition.feather")

# Sample 70 rows using simple random sampling and set the seed
attrition_samp = attrition.sample(n=70, random_state=18900217)

# Print the sample
print(attrition_samp)
```

```
      Age  Attrition     BusinessTravel  DailyRate              Department  \
1134   35        0.0      Travel_Rarely        583  Research_Development
1150   52        0.0         Non-Travel        585                 Sales
531    33        0.0      Travel_Rarely        931  Research_Development
395    31        0.0      Travel_Rarely       1332  Research_Development
392    29        0.0      Travel_Rarely        942  Research_Development
...   ...        ...                ...        ...                   ...
361    27        0.0  Travel_Frequently       1410                 Sales
1180   36        0.0      Travel_Rarely        530                 Sales
230    26        0.0      Travel_Rarely       1443                 Sales
211    29        0.0  Travel_Frequently        410  Research_Development
890    30        0.0  Travel_Frequently       1312  Research_Development

      DistanceFromHome      Education    EducationField  \
1134                25         Master           Medical
1150                29         Master     Life_Sciences
531                 14       Bachelor           Medical
395                 11        College           Medical
392                 15  Below_College     Life_Sciences
...                ...            ...              ...
361                  3  Below_College           Medical
1180                 2         Master     Life_Sciences
230                 23       Bachelor         Marketing
211                  2  Below_College     Life_Sciences
890                  2         Master  Technical_Degree

     EnvironmentSatisfaction  Gender  ...  PerformanceRating  \
1134                    High  Female  ...          Excellent
1150                     Low    Male  ...          Excellent
531                Very_High  Female  ...          Excellent
395                     High    Male  ...          Excellent
392                   Medium  Female  ...          Excellent
...                      ...     ...  ...                ...
```

```
361                Very_High  Female  ...        Outstanding
1180                    High  Female  ...          Excellent
230                     High  Female  ...          Excellent
211                Very_High  Female  ...          Excellent
890                Very_High  Female  ...          Excellent

     RelationshipSatisfaction  StockOptionLevel TotalWorkingYears  \
1134                     High                 1                16
1150                   Medium                 2                16
531                 Very_High                 1                 8
395                 Very_High                 0                 6
392                       Low                 1                 6
...                       ...               ...               ...
361                    Medium                 2                 6
1180                     High                 0                17
230                      High                 1                 5
211                      High                 3                 4
890                 Very_High                 0                10

     TrainingTimesLastYear WorkLifeBalance  YearsAtCompany  \
1134                     3           Good              16
1150                     3           Good               9
531                      5         Better               8
395                      2           Good               6
392                      2           Good               5
...                    ...            ...             ...
361                      3         Better               6
1180                     2           Good              13
230                      2           Good               2
211                      3         Better               3
890                      2         Better               9

      YearsInCurrentRole  YearsSinceLastPromotion YearsWithCurrManager
1134                  10                       10                    1
1150                   8                        0                    0
531                    7                        1                    6
395                    5                        0                    1
392                    4                        1                    3
...                  ...                      ...                  ...
361                    5                        0                    4
1180                   7                        6                    7
230                    2                        0                    0
211                    2                        0                    2
890                    7                        0                    7
```

```
[70 rows x 31 columns]
```

0.2.3 Exercise 2.1.2

Systematic sampling

One sampling method that avoids randomness is called systematic sampling. Here, you pick rows from the population at regular intervals.

For example, if the population dataset had one thousand rows, and you wanted a sample size of five, you could pick rows 0, 200, 400, 600, and 800.

Instructions

1.Set the sample size to 70. - Calculate the population size from `attrition`. - Calculate the interval between the rows to be sampled.

2. Systematically sample `attrition` to get the rows of the population at each `interval`, starting at 0; assign the rows to `attrition_sys_samp`

```
# Importing pandas
import pandas as pd

# Importing the course arrays
attrition = pd.read_feather("datasets/attrition.feather")

# Set the sample size to 70
sample_size = 70

# Calculate the population size from attrition_pop
pop_size = len(attrition)

# Calculate the interval
interval = pop_size//sample_size

# Systematically sample 70 rows
attrition_sys_samp = attrition.iloc[::interval]

# Print the sample
print(attrition_sys_samp)
```

```
      Age  Attrition BusinessTravel  DailyRate            Department  \
0      21        0.0  Travel_Rarely        391  Research_Development
21     19        0.0  Travel_Rarely       1181  Research_Development
42     45        0.0  Travel_Rarely        252  Research_Development
```

```
63     23        0.0  Travel_Rarely        373  Research_Development
84     30        1.0  Travel_Rarely        945                 Sales
...   ...        ...            ...        ...                   ...
1365   48        0.0  Travel_Rarely        715  Research_Development
1386   48        0.0  Travel_Rarely       1355  Research_Development
1407   50        0.0  Travel_Rarely        989  Research_Development
1428   50        0.0     Non-Travel        881  Research_Development
1449   52        0.0  Travel_Rarely        699  Research_Development

      DistanceFromHome      Education EducationField EnvironmentSatisfaction  \
0                   15        College  Life_Sciences                    High
21                   3  Below_College        Medical                  Medium
42                   2       Bachelor  Life_Sciences                  Medium
63                   1        College  Life_Sciences               Very_High
84                   9       Bachelor        Medical                  Medium
...                ...            ...            ...                     ...
1365                 1       Bachelor  Life_Sciences               Very_High
1386                 4         Master  Life_Sciences                    High
1407                 7        College        Medical                  Medium
1428                 2         Master  Life_Sciences                     Low
1449                 1         Master  Life_Sciences                    High

      Gender  ...  PerformanceRating RelationshipSatisfaction  \
0       Male  ...          Excellent                Very_High
21    Female  ...          Excellent                Very_High
42    Female  ...          Excellent                Very_High
63      Male  ...        Outstanding                Very_High
84      Male  ...          Excellent                     High
...      ...  ...                ...                      ...
1365    Male  ...          Excellent                     High
1386    Male  ...          Excellent                   Medium
1407  Female  ...          Excellent                Very_High
1428    Male  ...          Excellent                Very_High
1449    Male  ...          Excellent                      Low

      StockOptionLevel TotalWorkingYears TrainingTimesLastYear  \
0                    0                 0                    6
21                   0                 1                    3
42                   0                 1                    3
63                   1                 1                    2
84                   0                 1                    3
...                ...               ...                  ...
1365                 0                25                    3
1386                 0                27                    3
1407                 1                29                    2
```

```
1428                1                31                     3
1449                1                34                     5

     WorkLifeBalance  YearsAtCompany  YearsInCurrentRole  \
0             Better               0                   0
21            Better               1                   0
42            Better               1                   0
63            Better               1                   0
84              Good               1                   0
...              ...             ...                 ...
1365            Best               1                   0
1386          Better              15                  11
1407            Good              27                   3
1428          Better              31                   6
1449          Better              33                  18

      YearsSinceLastPromotion YearsWithCurrManager
0                           0                    0
21                          0                    0
42                          0                    0
63                          0                    1
84                          0                    0
...                       ...                  ...
1365                        0                    0
1386                        4                    8
1407                       13                    8
1428                       14                    7
1449                       11                    9

[70 rows x 31 columns]
```

0.2.4 Exercise 2.1.3

Is systematic sampling OK?

Systematic sampling has a problem: if the data has been sorted, or there is some sort of pattern or meaning behind the row order, then the resulting sample may not be representative of the whole population. The problem can be solved by shuffling the rows, but then systematic sampling is equivalent to simple random sampling.

Here you'll look at how to determine whether or not there is a problem.

Instructions

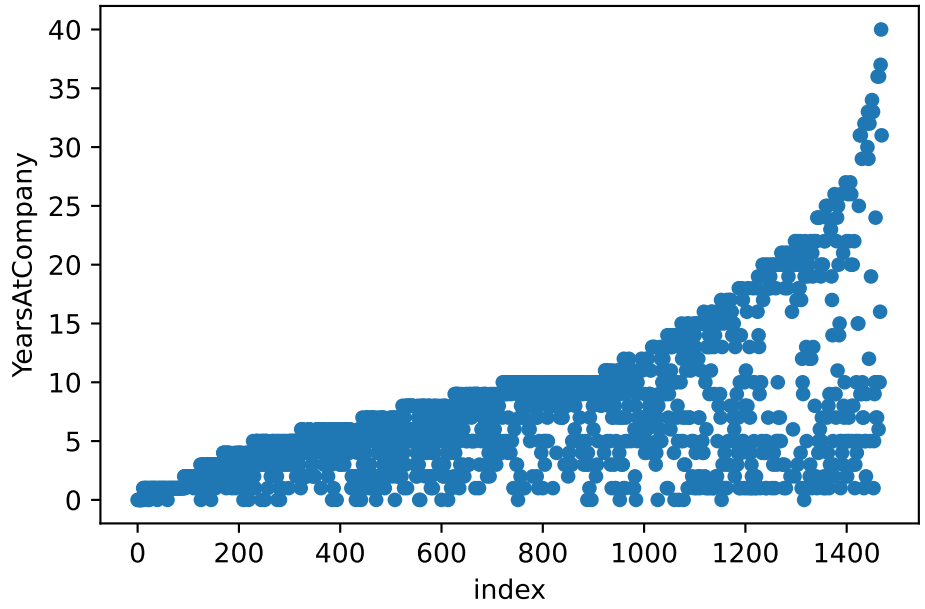
1. Add an index column to `attrition`, assigning the result to `attrition_id`.

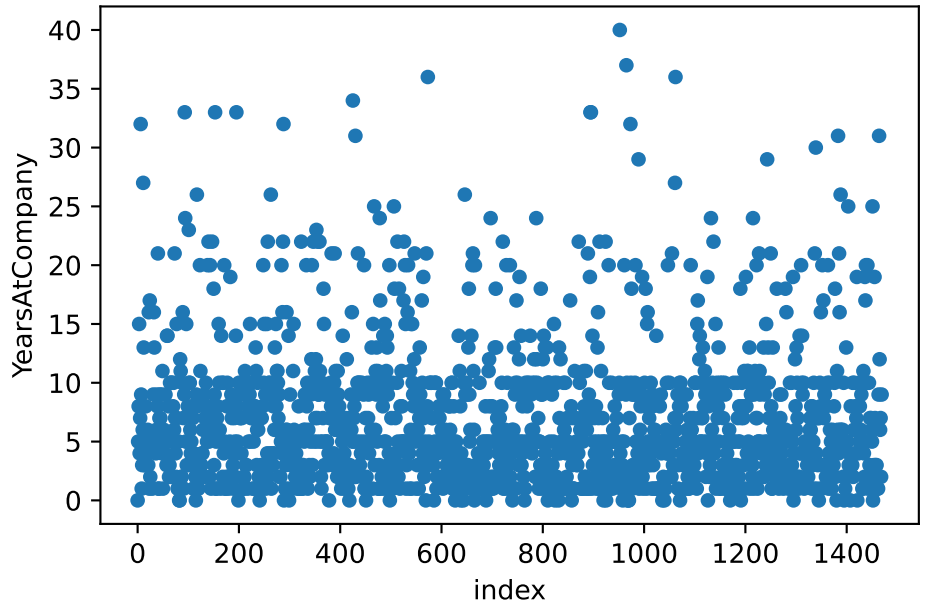
- Create a scatter plot of `YearsAtCompany` versus `index` for `attrition_id` using pandas `.plot()`.

2. Randomly shuffle the rows of `attrition`.

   - Reset the row indexes, and add an `index` column to `attrition`.
   - Repeat the scatter plot of `YearsAtCompany` versus `index`, this time using `attrition_shuffled`.

```
# Importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Importing the course arrays
attrition = pd.read_feather("datasets/attrition.feather")

# Add an index column to attrition_pop
attrition_id = attrition.reset_index()

# Plot YearsAtCompany vs. index for attrition_pop_id
attrition_id.plot(x="index", y="YearsAtCompany", kind="scatter")
plt.show()

# Shuffle the rows of attrition_pop
attrition_shuffled = attrition.sample(frac=1)

# Reset the row indexes and create an index column
attrition_shuffled = attrition_shuffled.reset_index(drop=True).reset_index()

# Plot YearsAtCompany vs. index for attrition_shuffled
attrition_shuffled.plot(x="index", y="YearsAtCompany", kind="scatter")
plt.show()
```





Question

Does a systematic sample always produce a sample similar to a simple random sample?
No, Systematic sampling has problems when the data are sorted or contain a pattern. Shuffling the rows makes it equivalent to simple random sampling.

0.2.5 Chapter 2.2: Stratified and weighted random sampling

Stratified sampling is a technique that allows us to sample a population that contains subgroups.

Coffees by country

For example, we could group the coffee ratings by country. If we count the number of coffees by country using the `value_counts` method, we can see that these six countries have the most data.

1. 1 The dataset lists Hawaii and Taiwan as countries for convenience, as they are notable coffee-growing regions.

Filtering for 6 countries

To make it easier to think about sampling subgroups, let's limit our analysis to these six countries. We can use the `.isin` method to filter the population and only return the rows corresponding to these six countries. This filtered dataset is stored as `coffee_ratings_top`.

Counts of a simple random sample

Let's take a ten percent simple random sample of the dataset using `.sample` with `frac` set to 0.1. We also set the `random_state` argument to ensure reproducibility. As with the whole dataset, we can look at the counts for each country. To make comparisons easier, we set `normalize` to `True` to convert the counts into a proportion, which shows what proportion of coffees in the sample came from each country.

Comparing proportions

Here are the proportions for the population and the ten percent sample side by side. Just by chance, in this sample, Taiwanese coffees form a disproportionately low percentage. The different makeup of the sample compared to the population could be a problem if we want to analyze the country of origin, for example.

Proportional stratified sampling

If we care about the proportions of each country in the sample closely matching those in the population, then we can group the data by country before taking the simple random sample. Note that we used the Python line continuation backslash here, which can be useful for breaking up longer chains of pandas code like this. Calling the `.sample` method after grouping takes a simple random sample within each country. Now the proportions of each country in the stratified sample are much closer to those in the population.

Equal counts stratified sampling

One variation of stratified sampling is to sample equal counts from each group, rather than an equal proportion. The code only has one change from before. This time, we use the `n` argument in `.sample` instead of `frac` to extract fifteen randomly-selected rows from each country. Here, the resulting sample has equal proportions of one-sixth from each country.

Weighted random sampling

A close relative of stratified sampling that provides even more flexibility is weighted random sampling. In this variant, we create a column of weights that adjust the relative probability of sampling each row. For example, suppose we thought that it was important to have a higher proportion of Taiwanese coffees in the sample than in the population. We create a condition, in this case, rows where the country of origin is Taiwan. Using the `where` function from NumPy, we can set a weight of two for rows that match the condition and a weight of one for rows that don't match the condition. This means when each row is randomly sampled, Taiwanese coffees have two times the chance of being picked compared to other coffees. When we call `.sample`, we pass the column of weights to the weights argument.

Weighted random sampling results

Here, we can see that Taiwan now contains seventeen percent of the sampled dataset, compared to eight-point-five percent in the population. This sort of weighted sampling is common in political polling, where we need to correct for under- or over-representation of demographic groups.

0.2.5.1 Exercise 2.2.1

Proportional stratified sampling

If you are interested in subgroups within the population, then you may need to carefully control the counts of each subgroup within the population. *Proportional stratified sampling* results in subgroup sizes within the sample that are representative of the subgroup sizes within the population. It is equivalent to performing a simple random sample on each subgroup.

Instructions

1. Get the proportion of employees by Education level from `attrition`.
2. Use proportional stratified sampling on `attrition_pop` to sample 40% of each `Education` group, setting the seed to 2022.
3. Get the proportion of employees by `Education` level from `attrition_strat`.

```
# Importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Importing the course arrays
attrition = pd.read_feather("datasets/attrition.feather")

# Proportion of employees by Education level
education_counts_pop = attrition['Education'].value_counts(normalize=True)

# Print education_counts_pop
print(education_counts_pop)

# Proportional stratified sampling for 40% of each Education group
attrition_strat = attrition.groupby('Education')\
.sample(frac=0.4, random_state=2022)

# Print the sample
print(attrition_strat)

# Calculate the Education level proportions from attrition_strat
education_counts_strat = attrition_strat['Education'].value_counts(normalize=True)

# Print education_counts_strat
print(education_counts_strat)
```

```
Education
Bachelor         0.389116
Master           0.270748
College          0.191837
Below_College    0.115646
Doctor           0.032653
Name: proportion, dtype: float64
      Age  Attrition     BusinessTravel  DailyRate            Department  \
1191   53        0.0      Travel_Rarely        238                 Sales
407    29        0.0  Travel_Frequently        995  Research_Development
1233   59        0.0  Travel_Frequently       1225                 Sales
366    37        0.0      Travel_Rarely        571  Research_Development
702    31        0.0  Travel_Frequently        163  Research_Development
...   ...        ...                ...        ...                   ...
733    38        0.0  Travel_Frequently        653  Research_Development
1061   44        0.0  Travel_Frequently        602       Human_Resources
1307   41        0.0      Travel_Rarely       1276                 Sales
1060   33        0.0      Travel_Rarely        516  Research_Development
```

```
177    29        0.0      Travel_Rarely        738  Research_Development

      DistanceFromHome      Education    EducationField  \
1191                 1  Below_College           Medical
407                  2  Below_College     Life_Sciences
1233                 1  Below_College     Life_Sciences
366                 10  Below_College     Life_Sciences
702                 24  Below_College  Technical_Degree
...                ...            ...               ...
733                 29         Doctor     Life_Sciences
1061                 1         Doctor   Human_Resources
1307                 2         Doctor     Life_Sciences
1060                 8         Doctor     Life_Sciences
177                  9         Doctor             Other

     EnvironmentSatisfaction  Gender  ...  PerformanceRating  \
1191               Very_High  Female  ...        Outstanding
407                      Low    Male  ...          Excellent
1233                     Low  Female  ...          Excellent
366                Very_High  Female  ...          Excellent
702                Very_High  Female  ...        Outstanding
...                      ...     ...  ...                ...
733                Very_High  Female  ...          Excellent
1061                     Low    Male  ...          Excellent
1307                  Medium  Female  ...          Excellent
1060               Very_High    Male  ...          Excellent
177                   Medium    Male  ...          Excellent

     RelationshipSatisfaction  StockOptionLevel TotalWorkingYears  \
1191                Very_High                 0                18
407                 Very_High                 1                 6
1233                Very_High                 0                20
366                    Medium                 2                 6
702                 Very_High                 0                 9
...                       ...               ...               ...
733                 Very_High                 0                10
1061                     High                 0                14
1307                  Medium                  1                22
1060                     Low                  0                14
177                      High                 0                 4

     TrainingTimesLastYear WorkLifeBalance  YearsAtCompany  \
1191                     2           Best              14
407                      0           Best               6
1233                     2           Good               4
```

```
366                       3           Good               5
702                       3           Good               5
...                     ...            ...             ...
733                       2         Better              10
1061                      3         Better              10
1307                      2         Better              18
1060                      6         Better               0
177                       2         Better               3

      YearsInCurrentRole  YearsSinceLastPromotion YearsWithCurrManager
1191                   7                        8                   10
407                    4                        1                    3
1233                   3                        1                    3
366                    3                        4                    3
702                    4                        1                    4
...                  ...                      ...                 ...
733                    3                        9                    9
1061                   7                        0                    2
1307                  16                       11                    8
1060                   0                        0                    0
177                    2                        2                    2

[588 rows x 31 columns]
Education
Bachelor         0.389456
Master           0.270408
College          0.192177
Below_College    0.115646
Doctor           0.032313
Name: proportion, dtype: float64
```

> **Note**
>
> *By grouping then sampling, the size of each group in the sample is representative of the size of the sample in the population.*

0.2.6 Exercise 2.2.2

Equal counts stratified sampling

If one subgroup is larger than another subgroup in the population, but you don't want to reflect that difference in your analysis, then you can use *equal counts stratified sampling* to generate samples where each subgroup has the same amount of data. For example, if you are analyzing

blood types, O is the most common blood type worldwide, but you may wish to have equal amounts of O, A, B, and AB in your sample.

Instructions

1. Use equal counts stratified sampling on `attrition` to get 30 employees from each `Education` group, setting the seed to 2022.
2. Get the proportion of employees by `Education` level from `attrition_eq`.

```
# Importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Importing the course arrays
attrition = pd.read_feather("datasets/attrition.feather")

# Proportion of employees by Education level
education_counts_pop = attrition['Education'].value_counts(normalize=True)

# Print education_counts_pop
print(education_counts_pop)

# Get 30 employees from each Education group
attrition_eq = attrition.groupby('Education')\
.sample(n=30, random_state=2022)

# Print the sample
print(attrition_eq)

# Get the proportions from attrition_eq
education_counts_eq = attrition_eq['Education'].value_counts(normalize=True)

# Print the results
print(education_counts_eq)
```

```
Education
Bachelor         0.389116
Master           0.270748
College          0.191837
Below_College    0.115646
Doctor           0.032653
Name: proportion, dtype: float64
      Age  Attrition     BusinessTravel  DailyRate            Department  \
1191   53        0.0      Travel_Rarely        238                 Sales
```

```
407    29        0.0  Travel_Frequently        995  Research_Development
1233   59        0.0  Travel_Frequently       1225                 Sales
366    37        0.0      Travel_Rarely        571  Research_Development
702    31        0.0  Travel_Frequently        163  Research_Development
...   ...        ...                ...        ...                   ...
774    33        0.0      Travel_Rarely        922  Research_Development
869    45        0.0      Travel_Rarely       1015  Research_Development
530    32        0.0      Travel_Rarely        120  Research_Development
1049   48        0.0      Travel_Rarely        163                 Sales
350    29        1.0      Travel_Rarely        408  Research_Development

      DistanceFromHome      Education    EducationField  \
1191                 1  Below_College           Medical
407                  2  Below_College     Life_Sciences
1233                 1  Below_College     Life_Sciences
366                 10  Below_College     Life_Sciences
702                 24  Below_College  Technical_Degree
...                ...            ...               ...
774                  1         Doctor           Medical
869                  5         Doctor           Medical
530                  6         Doctor     Life_Sciences
1049                 2         Doctor         Marketing
350                 25         Doctor  Technical_Degree

     EnvironmentSatisfaction  Gender  ...  PerformanceRating  \
1191               Very_High  Female  ...        Outstanding
407                      Low    Male  ...          Excellent
1233                     Low  Female  ...          Excellent
366                Very_High  Female  ...          Excellent
702                Very_High  Female  ...        Outstanding
...                      ...     ...  ...                ...
774                      Low  Female  ...          Excellent
869                     High  Female  ...          Excellent
530                     High    Male  ...        Outstanding
1049                  Medium  Female  ...          Excellent
350                     High  Female  ...          Excellent

     RelationshipSatisfaction  StockOptionLevel TotalWorkingYears  \
1191                Very_High                 0                18
407                 Very_High                 1                 6
1233                Very_High                 0                20
366                    Medium                 2                 6
702                 Very_High                 0                 9
...                       ...               ...               ...
774                      High                 1                10
```

```
869                      Low                 0                10
530                      Low                 0                 8
1049                     Low                 1                14
350                   Medium                 0                 6

     TrainingTimesLastYear WorkLifeBalance  YearsAtCompany  \
1191                     2            Best              14
407                      0            Best               6
1233                     2            Good               4
366                      3            Good               5
702                      3            Good               5
...                    ...             ...             ...
774                      2          Better               6
869                      3          Better              10
530                      2          Better               5
1049                     2          Better               9
350                      2            Best               2

      YearsInCurrentRole  YearsSinceLastPromotion YearsWithCurrManager
1191                   7                        8                  10
407                    4                        1                   3
1233                   3                        1                   3
366                    3                        4                   3
702                    4                        1                   4
...                  ...                      ...                 ...
774                    1                        0                   5
869                    7                        1                   4
530                    4                        1                   4
1049                   7                        6                   7
350                    2                        1                   1

[150 rows x 31 columns]
Education
Below_College    0.2
College          0.2
Bachelor         0.2
Master           0.2
Doctor           0.2
Name: proportion, dtype: float64
```

> **Note**
>
> *If you want each subgroup to have equal weight in your analysis, then equal counts stratified sampling is the appropriate technique.*

0.2.7 Exercise 2.2.3

Weighted sampling

Stratified sampling provides rules about the probability of picking rows from your dataset at the subgroup level. A generalization of this is weighted sampling, which lets you specify rules about the probability of picking rows at the row level. The probability of picking any given row is proportional to the weight value for that row.

Instructions

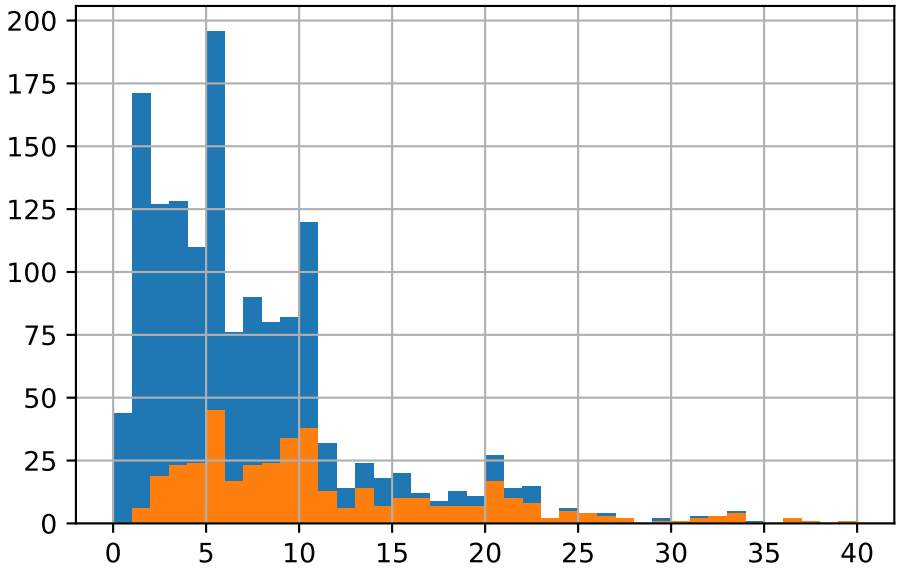
1. Plot `YearsAtCompany` from `attrition` as a histogram with bins of width `1` from `0` to `40`.
2. Sample 400 employees from `attrition` weighted by `YearsAtCompany`.
3. Plot `YearsAtCompany` from `attrition_weight` as a histogram with bins of width `1` from `0` to `40`.
4. Which is higher? The mean `YearsAtCompany` from `attrition` or the mean `YearsAtCompany` from `attrition_weight`? **Answer**: *The weighted sample mean is around 11, which is higher than the population mean of around 7. The fact that the two numbers are different means that the weighted simple random sample is biased.*

```
# Importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Importing the course arrays
attrition = pd.read_feather("datasets/attrition.feather")

# Plot YearsAtCompany from attrition_pop as a histogram
attrition['YearsAtCompany'].hist(bins=np.arange(0,41,1))

# Sample 400 employees weighted by YearsAtCompany
attrition_weight = attrition.sample(n=400, weights='YearsAtCompany')

# Print the sample
print(attrition_weight)

# Plot YearsAtCompany from attrition_weight as a histogram
attrition_weight['YearsAtCompany'].hist(bins=np.arange(0, 41, 1))
plt.show()

# The mean YearsAtCompany from attrition dataset
print(attrition['YearsAtCompany'].mean())
```

```
# The mean YearsAtCompany from attrition_weight
print(attrition_weight['YearsAtCompany'].mean())
```

```
      Age  Attrition     BusinessTravel  DailyRate            Department  \
1036   46        0.0      Travel_Rarely       1277                 Sales
489    25        0.0      Travel_Rarely        882  Research_Development
1314   49        0.0      Travel_Rarely        174                 Sales
1441   50        0.0  Travel_Frequently       1234  Research_Development
1152   36        0.0      Travel_Rarely       1223  Research_Development
...   ...        ...                ...        ...                   ...
795    33        1.0      Travel_Rarely        527  Research_Development
939    38        0.0      Travel_Rarely       1009                 Sales
886    41        0.0         Non-Travel        552       Human_Resources
1165   35        0.0      Travel_Rarely       1296  Research_Development
538    26        1.0      Travel_Rarely        950                 Sales

      DistanceFromHome      Education    EducationField  \
1036                 2       Bachelor     Life_Sciences
489                 19  Below_College           Medical
1314                 8         Master  Technical_Degree
1441                20         Doctor           Medical
1152                 8       Bachelor  Technical_Degree
...                ...            ...               ...
795                  1         Master             Other
939                  2        College     Life_Sciences
886                  4       Bachelor   Human_Resources
1165                 5         Master  Technical_Degree
538                  4         Master         Marketing

     EnvironmentSatisfaction  Gender  ...  PerformanceRating  \
1036                    High    Male  ...          Excellent
489                Very_High    Male  ...          Excellent
1314               Very_High    Male  ...          Excellent
1441                  Medium    Male  ...          Excellent
1152                    High  Female  ...          Excellent
...                      ...     ...  ...                ...
795                Very_High    Male  ...          Excellent
939                   Medium  Female  ...          Excellent
886                     High    Male  ...          Excellent
1165                    High    Male  ...          Excellent
538                Very_High    Male  ...          Excellent

     RelationshipSatisfaction  StockOptionLevel TotalWorkingYears  \
1036                   Medium                 1                13
489                      High                 3                 7
```

```
1314                    Medium                 1                 22
1441                      High                 1                 32
1152                    Medium                 3                 17
...                        ...               ...                ...
795                       High                 0                 10
939                  Very_High                 1                 11
886                     Medium                 1                 10
1165                 Very_High                 0                 17
538                     Medium                 0                  8

     TrainingTimesLastYear WorkLifeBalance  YearsAtCompany  \
1036                     5            Good              10
489                      6            Good               3
1314                     3          Better               9
1441                     3          Better              30
1152                     2          Better              17
...                    ...             ...             ...
795                      2            Good              10
939                      3          Better               7
886                      4          Better               3
1165                     5          Better              16
538                      0          Better               8

      YearsInCurrentRole  YearsSinceLastPromotion YearsWithCurrManager
1036                   6                        0                    3
489                    2                        1                    2
1314                   8                        2                    3
1441                   8                       12                   13
1152                  14                       12                    8
...                  ...                      ...                  ...
795                    9                        7                    8
939                    7                        1                    7
886                    2                        1                    2
1165                   6                        0                   13
538                    7                        7                    4

[400 rows x 31 columns]
```




```
7.0081632653061225
10.95
```

0.2.8 Chapter 2.3: Cluster sampling

One problem with stratified sampling is that we need to collect data from every subgroup. In cases where collecting data is expensive, for example, when we have to physically travel to a location to collect it, it can make our analysis prohibitively expensive. There's a cheaper alternative called cluster sampling.

Stratified sampling vs. cluster sampling

The stratified sampling approach was to split the population into subgroups, then use simple random sampling on each of them. Cluster sampling means that we limit the number of subgroups in the analysis by picking a few of them with simple random sampling. We then perform simple random sampling on each subgroup as before.

Varieties of coffee

Let's return to the coffee dataset and look at the varieties of coffee. In this image, each bean represents the whole subgroup rather than an individual coffee, and there are twenty-eight of them. To extract unique varieties, we use the `.unique` method. This returns an array, so wrapping it in the list function creates a list of unique varieties. Let's suppose that it's expensive to work with all of the different varieties. Enter cluster sampling.

Stage 1: sampling for subgroups

The first stage of cluster sampling is to randomly cut down the number of varieties, and we do this by randomly selecting them. Here, we've used the `random.sample` function from the random package to get three varieties, specified using the argument `k`.

Stage 2: sampling each group

The second stage of cluster sampling is to perform simple random sampling on each of the three varieties we randomly selected. We first filter the dataset for rows where the variety is one of the three selected, using the `.isin` method. To ensure that the `isin` filtering removes levels with zero rows, we apply the `cat.remove_unused_categories` method on the Series of focus, which is variety here. If we exclude this method, we might receive an error when sampling by variety level. The pandas code is the same as for stratified sampling. Here, we've opted for equal counts sampling, with five rows from each remaining variety.

Stage 2 output

Here's the first few columns of the result. Notice that there are the fifteen rows, which we'd expect from sampling five rows from three varieties.

Multistage sampling

Note that we had two stages in the cluster sampling. We randomly sampled the subgroups to include, then we randomly sampled rows from those subgroups. Cluster sampling is a special case of multistage sampling. It's possible to use more than two stages. A common example is national surveys, which can include several levels of administrative regions, like states, counties, cities, and neighborhoods.

0.2.9 Exercise 2.3.1

Performing cluster sampling

Now that you know when to use cluster sampling, it's time to put it into action. In this exercise, you'll explore the JobRole column of the attrition dataset. You can think of each job role as a subgroup of the whole population of employees.

Use a seed of 19790801 to set the seed with `random.seed()`.

Instructions

1.

- Create a list of unique `JobRole` values from `attrition`, and assign to `job_roles_pop`.
- Randomly sample four `JobRole` values from `job_roles_pop`.

2. Subset attrition_pop for the sampled job roles by filtering for rows where `JobRole` is in `job_roles_samp`.
3.

- Remove any unused categories from `JobRole`.
- For each job role in the filtered dataset, take a random sample of ten rows, setting the seed to 2022.

```
# Importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import random

# Importing the course arrays
attrition = pd.read_feather("datasets/attrition.feather")

# Set the seed
random.seed(19790801)

# Create a list of unique JobRole values
job_roles_pop = list(attrition['JobRole'].unique())

# Randomly sample four JobRole values
job_roles_samp = random.sample(job_roles_pop, k=4)

# Print the result
print(job_roles_samp)

# Filter for rows where JobRole is in job_roles_samp
jobrole_condition = attrition['JobRole'].isin(job_roles_samp)
attrition_filtered = attrition[jobrole_condition]

# Print the result
print(attrition_filtered)

# Remove categories with no rows
attrition_filtered['JobRole'] = attrition_filtered['JobRole'].cat.remove_unused_categories()
```

```
# Randomly sample 10 employees from each sampled job role
attrition_clust = attrition_filtered.groupby('JobRole')\
.sample(n=10, random_state=2022)


# Print the sample
print(attrition_clust)
```

```
['Research_Director', 'Research_Scientist', 'Human_Resources', 'Manager']
      Age  Attrition BusinessTravel  DailyRate            Department  \
0      21        0.0  Travel_Rarely        391  Research_Development
5      27        0.0     Non-Travel        443  Research_Development
6      18        0.0     Non-Travel        287  Research_Development
10     18        0.0     Non-Travel       1431  Research_Development
17     31        0.0  Travel_Rarely       1082  Research_Development
...   ...        ...            ...        ...                   ...
1462   54        0.0  Travel_Rarely        584  Research_Development
1464   55        0.0  Travel_Rarely        452  Research_Development
1465   55        0.0  Travel_Rarely       1117                 Sales
1466   58        0.0     Non-Travel        350                 Sales
1469   58        1.0  Travel_Rarely        286  Research_Development

      DistanceFromHome Education EducationField EnvironmentSatisfaction  \
0                   15   College  Life_Sciences                    High
5                    3  Bachelor        Medical               Very_High
6                    5   College  Life_Sciences                  Medium
10                  14  Bachelor        Medical                  Medium
17                   1    Master        Medical                    High
...                ...       ...            ...                     ...
1462                22    Doctor        Medical                  Medium
1464                 1  Bachelor        Medical               Very_High
1465                18    Doctor  Life_Sciences                     Low
1466                 2  Bachelor        Medical                  Medium
1469                 2    Master  Life_Sciences               Very_High

      Gender  ...  PerformanceRating RelationshipSatisfaction  \
0       Male  ...          Excellent               Very_High
5       Male  ...          Excellent                    High
6       Male  ...          Excellent               Very_High
10    Female  ...          Excellent                    High
17      Male  ...          Excellent                  Medium
...      ...  ...                ...                     ...
1462  Female  ...        Outstanding                    High
1464    Male  ...          Excellent                    High
1465  Female  ...        Outstanding               Very_High
```

```
1466    Male  ...        Outstanding                Very_High
1469    Male  ...          Excellent                Very_High

      StockOptionLevel TotalWorkingYears TrainingTimesLastYear  \
0                    0                 0                     6
5                    3                 0                     6
6                    0                 0                     2
10                   0                 0                     4
17                   0                 1                     4
...                ...               ...                   ...
1462                 1                36                     6
1464                 0                37                     2
1465                 0                37                     2
1466                 1                37                     0
1469                 0                40                     2

     WorkLifeBalance  YearsAtCompany  YearsInCurrentRole  \
0             Better               0                   0
5               Good               0                   0
6             Better               0                   0
10               Bad               0                   0
17            Better               1                   1
...              ...             ...                 ...
1462          Better              10                   8
1464          Better              36                  10
1465          Better              10                   9
1466            Good              16                   9
1469          Better              31                  15

      YearsSinceLastPromotion YearsWithCurrManager
0                           0                    0
5                           0                    0
6                           0                    0
10                          0                    0
17                          1                    0
...                       ...                  ...
1462                        4                    7
1464                        4                   13
1465                        7                    7
1466                       14                   14
1469                       13                    8

[526 rows x 31 columns]
      Age  Attrition     BusinessTravel  DailyRate            Department  \
1348   44        1.0      Travel_Rarely       1376       Human_Resources
```

```
886    41         0.0         Non-Travel        552       Human_Resources
983    39         0.0      Travel_Rarely        141       Human_Resources
88     27         1.0  Travel_Frequently       1337       Human_Resources
189    34         0.0      Travel_Rarely        829       Human_Resources
160    24         0.0  Travel_Frequently        897       Human_Resources
839    46         0.0      Travel_Rarely        991       Human_Resources
966    30         0.0      Travel_Rarely       1240       Human_Resources
162    28         0.0         Non-Travel        280       Human_Resources
1231   37         0.0      Travel_Rarely       1239       Human_Resources
1375   44         0.0      Travel_Rarely       1315  Research_Development
1462   54         0.0      Travel_Rarely        584  Research_Development
1316   45         0.0  Travel_Frequently        364  Research_Development
1356   48         0.0  Travel_Frequently        117  Research_Development
1387   48         0.0         Non-Travel       1262  Research_Development
1321   54         0.0         Non-Travel        142       Human_Resources
1266   50         0.0      Travel_Rarely       1452  Research_Development
1330   46         0.0      Travel_Rarely        406                 Sales
1052   59         0.0      Travel_Rarely       1089                 Sales
1449   52         0.0      Travel_Rarely        699  Research_Development
1439   58         0.0      Travel_Rarely       1055  Research_Development
1339   58         0.0  Travel_Frequently       1216  Research_Development
1426   49         0.0      Travel_Rarely       1245  Research_Development
1415   48         0.0      Travel_Rarely       1224  Research_Development
1322   51         0.0      Travel_Rarely        684  Research_Development
1284   40         0.0      Travel_Rarely       1308  Research_Development
1149   37         0.0      Travel_Rarely        161  Research_Development
1126   42         0.0      Travel_Rarely        810  Research_Development
1374   46         0.0      Travel_Rarely       1009  Research_Development
1050   33         0.0      Travel_Rarely        213  Research_Development
86     26         0.0      Travel_Rarely        482  Research_Development
930    52         1.0      Travel_Rarely        723  Research_Development
860    37         0.0      Travel_Rarely        674  Research_Development
36     20         1.0      Travel_Rarely       1362  Research_Development
997    32         0.0      Travel_Rarely        824  Research_Development
1358   45         0.0      Travel_Rarely       1339  Research_Development
993    41         0.0  Travel_Frequently       1200  Research_Development
421    34         0.0      Travel_Rarely        181  Research_Development
789    28         1.0      Travel_Rarely        654  Research_Development
94     36         1.0      Travel_Rarely        318  Research_Development

      DistanceFromHome      Education    EducationField  \
1348                 1        College           Medical
886                  4       Bachelor   Human_Resources
983                  3       Bachelor   Human_Resources
88                  22       Bachelor   Human_Resources
```

```
189                  3        College   Human_Resources
160                 10       Bachelor           Medical
839                  1        College     Life_Sciences
966                  9       Bachelor   Human_Resources
162                  1        College     Life_Sciences
1231                 8        College             Other
1375                 3         Master             Other
1462                22         Doctor           Medical
1316                25       Bachelor           Medical
1356                22       Bachelor           Medical
1387                 1         Master           Medical
1321                26       Bachelor   Human_Resources
1266                11       Bachelor     Life_Sciences
1330                 3  Below_College         Marketing
1052                 1        College  Technical_Degree
1449                 1         Master     Life_Sciences
1439                 1       Bachelor           Medical
1339                15         Master     Life_Sciences
1426                18         Master     Life_Sciences
1415                10       Bachelor     Life_Sciences
1322                 6       Bachelor     Life_Sciences
1284                14       Bachelor           Medical
1149                10       Bachelor     Life_Sciences
1126                23         Doctor     Life_Sciences
1374                 2       Bachelor     Life_Sciences
1050                 7       Bachelor           Medical
86                   1        College     Life_Sciences
930                  8         Master           Medical
860                 13       Bachelor           Medical
36                  10  Below_College           Medical
997                  5        College     Life_Sciences
1358                 7       Bachelor     Life_Sciences
993                 22       Bachelor     Life_Sciences
421                  2         Master           Medical
789                  1        College     Life_Sciences
94                   9       Bachelor           Medical

     EnvironmentSatisfaction  Gender  ...  PerformanceRating  \
1348                  Medium    Male  ...          Excellent
886                     High    Male  ...          Excellent
983                     High  Female  ...          Excellent
88                       Low  Female  ...          Excellent
189                     High    Male  ...          Excellent
160                      Low    Male  ...          Excellent
839                Very_High  Female  ...          Excellent
```

```
966                    High    Male  ...          Excellent
162                    High    Male  ...          Excellent
1231                   High    Male  ...          Excellent
1375              Very_High    Male  ...          Excellent
1462                 Medium  Female  ...        Outstanding
1316                 Medium  Female  ...        Outstanding
1356              Very_High  Female  ...          Excellent
1387                    Low    Male  ...        Outstanding
1321              Very_High  Female  ...          Excellent
1266                   High  Female  ...          Excellent
1330                    Low    Male  ...          Excellent
1052                 Medium    Male  ...          Excellent
1449                   High    Male  ...          Excellent
1439              Very_High  Female  ...        Outstanding
1339                    Low    Male  ...          Excellent
1426              Very_High    Male  ...          Excellent
1415              Very_High    Male  ...          Excellent
1322                    Low    Male  ...          Excellent
1284                   High    Male  ...          Excellent
1149                   High  Female  ...        Outstanding
1126                    Low  Female  ...          Excellent
1374                    Low    Male  ...          Excellent
1050                   High    Male  ...          Excellent
86                   Medium  Female  ...          Excellent
930                    High    Male  ...          Excellent
860                     Low    Male  ...          Excellent
36                Very_High    Male  ...          Excellent
997               Very_High  Female  ...          Excellent
1358                 Medium    Male  ...          Excellent
993               Very_High  Female  ...          Excellent
421               Very_High    Male  ...          Excellent
789                     Low  Female  ...          Excellent
94                Very_High    Male  ...          Excellent

     RelationshipSatisfaction  StockOptionLevel TotalWorkingYears  \
1348                Very_High                 1                24
886                    Medium                 1                10
983                    High                   1                12
88                      Low                   0                 1
189                    High                   1                 4
160                 Very_High                 1                 3
839                    High                   0                10
966                 Very_High                 0                12
162                    Medium                 1                 3
1231                   High                   0                19
```

1375	Low	1	26
1462	High	1	36
1316	High	0	22
1356	Medium	1	24
1387	High	0	27
1321	High	0	23
1266	Medium	0	21
1330	Very_High	1	23
1052	High	1	14
1449	Low	1	34
1439	High	1	32
1339	Medium	0	23
1426	High	1	31
1415	Very_High	0	29
1322	High	0	23
1284	Low	0	21
1149	Low	1	16
1126	Medium	0	16
1374	High	0	26
1050	Very_High	0	14
86	High	1	1
930	Low	0	11
860	Low	0	10
36	Very_High	0	1
997	Low	1	12
1358	High	1	25
993	Low	2	12
421	Low	3	6
789	Very_High	0	10
94	Low	1	2

	TrainingTimesLastYear	WorkLifeBalance	YearsAtCompany	\
1348	1	Better	20	
886	4	Better	3	
983	3	Bad	8	
88	2	Better	1	
189	1	Bad	3	
160	2	Better	2	
839	3	Best	7	
966	2	Bad	11	
162	2	Better	3	
1231	4	Good	10	
1375	2	Best	2	
1462	6	Better	10	
1316	4	Better	0	

1356	3	Better	22
1387	3	Good	5
1321	3	Better	5
1266	5	Better	5
1330	3	Better	12
1052	1	Bad	6
1449	5	Better	33
1439	3	Better	9
1339	3	Better	2
1426	5	Better	31
1415	3	Better	22
1322	5	Better	20
1284	2	Best	20
1149	2	Better	16
1126	2	Better	1
1374	2	Bad	3
1050	3	Best	13
86	3	Good	1
930	3	Good	8
860	2	Better	10
36	5	Better	1
997	2	Better	7
1358	2	Better	1
993	4	Good	6
421	3	Better	5
789	4	Better	7
94	0	Good	1

	YearsInCurrentRole	YearsSinceLastPromotion	YearsWithCurrManager
1348	6	3	6
886	2	1	2
983	3	3	6
88	0	0	0
189	2	0	2
160	2	2	1
839	6	5	7
966	9	4	7
162	2	2	2
1231	0	4	7
1375	2	0	1
1462	8	4	7
1316	0	0	0
1356	17	4	7
1387	4	2	1
1321	3	4	4

1266	4	4	4
1330	9	4	9
1052	4	0	4
1449	18	11	9
1439	8	1	5
1339	2	2	2
1426	9	0	9
1415	10	12	9
1322	18	15	15
1284	7	4	9
1149	11	6	8
1126	0	0	0
1374	2	0	1
1050	9	3	7
86	0	1	0
930	2	7	7
860	8	3	7
36	0	1	1
997	1	2	5
1358	0	0	0
993	2	3	3
421	0	1	2
789	7	3	7
94	0	0	0

[40 rows x 31 columns]

0.2.10 Chapter 2.4: Comparing sampling methods

Let's review the various sampling techniques we learned about.

Review of sampling techniques - setup

For convenience, we'll stick to the six countries with the most coffee varieties that we used before. This corresponds to eight hundred and eighty rows and eight columns, retrieved using the `.shape` attribute.

Review of simple random sampling

Simple random sampling uses `.sample` with either `n` or `frac` set to determine how many rows to pseudo-randomly choose, given a seed value set with `random_state`. The simple random sample returns two hundred and ninety-three rows because we specified `frac` as one-third, and one-third of eight hundred and eighty is just over two hundred and ninety-three.

Review of stratified sampling

Stratified sampling groups by the country subgroup before performing simple random sampling on each subgroup. Given each of these top countries have quite a few rows, stratifying produces the same number of rows as the simple random sample.

Review of cluster sampling

In the cluster sample, we've used two out of six countries to roughly mimic frac equals one-third from the other sample types. Setting n equal to one-sixth of the total number of rows gives roughly equal sample sizes in each of the two subgroups. Using `.shape` again, we see that this cluster sample has close to the same number of rows: two-hundred-ninety-two compared to two-hundred-ninety-three for the other sample types.

Calculating mean cup points

Let's calculate a population parameter, the mean of the total cup points. When the population parameter is the mean of a field, it's often called the population mean. Remember that in real-life scenarios, we typically wouldn't know what the population mean is. Since we have it here, though, we can use this value of eighty-one-point-nine as a gold standard to measure against. Now we'll calculate the same value using each of the sampling techniques we've discussed. These are point estimates of the mean, often called sample means. The simple and stratified sample means are really close to the population mean. Cluster sampling isn't quite as close, but that's typical. Cluster sampling is designed to give us an answer that's almost as good while using less data.

Mean cup points by country: simple random

Here's a slightly more complicated calculation of the mean total cup points for each country. We group by country before calculating the mean to return six numbers. So how do the numbers from the simple random sample compare? The sample means are pretty close to the population means.

Mean cup points by country: stratified

The same is true of the sample means from the stratified technique. Each sample mean is relatively close to the population mean.

Mean cup points by country: cluster

With cluster sampling, while the sample means are pretty close to the population means, the obvious limitation is that we only get values for the two countries that were included in the sample. If the mean cup points for each country is an important metric in our analysis, cluster sampling would be a bad idea.

0.2.11 Exercise 2.4.1

3 kinds of sampling

You're going to compare the performance of point estimates using simple, stratified, and cluster sampling. Before doing that, you'll have to set up the samples.

You'll use the `RelationshipSatisfaction` column of the `attrition` dataset, which categorizes the employee's relationship with the company. It has four levels: `Low`, `Medium`, `High`, and `Very_High`.

Instructions

1. Perform simple random sampling on `attrition` to get one-quarter of the population, setting the seed to 2022.
2. Perform stratified sampling on `attrition` to sample one-quarter of each `RelationshipSatisfaction` group, setting the seed to 2022.
3. Create a list of unique values from `attrition`'s `RelationshipSatisfaction` column. Randomly sample `satisfaction_unique` to get two values. Subset the population for rows where `RelationshipSatisfaction` is in `satisfaction_samp` and clear any unused categories from `RelationshipSatisfaction`; assign to `attrition_clust_prep`. Perform cluster sampling on the selected satisfaction groups, sampling one quarter of the *population* and setting the seed to 2022.

```
# Importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import random

# Importing the course arrays
attrition = pd.read_feather("datasets/attrition.feather")

# Perform simple random sampling to get 0.25 of the population
attrition_srs = attrition.sample(frac=1/4, random_state=2022)

# Perform stratified sampling to get 0.25 of each relationship group
attrition_strat = attrition.groupby('RelationshipSatisfaction')\
.sample(frac=1/4, random_state=2022)

# Create a list of unique RelationshipSatisfaction values
satisfaction_unique = list(attrition['RelationshipSatisfaction'].unique())

# Randomly sample 2 unique satisfaction values
satisfaction_samp = random.sample(satisfaction_unique, k=2)
```

```
# Filter for satisfaction_samp and clear unused categories from RelationshipSatisfaction
satis_condition = attrition['RelationshipSatisfaction'].isin(satisfaction_samp)
attrition_clust_prep = attrition[satis_condition]
attrition_clust_prep['RelationshipSatisfaction'] = attrition_clust_prep['RelationshipSatisfaction

# Perform cluster sampling on the selected group, getting 0.25 of attrition_clust_prep
attrition_clust = attrition_clust_prep.groupby("RelationshipSatisfaction")\
.sample(n=len(attrition) // 6, random_state=2022)

print(attrition_clust)
```

```
      Age  Attrition     BusinessTravel  DailyRate            Department  \
1381   45        1.0      Travel_Rarely       1449                 Sales
1357   42        0.0      Travel_Rarely        300  Research_Development
924    30        0.0      Travel_Rarely        288  Research_Development
1224   46        0.0      Travel_Rarely       1003  Research_Development
1277   48        0.0      Travel_Rarely       1236  Research_Development
...   ...        ...                ...        ...                   ...
357    27        0.0      Travel_Rarely        798  Research_Development
424    44        1.0  Travel_Frequently        429  Research_Development
1182   36        0.0  Travel_Frequently        884  Research_Development
1055   34        0.0  Travel_Frequently        669  Research_Development
962    34        0.0      Travel_Rarely       1031  Research_Development

      DistanceFromHome Education EducationField EnvironmentSatisfaction  \
1381                 2  Bachelor      Marketing                      Low
1357                 2  Bachelor  Life_Sciences                      Low
924                  2  Bachelor  Life_Sciences                     High
1224                 8    Master  Life_Sciences                Very_High
1277                 1    Master  Life_Sciences                Very_High
...                ...       ...            ...                      ...
357                  6    Master        Medical                      Low
424                  1   College        Medical                     High
1182                23   College        Medical                     High
1055                 1  Bachelor        Medical                Very_High
962                  6    Master  Life_Sciences                     High

      Gender  ...  PerformanceRating RelationshipSatisfaction  \
1381  Female  ...          Excellent                      Low
1357    Male  ...          Excellent                      Low
924     Male  ...          Excellent                      Low
1224  Female  ...        Outstanding                      Low
1277  Female  ...          Excellent                      Low
...      ...  ...                ...                      ...
357   Female  ...          Excellent                     High
```

```
424     Male  ...          Excellent                       High
1182    Male  ...          Excellent                       High
1055    Male  ...        Outstanding                       High
962   Female  ...          Excellent                       High

      StockOptionLevel TotalWorkingYears TrainingTimesLastYear  \
1381                 0                26                     2
1357                 0                24                     2
924                  3                11                     3
1224                 3                19                     2
1277                 1                21                     3
...                ...               ...                   ...
357                  2                 6                     5
424                  3                 6                     2
1182                 1                17                     3
1055                 0                14                     3
962                  1                12                     3

     WorkLifeBalance  YearsAtCompany  YearsInCurrentRole  \
1381          Better              24                  10
1357            Good              22                   6
924           Better              11                  10
1224          Better              16                  13
1277             Bad               3                   2
...              ...             ...                 ...
357             Good               5                   3
424             Good               5                   3
1182          Better               5                   2
1055          Better              13                   9
962           Better               1                   0

      YearsSinceLastPromotion YearsWithCurrManager
1381                        1                  11
1357                        4                  14
924                        10                   8
1224                        1                   7
1277                        0                   2
...                       ...                 ...
357                         0                   3
424                         2                   3
1182                        0                   3
1055                        4                   9
962                         0                   0

[490 rows x 31 columns]
```

0.2.12 Exercise 2.4.4

Comparing point estimates

Now that you have three types of sample (simple, stratified, and cluster), you can compare point estimates from each sample to the population parameter. That is, you can calculate the same summary statistic on each sample and see how it compares to the summary statistic for the population.

Here, we'll look at how satisfaction with the company affects whether or not the employee leaves the company. That is, you'll calculate the proportion of employees who left the company (they have an Attrition value of `1`) for each value of `RelationshipSatisfaction`.

Instructions

1. Group `attrition` by `RelationshipSatisfaction` levels and calculate the mean of Attrition for each level.
2. Calculate the proportion of employee attrition for each relationship satisfaction group, this time on the simple random sample, `attrition_srs`.
3. Calculate the proportion of employee attrition for each relationship satisfaction group, this time on the stratified sample, `attrition_strat`.
4. Calculate the proportion of employee attrition for each relationship satisfaction group, this time on the cluster sample, `attrition_clust`.

```
# Importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import random

# Importing the course arrays
attrition = pd.read_feather("datasets/attrition.feather")

# Perform simple random sampling to get 0.25 of the population
attrition_srs = attrition.sample(frac=1/4, random_state=2022)

# Perform stratified sampling to get 0.25 of each relationship group
attrition_strat = attrition.groupby('RelationshipSatisfaction')\
.sample(frac=1/4, random_state=2022)

# Mean Attrition by RelationshipSatisfaction group
mean_attrition_pop = attrition.groupby('RelationshipSatisfaction')\
['Attrition'].mean()

# Print the result
print(mean_attrition_pop)
```

```
# Calculate the same thing for the simple random sample
mean_attrition_srs = attrition_srs.groupby('RelationshipSatisfaction')\
['Attrition'].mean()


# Print the result
print(mean_attrition_srs)

# Calculate the same thing for the stratified sample
mean_attrition_strat = attrition_strat.groupby('RelationshipSatisfaction')\
['Attrition'].mean()


# Print the result
print(mean_attrition_strat)

# Calculate the same thing for the cluster sample
mean_attrition_clust = attrition_clust.groupby('RelationshipSatisfaction')\
['Attrition'].mean()

# Print the result
print(mean_attrition_clust)
```

```
RelationshipSatisfaction
Low          0.206522
Medium       0.148515
High         0.154684
Very_High    0.148148
Name: Attrition, dtype: float64
RelationshipSatisfaction
Low          0.134328
Medium       0.164179
High         0.160000
Very_High    0.155963
Name: Attrition, dtype: float64
RelationshipSatisfaction
Low          0.144928
Medium       0.078947
High         0.165217
Very_High    0.129630
Name: Attrition, dtype: float64
RelationshipSatisfaction
Low     0.191837
High    0.134694
```

```
Name: Attrition, dtype: float64
```

0.3 CHAPTER 3: Sampling Distributions

Let's test your sampling. In this chapter, you'll discover how to quantify the accuracy of sample statistics using relative errors, and measure variation in your estimates by generating sampling distributions.

0.3.1 Chapter 3.1: Relative error of point estimates

Let's see how the size of the sample affects the accuracy of the point estimates we calculate.

Sample size is number of rows

The sample size, calculated here with the `len` function, is the number of observations, that is, the number of rows in the sample. That's true whichever method we use to create the sample. We'll stick to looking at simple random sampling since it works well in most cases and it's easier to reason about.

Various sample sizes

Let's calculate a population parameter, the mean cup points of the coffees. It's around eighty-two-point-one-five. This is our gold standard to compare against. If we take a sample size of ten, the point estimate of this parameter is wrong by about point-eight-eight. Increasing the sample size to one hundred gets us closer; the estimate is only wrong by about point-three-four. Increasing the sample size further to one thousand brings the estimate to about point-zero-three away from the population parameter. In general, larger sample sizes will give us more accurate results.

Relative errors

For any of these sample sizes, we want to compare the population mean to the sample mean. This is the same code we just saw, but with the numerical sample size replaced with a variable named `sample_size`. The most common metric for assessing the difference between the population and a sample mean is the relative error. The relative error is the absolute difference between the two numbers; that is, we ignore any minus signs, divided by the population mean. Here, we also multiply by one hundred to make it a percentage.

Relative error vs. sample size

Here's a line plot of relative error versus sample size. We see that the relative error decreases as the sample size increases, and beyond that, the plot has other important properties. Firstly, the blue line is really noisy, particularly for small sample sizes. If our sample size is small, the sample mean we calculate can be wildly different by adding one or two more random rows to the sample. Secondly, the amplitude of the line is quite steep, to begin with. When we have a small sample size, adding just a few more samples can give us much better accuracy. Further to the right of the plot, the line is less steep. If we already have a large sample size, adding a few more rows to the sample doesn't bring as much benefit. Finally, at the far right of the plot, where the sample size is the whole population, the relative error decreases to zero.

0.3.2 Exercise 3.1.1

Calculating relative errors

The size of the sample you take affects how accurately the point estimates reflect the corresponding population parameter. For example, when you calculate a sample mean, you want it to be close to the population mean. However, if your sample is too small, this might not be the case.

The most common metric for assessing accuracy is *relative error*. This is the absolute difference between the population parameter and the point estimate, all divided by the population parameter. It is sometimes expressed as a percentage.

Instructions

1. Generate a simple random sample from attrition_pop of fifty rows, setting the seed to 2022.

- Calculate the mean employee `Attrition` in the sample.
- Calculate the relative error between `mean_attrition_srs50` and `mean_attrition_pop` as a *percentage*.

2. Calculate the *relative error percentage* again. This time, use a simple random sample of one hundred rows of `attrition`.

```
# Importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import random

# Importing the course arrays
attrition = pd.read_feather("datasets/attrition.feather")

# Population Attrtion mean
mean_attrition_pop = attrition['Attrition'].mean()
```

```
# Print the result
print(mean_attrition_pop)

# Generate a simple random sample of 50 rows, with seed 2022
attrition_srs50 = attrition.sample(n=50, random_state = 2022)

# Calculate the mean employee attrition in the sample
mean_attrition_srs50 = attrition_srs50['Attrition'].mean()

# Calculate the relative error percentage
rel_error_pct50 = 100 * abs(mean_attrition_pop - mean_attrition_srs50)/mean_attrition_pop

# Print rel_error_pct50
print(rel_error_pct50)

# Generate a simple random sample of 100 rows, with seed 2022
attrition_srs100 = attrition.sample(n=100, random_state = 2022)

# Calculate the mean employee attrition in the sample
mean_attrition_srs100 = attrition_srs100['Attrition'].mean()

# Calculate the relative error percentage
rel_error_pct100 = 100 * abs(mean_attrition_pop - mean_attrition_srs100)/mean_attrition_pop

# Print rel_error_pct100
print(rel_error_pct100)
```

```
0.16122448979591836
62.78481012658227
6.962025316455695
```

0.3.3 Chapter 3.2: Creating a sampling distribution

We just saw how point estimates like the sample mean will vary depending on which rows end up in the sample.

Same code, different answer

For example, this same code to calculate the mean cup points from a simple random sample of thirty coffees gives a slightly different answer each time. Let's try to visualize and quantify this variation.

Same code, 1000 times

A for loop lets us run the same code many times. It's especially useful for situations like this where the result contains some randomness. We start by creating an empty list to store the means. Then, we set up the for loop to repeatedly sample 30 coffees from coffee_ratings a total of 1000 times, calculating the mean cup points each time. After each calculation, we append the result, also called a replicate, to the list. Each time the code is run, we get one sample mean, so running the code a thousand times generates a list of one thousand sample means.

Distribution of sample means for size 30

The one thousand sample means form a distribution of sample means. To visualize a distribution, the best plot is often a histogram. Here we can see that most of the results lie between eighty-one and eighty-three, and they roughly follow a bell-shaped curve, like a normal distribution. There's an important piece of jargon we need to know here. A distribution of replicates of sample means, or other point estimates, is known as a sampling distribution.

Different sample sizes

Here are histograms from running the same code again with different sample sizes. When we decrease the original sample size of thirty to six, we can see from the x-values that the range of the results is broader. The bulk of the results now lie between eighty and eighty-four. On the other hand, increasing the sample size to one hundred and fifty results in a much narrower range. Now most of the results are between eighty-one-point-eight and eighty-two-point-six. As we saw previously, bigger sample sizes give us more accurate results. By replicating the sampling many times, as we've done here, we can quantify that accuracy.

0.3.4 Exercise 3.2.1

Replicating samples

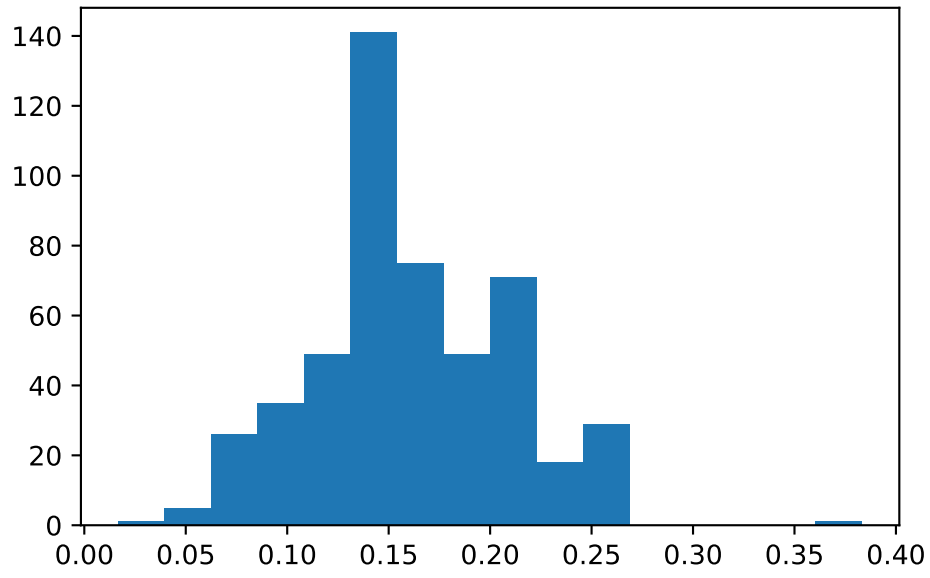
When you calculate a point estimate such as a sample mean, the value you calculate depends on the rows that were included in the sample. That means that there is some randomness in the answer. In order to quantify the variation caused by this randomness, you can create many samples and calculate the sample mean (or another statistic) for each sample.

Instructions

1. Replicate the provided code so that it runs 500 times. Assign the resulting list of sample means to `mean_attritions`.
2. Draw a histogram of the `mean_attritions` list with 16 bins.

```
# Importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import random

# Importing the course arrays
attrition = pd.read_feather("datasets/attrition.feather")

# Create an empty list
mean_attritions = []
# Loop 500 times to create 500 sample means
for i in range(500):
    mean_attritions.append(
        attrition.sample(n=60)['Attrition'].mean()
    )

# Print out the first few entries of the list
print(mean_attritions[0:5])

# Create a histogram of the 500 sample means
plt.hist(mean_attritions, bins=16)
plt.show()
```

```
[0.13333333333333333, 0.26666666666666666, 0.11666666666666667, 0.15, 0.11666666666666667]
```



0.3.5 Chapter 3.3: Approximate sampling distributions

In the last exercise, we saw that while increasing the number of replicates didn't affect the relative error of the sample means; it did result in a more consistent shape to the distribution.

4 dice

Let's consider the case of four six-sided dice rolls. We can generate all possible combinations of rolls using the `expand_grid` function, which is defined in the pandas documentation, and uses the `itertools` package. There are six to the power four, or one-thousand-two-hundred-ninety-six possible dice roll combinations.

Mean roll

Let's consider the mean of the four rolls by adding a column to our DataFrame called `mean_roll`. `mean_roll` ranges from 1, when four ones are rolled, to 6, when four sixes are rolled.

Exact sampling distribution

Since the mean roll takes discrete values instead of continuous values, the best way to see the distribution of mean_roll is to draw a bar plot. First, we convert mean_roll to a categorical by setting its type to category. We are interested in the counts of each value, so we use dot-`value_counts`, passing the sort equals False argument. This ensures the x-axis ranges from one to six instead of sorting the bars by frequency. Chaining `.plot` to `value_counts`, and setting kind to `"bar"`, produces a bar plot of the mean roll distribution. This is the exact sampling distribution of the mean roll because it contains every single combination of die rolls.

The number of outcomes increases fast

If we increase the number of dice in our scenario, the number of possible outcomes increases by a factor of six each time. These values can be shown by creating a DataFrame with two columns: `n_dice`, ranging from 1 to 100, and `n_outcomes`, which is the number of possible outcomes, calculated using six to the power of the number of dice. With just one hundred dice, the number of outcomes is about the same as the number of atoms in the universe: six-point-five times ten to the seventy-seventh power. Long before you start dealing with big datasets, it becomes computationally impossible to calculate the exact sampling distribution. That means we need to rely on approximations.

Simulating the mean of four dice rolls

We can generate a sample mean of four dice rolls using NumPy's `random.choice` method, specifying size as four. This will randomly choose values from a specified list, in this case, four values from the numbers one to six, which is created using a range from one to seven wrapped in the list function. Notice that we set `replace` equals `True` because we can roll the same number several times.

Simulating the mean of four dice rolls

Then we use a for loop to generate lots of sample means, in this case, one thousand. We again use the `.append` method to populate the sample means list with our simulated sample means. The output contains a sampling of many of the same values we saw with the exact sampling distribution.

Approximate sampling distribution

Here's a histogram of the approximate sampling distribution of mean rolls. This time, it uses the simulated rather than the exact values. It's known as an approximate sampling distribution. Notice that although it isn't perfect, it's pretty close to the exact sampling distribution. Usually, we don't have access to the whole population, so we can't calculate the exact sampling distribution. However, we can feel relatively confident that using an approximation will provide a good guess as to how the sampling distribution will behave.

0.3.6 Exercise 3.3.1

Exact sampling distribution

To quantify how the point estimate (sample statistic) you are interested in varies, you need to know all the possible values it can take and how often. That is, you need to know its distribution.

The distribution of a sample statistic is called the *sampling distribution*. When we can calculate this exactly, rather than using an approximation, it is known as the *exact sampling distribution*.

Let's take another look at the sampling distribution of dice rolls. This time, we'll look at five eight-sided dice. (These have the numbers one to eight.)

Instructions

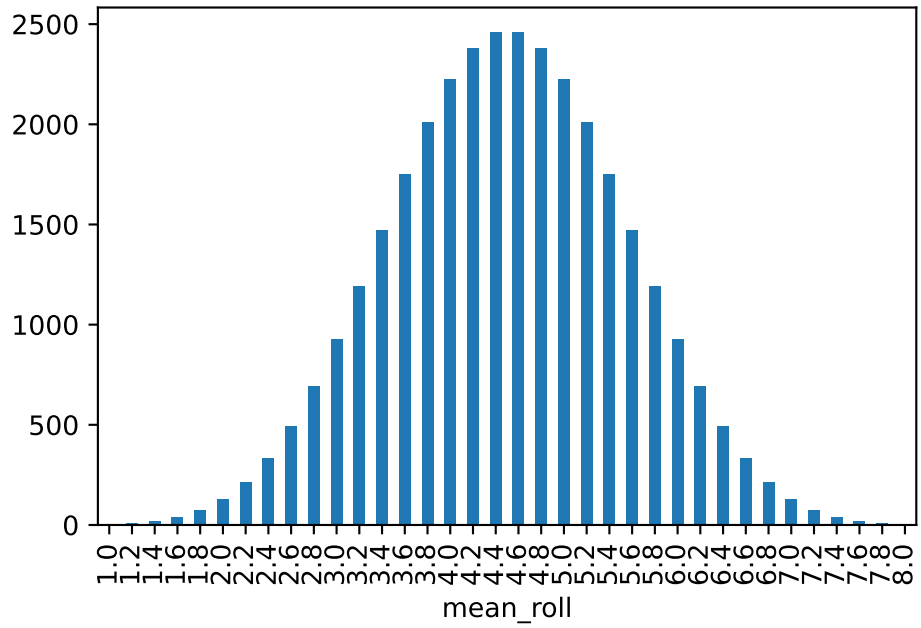
1. Expand a grid representing 5 8-sided dice. That is, create a DataFrame with five columns from a dictionary, named `die1` to `die5`. The rows should contain all possibilities for throwing five dice, each numbered `1` to `8`.
2. Add a column, `mean_roll`, to dice, that contains the mean of the five rolls as a categorical.
3. Create a bar plot of the `mean_roll` categorical column, so it displays the count of each `mean_roll` in increasing order from `1.0` to `8.0`.

```
# Importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import random

# Function to create a grid of all possible combinations
def expand_grid(dictionary):
```

```
    from itertools import product
    return pd.DataFrame([row for row in product(*dictionary.values())], columns=dictionary.keys()

# Expand a grid representing 5 8-sided dice
dice = expand_grid(
    {'die1': range(1, 9),
     'die2': range(1, 9),
     'die3': range(1, 9),
     'die4': range(1, 9),
     'die5': range(1, 9)}
)

# Print the result
print(dice)

# Add a column of mean rolls and convert to a categorical
dice['mean_roll'] = (dice['die1']+ dice['die2']+ dice['die3']+ dice['die4']+ dice['die5'])/5


dice['mean_roll'] = dice['mean_roll'].astype('category')

# Print result
print(dice)

# Draw a bar plot of mean_roll
dice['mean_roll'].value_counts(sort=False).plot(kind='bar')
plt.show()
```

```
       die1  die2  die3  die4  die5
0         1     1     1     1     1
1         1     1     1     1     2
2         1     1     1     1     3
3         1     1     1     1     4
4         1     1     1     1     5
...     ...   ...   ...   ...   ...
32763     8     8     8     8     4
32764     8     8     8     8     5
32765     8     8     8     8     6
32766     8     8     8     8     7
32767     8     8     8     8     8

[32768 rows x 5 columns]
       die1  die2  die3  die4  die5 mean_roll
0         1     1     1     1     1       1.0
1         1     1     1     1     2       1.2
```

```
2          1     1     1     1     3       1.4
3          1     1     1     1     4       1.6
4          1     1     1     1     5       1.8
...      ...   ...   ...   ...   ...       ...
32763      8     8     8     8     4       7.2
32764      8     8     8     8     5       7.4
32765      8     8     8     8     6       7.6
32766      8     8     8     8     7       7.8
32767      8     8     8     8     8       8.0

[32768 rows x 6 columns]
```



0.3.7 Exercise 3.3.2

Generating an approximate sampling distribution

Calculating the exact sampling distribution is only possible in very simple situations. With just five eight-sided dice, the number of possible rolls is `8**5`, which is over thirty thousand. When the dataset is more complicated, for example, where a variable has hundreds or thousands of categories, the number of possible outcomes becomes too difficult to compute exactly.

In this situation, you can calculate an *approximate sampling distribution* by simulating the exact sampling distribution. That is, you can repeat a procedure over and over again to simulate both the sampling process and the sample statistic calculation process.

Instructions

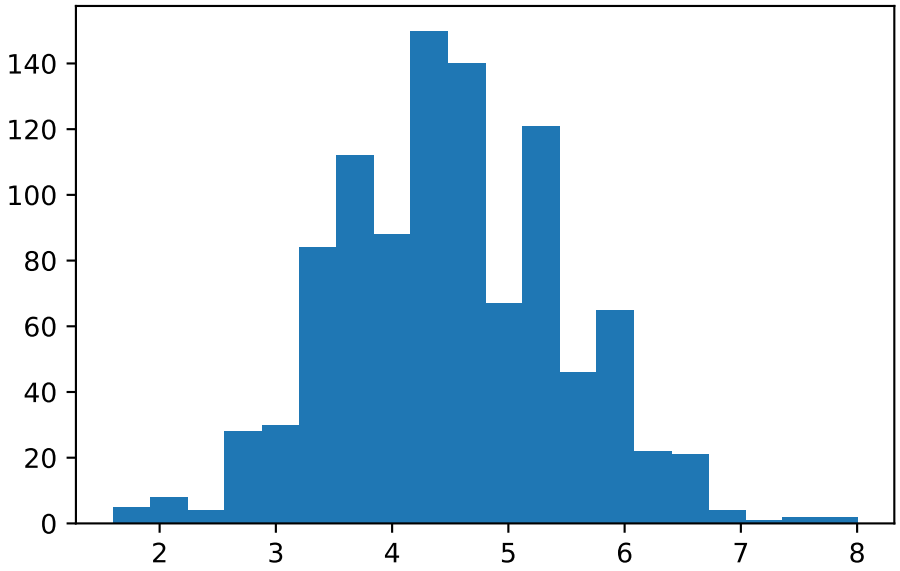
1. Sample one to eight, five times, with replacement. Assign to `five_rolls`.

- Calculate the mean of `five_rolls`.

2. Replicate the sampling code 1000 times, assigning each result to the list `sample_means_1000`.
3. Plot `sample_means_1000` as a histogram with 20 bins.

```
# Importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import random

# Sample one to eight, five times, with replacement
five_rolls = np.random.choice(list(range(1, 9)), size=5, replace=True)

# Print the mean of five_rolls
print(five_rolls.mean())

# Replicate the sampling code 1000 times
sample_means_1000 = []
for i in range(1000):
    sample_means_1000.append(
        np.random.choice(list(range(1, 9)), size=5, replace=True).mean()
    )

# Print the first 10 entries of the result
print(sample_means_1000[0:10])

# Draw a histogram of sample_means_1000 with 20 bins
plt.hist(sample_means_1000, bins=20)
plt.show()
```

```
4.8
[4.6, 5.0, 5.2, 4.2, 3.8, 5.6, 4.8, 4.8, 4.2, 5.2]
```



0.3.8 Chapter 3.4: Standard errors and the Central Limit Theorem

The Gaussian distribution (also known as the normal distribution) plays an important role in statistics. Its distinctive bell-shaped curve has been cropping up throughout this course.

Sampling distribution of mean cup points

Here are approximate sampling distributions of the mean cup points from the coffee dataset. Each histogram shows five thousand replicates, with different sample sizes in each case. Look at the x-axis labels. We already saw how increasing the sample size results in greater accuracy in our estimates of the population parameter, so the width of the distribution shrinks as the sample size increases. When the sample size is five, the x-axis ranges from seventy-six to eighty-six, whereas, for a sample size of three hundred and twenty, the range is from eighty-one-point-six to eighty-two-point-six. Now, look at the shape of each distribution. As the sample size increases, we can see that the shape of the curve gets closer and closer to being a normal distribution. At sample size five, the curve is only a very loose approximation since it isn't very symmetric. By sample size eighty, it is a very reasonable approximation.

Consequences of the central limit theorem

What we just saw is, in essence, what the central limit theorem tells us. The means of independent samples have normal distributions. Then, as the sample size increases, we see two things. The distribution of these averages gets closer to being normal, and the width of this sampling distribution gets narrower.

Population & sampling distribution means

Recall the population parameter of the mean cup points. We've seen this calculation before, and its value is eighty-two-point-one-five. We can also calculate summary statistics on our sampling distributions to see how they compare. For each of our four sampling distributions, if we take the mean of our sample means, we can see that we get values that are pretty close to the population parameter that the sampling distributions are trying to estimate.

Population & sampling distribution standard deviations

Now let's consider the standard deviation of the population cup points. It's about two-point-seven. By comparison, if we take the standard deviation of the sample means from each of the sampling distributions using NumPy, we get much smaller numbers, and they decrease as the sample size increases. Note that when we are calculating a population standard deviation with pandas `.std`, we must specify `ddof` equals zero, as `.std` calculates a sample standard deviation by default. When we are calculating a standard deviation on a sample of the population using NumPy's std function, like in these calculations on the sampling distribution, we must specify a `ddof` of one. So what are these smaller standard deviation values?

Population mean over square root sample size

One other consequence of the central limit theorem is that if we divide the population standard deviation, in this case around 2.7, by the square root of the sample size, we get an estimate of the standard deviation of the sampling distribution for that sample size. It isn't exact because of the randomness involved in the sampling process, but it's pretty close.

Standard error

We just saw the impact of the sample size on the standard deviation of the sampling distribution. This standard deviation of the sampling distribution has a special name: the standard error. It is useful in a variety of contexts, from estimating population standard deviation to setting expectations on what level of variability we would expect from the sampling process.

0.3.9 Exercise 3.4.1

Population & sampling distribution means

One of the useful features of sampling distributions is that you can quantify them. Specifically, you can calculate summary statistics on them. Here, you'll look at the relationship between the mean of the sampling distribution and the population parameter's mean.

Three sampling distributions are provided. For each, the employee attrition dataset was sampled using simple random sampling, then the mean attrition was calculated. This was done 1000 times to get a sampling distribution of mean attritions. One sampling distribution used a sample size of 5 for each replicate, one used 50, and one used 500.

Instructions

1. Calculate the mean of `sampling_distribution_5`, `sampling_distribution_50`, and `sampling_distribution_500` (a mean of sample means).

```
# Importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import random

# Importing the course arrays
attrition = pd.read_feather("datasets/attrition.feather")

# Set a seed for reproducibility
random_seed = 2021

# Create three empty lists to hold the sampling distributions
sampling_distribution_5 = []   # Sample size of 5
sampling_distribution_50 = []  # Sample size of 50
sampling_distribution_500 = [] # Sample size of 500

# Perform biased sampling and calculate mean attrition 1000 times for each sample size
for i in range(1000):
    # Sample size = 5 (heavier weights toward high attrition)
    sampling_distribution_5.append(
        attrition.sample(n=5, random_state=random_seed + i)['Attrition'].mean()
    )

    # Sample size = 50 (bias reduces as sample size increases)
    sampling_distribution_50.append(
        attrition.sample(n=50, random_state=random_seed + i)['Attrition'].mean()
    )

    # Sample size = 500 (approaching unbiased mean)
    sampling_distribution_500.append(
        attrition.sample(n=500, random_state=random_seed + i)['Attrition'].mean()
    )

# Optional: Convert the sampling distributions to DataFrame for analysis
sampling_df = pd.DataFrame({
    'Sample_Size_5': sampling_distribution_5,
    'Sample_Size_50': sampling_distribution_50,
    'Sample_Size_500': sampling_distribution_500
})
```

```
# Calculate the mean of the mean attritions for each sampling distribution
mean_of_means_5 = np.mean(sampling_distribution_5)
mean_of_means_50 = np.mean(sampling_distribution_50)
mean_of_means_500 = np.mean(sampling_distribution_500)

# Print the results
print(mean_of_means_5)
print(mean_of_means_50)
print(mean_of_means_500)
```

```
0.155
0.15998
0.160622
```

> **Note**
>
> *Even for small sample sizes, the mean of the sampling distribution is a good approximation of the population mean.*

0.3.10 Exercise 3.4.2

Population & sampling distribution variation

You just calculated the mean of the sampling distribution and saw how it is an estimate of the corresponding population parameter. Similarly, as a result of the central limit theorem, the standard deviation of the sampling distribution has an interesting relationship with the population parameter's standard deviation and the sample size.

Instructions

1. Calculate the standard deviation of `sampling_distribution_5`, `sampling_distribution_50`, and `sampling_distribution_500` (a standard deviation of sample means).

```
# Importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import random

# Importing the course arrays
attrition = pd.read_feather("datasets/attrition.feather")

# Set a seed for reproducibility
random_seed = 2021
```

```python
# Create three empty lists to hold the sampling distributions
sampling_distribution_5 = []   # Sample size of 5
sampling_distribution_50 = []  # Sample size of 50
sampling_distribution_500 = [] # Sample size of 500

# Perform biased sampling and calculate mean attrition 1000 times for each sample size
for i in range(1000):
    # Sample size = 5 (heavier weights toward high attrition)
    sampling_distribution_5.append(
        attrition.sample(n=5, random_state=random_seed + i)['Attrition'].mean()
    )

    # Sample size = 50 (bias reduces as sample size increases)
    sampling_distribution_50.append(
        attrition.sample(n=50, random_state=random_seed + i)['Attrition'].mean()
    )

    # Sample size = 500 (approaching unbiased mean)
    sampling_distribution_500.append(
        attrition.sample(n=500, random_state=random_seed + i)['Attrition'].mean()
    )

# Optional: Convert the sampling distributions to DataFrame for analysis
sampling_df = pd.DataFrame({
    'Sample_Size_5': sampling_distribution_5,
    'Sample_Size_50': sampling_distribution_50,
    'Sample_Size_500': sampling_distribution_500
})

# Calculate the std. dev. of the mean attritions for each sampling distribution
sd_of_means_5 = np.std(sampling_distribution_5, ddof = 1)
sd_of_means_50 = np.std(sampling_distribution_50, ddof = 1)
sd_of_means_500 = np.std(sampling_distribution_500, ddof = 1)

# Print the results
print(sd_of_means_5)
print(sd_of_means_50)
print(sd_of_means_500)
```

```
0.15244093360458746
0.04970785119546479
0.014243454356018837
```

> **Note**
>
> *The amount of variation in the sampling distribution is related to the amount of variation in the population and the sample size. This is another consequence of the Central Limit Theorem.*

0.4 CHAPTER 4: Bootstrap Distributions

You'll get to grips with resampling to perform bootstrapping and estimate variation in an unknown population. You'll learn the difference between sampling distributions and bootstrap distributions using resampling.

0.4.1 Chapter 4.1: Introduction to bootstrapping

So far, we've mostly focused on the idea of sampling without replacement.

With or without

Sampling without replacement is like dealing a pack of cards. When we deal the ace of spades to one player, we can't then deal the ace of spades to another player. Sampling with replacement is like rolling dice. If we roll a six, we can still get a six on the next roll. Sampling with replacement is sometimes called resampling. We'll use the terms interchangeably.

Simple random sampling without replacement

If we take a simple random sample without replacement, each row of the dataset, or each type of coffee, can only appear once in the sample.

Simple random sampling with replacement

If we sample with replacement, it means that each row of the dataset, or each coffee, can be sampled multiple times.

Why sample with replacement?

So far, we've been treating the `coffee_ratings` dataset as the population of all coffees. Of course, it doesn't include every coffee in the world, so we could treat the coffee dataset as just being a big sample of coffees. To imagine what the whole population is like, we need to approximate the other coffees that aren't in the dataset. Each of the coffees in the sample dataset will have properties that are representative of the coffees that we don't have. Resampling lets us use the existing coffees to approximate those other theoretical coffees.

Coffee data preparation

To keep it simple, let's focus on three columns of the coffee dataset. To make it easier to see which rows ended up in the sample, we'll add a row index column called index using the `reset_index` method.

Resampling with `.sample()`

To sample with replacement, we call sample as usual but set the `replace` argument to `True`. Setting `frac` to `1` produces a sample of the same size as the original dataset.

Repeated coffees

Counting the values of the index column shows how many times each coffee ended up in the resampled dataset. Some coffees were sampled four or five times.

Missing coffees

That means that some coffees didn't end up in the resample. By taking the number of distinct index values in the resampled dataset, using `len` on `drop_duplicates`, we see that eight hundred and sixty-eight different coffees were included. By comparing this number with the total number of coffees, we can see that four hundred and seventy coffees weren't included in the resample.

Bootstrapping

We're going to use resampling for a technique called bootstrapping. In some sense, bootstrapping is the opposite of sampling from a population. With sampling, we treat the dataset as the population and move to a smaller sample. With bootstrapping, we treat the dataset as a sample and use it to build up a theoretical population. A use case of bootstrapping is to try to understand the variability due to sampling. This is important in cases where we aren't able to sample the population multiple times to create a sampling distribution.

Bootstrapping process

The bootstrapping process has three steps. First, randomly sample with replacement to get a resample the same size as the original dataset. Then, calculate a statistic, such as a mean of one of the columns. Note that the mean isn't always the choice here and bootstrapping allows for complex statistics to be computed, too. Then, replicate this many times to get lots of these bootstrap statistics. Earlier in the course, we did something similar. We took a simple random sample, then calculated a summary statistic, then repeated those two steps to form a sampling distribution. This time, when we've used resampling instead of sampling, we get a bootstrap distribution.

Bootstrapping coffee mean flavor

The resampling step uses the code we just saw: calling sample with `frac` set to `one` and `replace` set to `True`. Calculating a bootstrap statistic can be done with mean from NumPy. In this case, we're calculating the mean flavor score. To repeat steps one and two one thousand times, we can wrap the code in a for loop and append the statistics to a list.

Bootstrap distribution histogram

Here's a histogram of the bootstrap distribution of the sample mean. Notice that it is close to following a normal distribution.

0.4.2 Exercise 4.1.1

Generating a bootstrap distribution

The process for generating a bootstrap distribution is similar to the process for generating a sampling distribution; only the first step is different.

To make a sampling distribution, you start with the population and sample without replacement. To make a bootstrap distribution, you start with a sample and sample that with replacement. After that, the steps are the same: calculate the summary statistic that you are interested in on that sample/resample, then replicate the process many times. In each case, you can visualize the distribution with a histogram.

Here, `spotify_sample` is a subset of the `spotify` dataset. To make it easier to see how resampling works, a row index column called `'index'` has been added, and only the artist name, song name, and `danceability` columns have been included.

Instructions

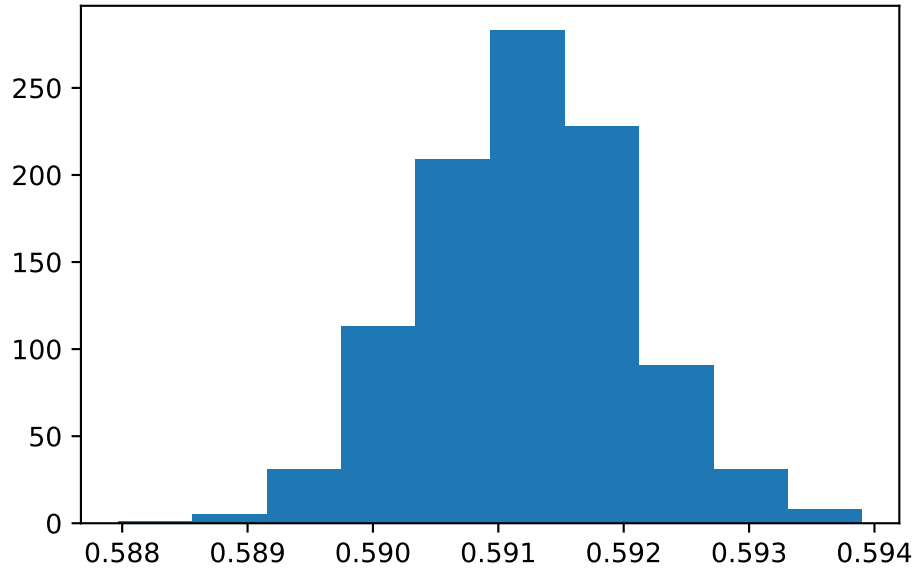
1. Generate a single bootstrap resample from `spotify_sample`.
2. Calculate the mean of the `danceability` column of `spotify_1_resample` using `numpy`.
3. Replicate the expression provided 1000 times.
4. Create a bootstrap distribution by drawing a histogram of `mean_danceability_1000`.

```
# Importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import random

# Importing the course array
spotify = pd.read_feather("datasets/spotify_2000_2020.feather")

# Subset of spotify sample to use
```

```
spotify_sample = spotify.sample(n=41656)[['artists', 'name', 'danceability']]
spotify_sample['index'] = spotify_sample.index

# Reorder columns to make 'index' the first column
spotify_sample = spotify_sample[['index', 'artists', 'name', 'danceability']]

# Generate 1 bootstrap resample
spotify_1_resample = spotify_sample.sample(frac=1, replace = True)

# Print the resample
print(spotify_1_resample)

# Calculate of the danceability column of spotify_1_resample
mean_danceability_1 = np.mean(spotify_1_resample['danceability'])

# Print the result
print(mean_danceability_1)

# Replicate this 1000 times
mean_danceability_1000 = []
for i in range(1000):
    mean_danceability_1000.append(
        np.mean(spotify_sample.sample(frac=1, replace=True)['danceability'])
    )

# Print the result
print(mean_danceability_1000)

# Draw a histogram of the resample means
plt.hist(mean_danceability_1000)
plt.show()
```

```
       index                       artists  \
38767  38767              ['Current Joys']
32960  32960    ['ODESZA', 'Leon Bridges']
1847    1847  ['Billie Eilish', 'TroyBoi']
2297    2297       ['Death Cab for Cutie']
27589  27589                 ['Daft Punk']
...      ...                           ...
6529    6529             ['Box Car Racer']
31310  31310           ['Reckless Kelly']
14976  14976  ['Benny Benassi', 'The Biz']
16604  16604               ['Lil Loaded']
1350    1350                  ['Sun Rai']
```

```
                                        name  danceability
38767                          Televisions          0.522
32960  Across The Room (feat. Leon Bridges)          0.566
1847                   MyBoi - TroyBoi Remix          0.879
2297                   A Movie Script Ending          0.505
27589                              Robot Rock          0.590
...                                       ...            ...
6529                        Watch The World          0.472
31310             Crazy Eddie's Last Hurrah          0.530
14976             Satisfaction - Radio Edit          0.854
16604                               Gang Unit          0.912
1350                     San Francisco Street          0.877

[41656 rows x 4 columns]
0.5911335509890532
[0.5915571250240061, 0.5923078548108316, 0.5920595472440945, 0.5905520789322066, 0.59064877808719
```



0.4.3 Chapter 4.2: Comparing sampling and bootstrap distributions

Coffee focused subset

we took a focused subset of the coffee dataset. Here's a five hundred row sample from it.

The bootstrap of mean coffee flavors

Here, we generate a bootstrap distribution of the mean coffee flavor scores from that sample. `.sample` generates a resample, `np.mean` calculates the statistic, and the for loop with `.append` repeats these steps to produce a distribution of bootstrap statistics.

Mean flavor bootstrap distribution

Observing the histogram of the bootstrap distribution, which is close to a normal distribution.

Sample, bootstrap distribution, population means

Here's the mean flavor score from the original sample. In the bootstrap distribution, each value is an estimate of the mean flavor score. Recall that each of these values corresponds to one potential sample mean from the theoretical population. If we take the mean of those means, we get our best guess of the population mean. The two values are really close. However, there's a problem. The true population mean is actually a little different.

Interpreting the means

The behavior that you just saw is typical. The bootstrap distribution mean is usually almost identical to the original sample mean. However, that is not often a good thing. If the original sample wasn't closely representative of the population, then the bootstrap distribution mean won't be a good estimate of the population mean. Bootstrapping cannot correct any potential biases due to differences between the sample and the population.

Sample `sd` vs. bootstrap distribution `sd`

While we do have that limitation in estimating the population mean, one great thing about distributions is that we can also quantify variation. The standard deviation of the sample flavors is around `0.354`. Recall that pandas `.std` calculates a sample standard deviation by default. If we calculate the standard deviation of the bootstrap distribution, specifying a `ddof` of one, then we get a completely different number. So what's going on here?

Sample, bootstrap dist'n, pop'n standard deviations

Remember that one goal of bootstrapping is to quantify what variability we might expect in our sample statistic as we go from one sample to another. Recall that this quantity is called the standard error as measured by the standard deviation of the sampling distribution of that statistic. The standard deviation of the bootstrap means can be used as a way to estimate this measure of uncertainty. If we multiply that standard error by the square root of the sample size, we get an estimate of the standard deviation in the original population. Our estimate of the standard deviation is around point-three-five-three. The true standard deviation is around point-three-four-one, so our estimate is pretty close. In fact, it is closer than just using the sample standard deviation alone.

Interpreting the standard errors

To recap, the estimated standard error is the standard deviation of the bootstrap distribution values for our statistic of interest. This estimated standard error times the square root of the sample size gives a really good estimate of the standard deviation of the population. That is, although bootstrapping was poor at estimating the population mean, it is generally great for estimating the population standard deviation.

0.4.4 Exercise 4.2.1

Sampling distribution vs. bootstrap distribution

The sampling distribution and bootstrap distribution are closely linked. In situations where you can repeatedly sample from a population (these occasions are rare), it's helpful to generate both the sampling distribution and the bootstrap distribution, one after the other, to see how they are related.

Here, the statistic you are interested in is the mean `popularity` score of the songs.

Instructions

1. Generate a sampling distribution of 2000 replicates.

- Sample 500 rows of the population without replacement and calculate the mean `popularity`.

2. Generate a bootstrap distribution of 2000 replicates.

- Sample 500 rows of the sample with replacement and calculate the mean `popularity`.

```
# Importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import random

# Importing the course array
spotify = pd.read_feather("datasets/spotify_2000_2020.feather")

spotify_sample = spotify.sample(n=500)

mean_popularity_2000_samp = []

# Generate a sampling distribution of 2000 replicates
for i in range(2000):
    mean_popularity_2000_samp.append(
        # Sample 500 rows and calculate the mean popularity
        spotify.sample(n=500)['popularity'].mean()
```

```
    )

# Print the sampling distribution results
print(mean_popularity_2000_samp)

mean_popularity_2000_boot = []

# Generate a bootstrap distribution of 2000 replicates
for i in range(2000):
    mean_popularity_2000_boot.append(
        # Resample 500 rows and calculate the mean popularity
        np.mean(spotify_sample.sample(frac=1, replace=True)['popularity'])
    )

# Print the bootstrap distribution results
print(mean_popularity_2000_boot)
```

```
[54.526, 54.586, 54.742, 54.964, 54.646, 53.94, 54.454, 54.884, 55.182, 54.766, 54.578, 54.632, 5
[54.788, 54.698, 55.536, 54.768, 55.348, 54.426, 54.722, 55.488, 54.966, 54.476, 55.644, 55.768,
```

> **Note**
>
> *The sampling distribution and bootstrap distribution are closely related, and so is the code to generate them.*

0.4.5 Exercise 4.2.2

Compare sampling and bootstrap means

To make calculation easier, distributions similar to those calculated from the previous exercise have been included, this time using a sample size of 5000.

spotify_population, spotify_sample, sampling_distribution, and bootstrap_distribution are available; pandas and numpy are loaded with their usual aliases.

Instructions

1. Calculate the mean `popularity` in 4 ways:

- Population: from `spotify`, take the mean of `popularity`.
- Sample: from `spotify_sample`, take the mean of `popularity`.
- Sampling distribution: from `sampling_distribution`, take its mean.
- Bootstrap distribution: from ‘`bootstrap_distribution`, take its mean.

```python
# Importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import random

# Importing the course array
spotify = pd.read_feather("datasets/spotify_2000_2020.feather")

spotify_sample = spotify.sample(n=500)

mean_popularity_2000_samp = []

# Generate a sampling distribution of 2000 replicates
for i in range(2000):
    mean_popularity_2000_samp.append(
        # Sample 500 rows and calculate the mean popularity
        spotify.sample(n=500)['popularity'].mean()
    )

# The sampling distribution results
sampling_distribution = mean_popularity_2000_samp

mean_popularity_2000_boot = []

# Generate a bootstrap distribution of 2000 replicates
for i in range(2000):
    mean_popularity_2000_boot.append(
        # Resample 500 rows and calculate the mean popularity
        np.mean(spotify_sample.sample(frac=1, replace=True)['popularity'])
    )

# The bootstrap distribution results
bootstrap_distribution = mean_popularity_2000_boot

# Calculate the population mean popularity
pop_mean = spotify['popularity'].mean()

# Calculate the original sample mean popularity
samp_mean = spotify_sample['popularity'].mean()

# Calculate the sampling dist'n estimate of mean popularity
samp_distn_mean = np.mean(sampling_distribution)

# Calculate the bootstrap dist'n estimate of mean popularity
```

```
boot_distn_mean = np.mean(bootstrap_distribution)

# Print the means
print([pop_mean, samp_mean, samp_distn_mean, boot_distn_mean])
```

```
[54.837142308430955, 55.512, 54.84017, 55.506617]
```

> **Note**
>
> *The sampling distribution mean can be used to estimate the population mean, but that is not the case with the bootstrap distribution.*

0.4.6 Exercise 4.2.3

Compare sampling and bootstrap standard deviations

In the same way that you looked at how the sampling distribution and bootstrap distribution could be used to estimate the population mean, you'll now take a look at how they can be used to estimate variation, or more specifically, the standard deviation, in the population.

Recall that the sample size is 5000.

Instructions

Calculate the standard deviation of `popularity` in 4 ways. - Population: from `spotify`, take the standard deviation of `popularity`. - Original sample: from `spotify_sample`, take the standard deviation of `popularity`. - Sampling distribution: from `sampling_distribution`, take its standard deviation and multiply by the square root of the sample size (5000). - Bootstrap distribution: from `bootstrap_distribution`, take its standard deviation and multiply by the square root of the sample size.

```
# Importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import random

# Importing the course array
spotify = pd.read_feather("datasets/spotify_2000_2020.feather")

spotify_sample = spotify.sample(n=5000, random_state=2022)

mean_popularity_2000_samp = []
```

```
# Generate a sampling distribution of 2000 replicates
for i in range(2000):
    mean_popularity_2000_samp.append(
        # Sample 500 rows and calculate the mean popularity
        spotify.sample(n=5000)['popularity'].mean()
    )

# The sampling distribution results
sampling_distribution = mean_popularity_2000_samp

mean_popularity_2000_boot = []

# Generate a bootstrap distribution of 2000 replicates
for i in range(2000):
    mean_popularity_2000_boot.append(
        # Resample 500 rows and calculate the mean popularity
        np.mean(spotify_sample.sample(frac=1, replace=True)['popularity'])
    )

# The bootstrap distribution results
bootstrap_distribution = mean_popularity_2000_boot

# Calculate the population std dev popularity
pop_sd = spotify['popularity'].std(ddof=0)

# Calculate the original sample std dev popularity
samp_sd = spotify_sample['popularity'].std(ddof=1)

# Calculate the sampling dist'n estimate of std dev popularity
samp_distn_sd = np.std(sampling_distribution, ddof=1) * np.sqrt(5000)

# Calculate the bootstrap dist'n estimate of std dev popularity
boot_distn_sd = np.std(bootstrap_distribution, ddof=1) * np.sqrt(5000)

# Print the standard deviations
print([pop_sd, samp_sd, samp_distn_sd, boot_distn_sd])
```

```
[10.880065274257536, 10.975581356685552, 10.118155913776867, 11.119889427921246]
```

0.4.7 Chapter 4.3: Confidence intervals

In the last few exercises, you looked at relationships between the sampling distribution and the bootstrap distribution.

One way to quantify these distributions is the idea of "values within one standard deviation of the mean", which gives a good sense of where most of the values in a distribution lie. In this final lesson, we'll formalize the idea of values close to a statistic by defining the term "confidence interval".

Predicting the weather

Consider meteorologists predicting weather in one of the world's most unpredictable regions - the northern Great Plains of the US and Canada. Rapid City, South Dakota was ranked as the least predictable of the 120 US cities with a National Weather Service forecast office. Suppose we've taken a job as a meteorologist at a news station in Rapid City. Our job is to predict tomorrow's high temperature.

Our weather prediction

We analyze the weather data using the best forecasting tools available to us and predict a high temperature of 47 degrees Fahrenheit. In this case, 47 degrees is our point estimate. Since the weather is variable, and many South Dakotans will plan their day tomorrow based on our forecast, we'd instead like to present a range of plausible values for the high temperature. On our weather show, we report that the high temperature will be between forty and fifty-four degrees tomorrow.

We just reported a confidence interval!

This prediction of forty to fifty-four degrees can be thought of as a confidence interval for the unknown quantity of tomorrow's high temperature. Although we can't be sure of the exact temperature, we are confident that it will be in that range. These results are often written as the point estimate followed by the confidence interval's lower and upper bounds in parentheses or square brackets. When the confidence interval is symmetric around the point estimate, we can represent it as the point estimate plus or minus the margin of error, in this case, seven degrees.

Bootstrap distribution of mean flavor

Here's the bootstrap distribution of the mean flavor from the coffee dataset.

Mean of the resamples

We can calculate the mean of these resampled mean flavors.

Mean plus or minus one standard deviation

If we create a confidence interval by adding and subtracting one standard deviation from the mean, we see that there are lots of values in the bootstrap distribution outside of this one standard deviation confidence interval.

Quantile method for confidence intervals

If we want to include ninety-five percent of the values in the confidence interval, we can use quantiles. Recall that quantiles split distributions into sections containing a particular proportion of the total data. To get the middle ninety-five percent of values, we go from the point-zero-two-five quantile to the point-nine-seven-five quantile since the difference between those two numbers is point-nine-five. To calculate the lower and upper bounds for this confidence interval, we call quantile from NumPy, passing the distribution values and the quantile values to use. The confidence interval is from around seven-point-four-eight to seven-point-five-four.

Inverse cumulative distribution function

There is a second method to calculate confidence intervals. To understand it, we need to be familiar with the normal distribution's inverse cumulative distribution function. The bell curve we've seen before is the probability density function or PDF. Using calculus, if we integrate this, we get the cumulative distribution function or CDF. If we flip the x and y axes, we get the inverse `CDF`. We can use `scipy.stats` and call `norm.ppf` to get the inverse `CDF`. It takes a quantile between zero and one and returns the values of the normal distribution for that quantile. The parameters of `loc` and `scale` are set to 0 and 1 by default, corresponding to the standard normal distribution. Notice that the values corresponding to point-zero-two-five and point-nine-seven-five are about minus and plus two for the standard normal distribution.

Standard error method for confidence interval

This second method for calculating a confidence interval is called the standard error method. First, we calculate the point estimate, which is the mean of the bootstrap distribution, and the standard error, which is estimated by the standard deviation of the bootstrap distribution. Then we call `norm.ppf` to get the inverse `CDF` of the normal distribution with the same mean and standard deviation as the bootstrap distribution. Again, the confidence interval is from seven-point-four-eight to seven-point-five-four, though the numbers differ slightly from last time since our bootstrap distribution isn't perfectly normal.

0.4.8 Exercise 4.3.1

0.4.8.1 Calculating confidence intervals

You have learned about two methods for calculating confidence intervals: *the quantile method* and *the standard error method*. The standard error method involves using the inverse cumulative distribution function (inverse CDF) of the normal distribution to calculate confidence intervals. In this exercise, you'll perform these two methods on the Spotify data.

0.4.8.2 Instructions

1. Generate a 95% confidence interval using the quantile method on the bootstrap distribution, setting the `0.025` quantile as `lower_quant` and the `0.975` quantile as `upper_quant`.
2. Generate a 95% confidence interval using the standard error method from the bootstrap distribution.

- Calculate `point_estimate` as the mean of `bootstrap_distribution`, and `standard_error` as the standard deviation of `bootstrap_distribution`.
- Calculate `lower_se` as the `0.025` quantile of an inv. CDF from a normal distribution with mean `point_estimate` and standard deviation `standard_error`.
- Calculate `upper_se` as the `0.975` quantile of that same inv. CDF.

```
# Importing libraries
import pandas as pd
import numpy as np
from scipy.stats import norm

# Importing the course array
spotify = pd.read_feather("datasets/spotify_2000_2020.feather")

spotify_sample = spotify.sample(n=5000, random_state=2022)

mean_popularity_2000_boot = []

# Generate a bootstrap distribution of 2000 replicates
for i in range(2000):
    mean_popularity_2000_boot.append(
        # Resample 500 rows and calculate the mean popularity
        np.mean(spotify_sample.sample(frac=1, replace=True)['popularity'])
    )

# The bootstrap distribution results
bootstrap_distribution = mean_popularity_2000_boot

# Generate a 95% confidence interval using the quantile method
lower_quant = np.quantile(bootstrap_distribution, 0.025)
upper_quant = np.quantile(bootstrap_distribution, 0.975)

# Print quantile method confidence interval
print((lower_quant, upper_quant))

# Find the mean and std dev of the bootstrap distribution
point_estimate = np.mean(bootstrap_distribution)
standard_error = np.std(bootstrap_distribution, ddof=1)
```

```
# Find the lower limit of the confidence interval
lower_se = norm.ppf(0.025, loc=point_estimate, scale=standard_error)

# Find the upper limit of the confidence interval
upper_se = norm.ppf(0.975, loc=point_estimate, scale=standard_error)

# Print standard error method confidence interval
print((lower_se, upper_se))
```

```
(54.47479, 55.08541)
(54.471152438068415, 55.07709576193159)
```

0.5 Reference

Sampling in Python in Intermediate Python Course for Associate Data Scientist in Python Carrer Track in DataCamp Inc by James Chapman.