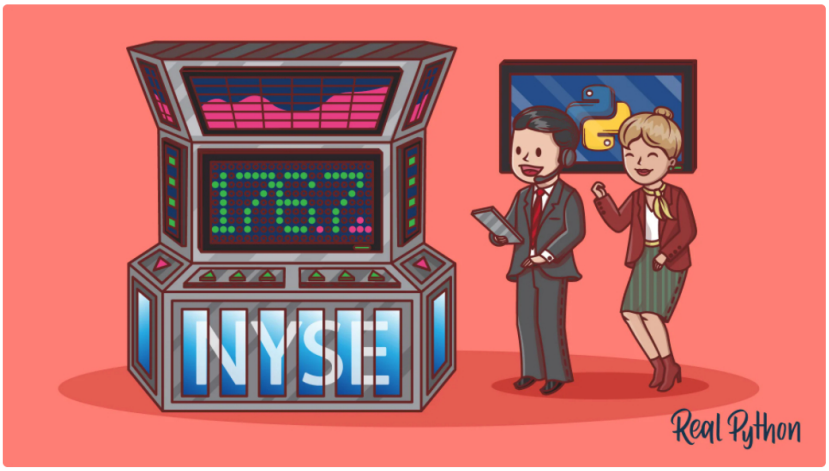


How to Round Numbers in Python

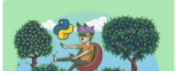
by David Amos Oct 08, 2018 18 Comments best-practices intermediate python

Table of Contents

It's the era of big data, and every day more and more business are trying to leverage their data to make informed decisions. Many businesses are turning to Python's powerful data science ecosystem to analyze their data, as evidenced by Python's rising popularity in the data science realm.

One thing every data science practitioner must keep in mind is how a dataset may be biased. Drawing conclusions from biased data can lead to costly mistakes.

There are many ways bias can creep into a dataset. If you've studied some statistics, you're probably familiar with terms like reporting bias, selection bias and sampling bias. There is another type of bias that plays an important role when you are dealing with numeric data:



another type of bias that plays an important role when you are dealing with numeric data: rounding bias.

» Click here to get the first lesson

In this article, you will learn:

- Why the way you round numbers is important
- How to round a number according to various rounding strategies, and how to implement each method in pure Python
- How rounding affects data, and which rounding strategy minimizes this effect
- How to round numbers in NumPy arrays and Pandas DataFrames
- When to apply different rounding strategies

🎓 **Take the Quiz:** Test your knowledge with our interactive "Rounding Numbers in Python" quiz. Upon completion you will receive a score so you can track your learning progress over time:

Take the Quiz »

This article is not a treatise on numeric precision in computing, although we will touch briefly on the subject. Only a familiarity with the fundamentals of Python is necessary, and the math involved here should feel comfortable to anyone familiar with the equivalent of high school algebra.

Let's start by looking at Python's built-in rounding mechanism.

Python's Built-in `round()` Function

Python has a built-in `round()` function that takes two numeric arguments, `n` and `ndigits`, and returns the number `n` rounded to `ndigits`. The `ndigits` argument defaults to zero, so leaving it out results in a number rounded to an integer. As you'll see, `round()` may not work quite as you expect.

The way most people are taught to round a number goes something like this:

> Round the number `n` to `p` decimal places by first shifting the decimal point in `n` by `p` places by multiplying `n` by 10^p (10 raised to the `p`th power) to get a new number `m`.
>
> Then look at the digit `d` in the first decimal place of `m`. If `d` is less than 5, round `m` down to the nearest integer. Otherwise, round `m` up.
>
> Finally, shift the decimal point back `p` places by dividing `m` by 10^p.

It's a straightforward algorithm! For example, the number `2.5` rounded to the nearest whole number is `3`. The number `1.64` rounded to one decimal place is `1.6`.

Now open up an interpreter session and round `2.5` to the nearest whole number using Python's built-in `round()` function:

```
Python >>>

>>> round(2.5)
2
```

Gasp!

How does `round()` handle the number `1.5`?

```
Python >>>

>>> round(1.5)
2
```

So, `round()` rounds `1.5` up to `2`, and `2.5` down to `2`!

Before you go raising an issue on the Python bug tracker, let me assure you that `round(2.5)` is supposed to return `2`. There is a good reason why `round()` behaves the way it does.

In this article, you'll learn that there are more ways to round a number than you might expect, each with unique advantages and disadvantages. `round()` behaves according to a particular rounding strategy—which may or may not be the one you need for a given situation.

You might be wondering, "Can the way I round numbers really have that much of an impact?" Let's take a look at just how extreme the effects of rounding can be.

How Much Impact Can Rounding Have?

Suppose you have an incredibly lucky day and find $100 on the ground. Rather than spending all your money at once, you decide to play it smart and invest your money by buying some shares of different stocks.

The value of a stock depends on supply and demand. The more people there are who want to buy a stock, the more value that stock has, and vice versa. In high volume stock markets, the value of a particular stock can fluctuate on a second-by-second basis.

Let's run a little experiment. We'll pretend the overall value of the stocks you purchased fluctuates by some small random number each second, say between $0.05 and -$0.05. This fluctuation may not necessarily be a nice value with only two decimal places. For example, the overall value may increase by $0.031286 one second and decrease the next second by $0.028476.

You don't want to keep track of your value to the fifth or sixth decimal place, so you decide to chop everything off after the third decimal place. In rounding jargon, this is called **truncating** the number to the third decimal place. There's some error to be expected here, but by keeping three decimal places, this error couldn't be substantial. Right?

To run our experiment using Python, let's start by writing a `truncate()` function that truncates a number to three decimal places:

Python >>>

```
>>> def truncate(n):
...     return int(n * 1000) / 1000
```

The `truncate()` function works by first shifting the decimal point in the number `n` three places to the right by multiplying `n` by `1000`. The integer part of this new number is taken with `int()`. Finally, the decimal point is shifted three places back to the left by dividing `n` by `1000`.

Next, let's define the initial parameters of the simulation. You'll need two variables: one to keep track of the actual value of your stocks after the simulation is complete and one for the value of your stocks after you've been truncating to three decimal places at each step.

Start by initializing these variables to `100`:

Python >>>

```
>>> actual_value, truncated_value = 100, 100
```

Now let's run the simulation for 1,000,000 seconds (approximately 11.5 days). For each second, generate a random value between `-0.05` and `0.05` with the `uniform()` function in the `random` module, and then update `actual` and `truncated`:

Python >>>

```
>>> import random
>>> random.seed(100)

>>> for _ in range(1000000):
...     randn = random.uniform(-0.05, 0.05)
```

```
...     actual_value = actual_value + randn
...     truncated_value = truncate(truncated_value + randn)
...

>>> actual_value
96.45273913513529

>>> truncated_value
0.239
```

The meat of the simulation takes place in the `for` loop, which loops over the `range(1000000)` of numbers between `0` and `999,999`. The value taken from `range()` at each step is stored in the variable _, which we use here because we don't actually need this value inside of the loop.

At each step of the loop, a new random number between `-0.05` and `0.05` is generated using `random.randn()` and assigned to the variable `randn`. The new value of your investment is calculated by adding `randn` to `actual_value`, and the truncated total is calculated by adding `randn` to `truncated_value` and then truncating this value with `truncate()`.

As you can see by inspecting the `actual_value` variable after running the loop, you only lost about $3.55. However, if you'd been looking at `truncated_value`, you'd have thought that you'd lost almost all of your money!

> **Note:** In the above example, the `random.seed()` function is used to seed the pseudo-random number generator so that you can reproduce the output shown here.
>
> To learn more about randomness in Python, check out Real Python's Generating Random Data in Python (Guide).

Ignoring for the moment that `round()` doesn't behave quite as you expect, let's try re-running the simulation. We'll use `round()` this time to round to three decimal places at each step, and `seed()` the simulation again to get the same results as before:

Python >>>

```
>>> random.seed(100)
>>> actual_value, rounded_value = 100, 100

>>> for _ in range(1000000):
...     randn = random.uniform(-0.05, 0.05)
...     actual_value = actual_value + randn
...     rounded_value = round(rounded_value + randn, 3)
...

>>> actual_value
96.45273913513529

>>> rounded_value
96.258
```

What a difference!

Shocking as it may seem, this exact error caused quite a stir in the early 1980s when the system designed for recording the value of the Vancouver Stock Exchange truncated the overall index value to three decimal places instead of rounding. Rounding errors have swayed elections and even resulted in the loss of life.

How you round numbers is important, and as a responsible developer and software designer, you need to know what the common issues are and how to deal with them. Let's dive in and investigate what the different rounding methods are and how you can implement each one in pure Python.

A Menagerie of Methods

A Menagerie of Methods

There are a plethora of rounding strategies, each with advantages and disadvantages. In this section, you'll learn about some of the most common techniques, and how they can influence your data.

Truncation

The simplest, albeit crudest, method for rounding a number is to truncate the number to a given number of digits. When you truncate a number, you replace each digit after a given position with 0. Here are some examples:

A Menagerie of Methods

There are a plethora of rounding strategies, each with advantages and disadvantages. In this section, you'll learn about some of the most common techniques, and how they can influence your data.

Truncation

The simplest, albeit crudest, method for rounding a number is to truncate the number to a given number of digits. When you truncate a number, you replace each digit after a given position with 0. Here are some examples:

A Menagerie of Methods

There are a plethora of rounding strategies, each with advantages and disadvantages. In this section, you'll learn about some of the most common techniques, and how they can influence your data.

Truncation

The simplest, albeit crudest, method for rounding a number is to truncate the number to a given number of digits. When you truncate a number, you replace each digit after a given position with 0. Here are some examples:

A Menagerie of Methods

There are a plethora of rounding strategies, each with advantages and disadvantages. In this section, you'll learn about some of the most common techniques, and how they can influence your data.

Truncation

The simplest, albeit crudest, method for rounding a number is to truncate the number to a given number of digits. When you truncate a number, you replace each digit after a given position with 0. Here are some examples:

A Menagerie of Methods

There are a plethora of rounding strategies, each with advantages and disadvantages. In this section, you'll learn about some of the most common techniques, and how they can influence your data.

Truncation

The simplest, albeit crudest, method for rounding a number is to truncate the number to a given number of digits. When you truncate a number, you replace each digit after a given position with

position with 0. Here are some examples:

A Menagerie of Methods

There are a plethora of rounding strategies, each with advantages and disadvantages. In this section, you'll learn about some of the most common techniques, and how they can influence your data.

Truncation

The simplest, albeit crudest, method for rounding a number is to truncate the number to a given number of digits. When you truncate a number, you replace each digit after a given position with 0. Here are some examples:

A Menagerie of Methods

There are a plethora of rounding strategies, each with advantages and disadvantages. In this section, you'll learn about some of the most common techniques, and how they can influence your data.

Truncation

The simplest, albeit crudest, method for rounding a number is to truncate the number to a given number of digits. When you truncate a number, you replace each digit after a given position with 0. Here are some examples:

A Menagerie of Methods

There are a plethora of rounding strategies, each with advantages and disadvantages. In this section, you'll learn about some of the most common techniques, and how they can influence your data.

Truncation

The simplest, albeit crudest, method for rounding a number is to truncate the number to a given number of digits. When you truncate a number, you replace each digit after a given position with 0. Here are some examples:

A Menagerie of Methods

There are a plethora of rounding strategies, each with advantages and disadvantages. In this section, you'll learn about some of the most common techniques, and how they can influence your data.

Truncation

The simplest, albeit crudest, method for rounding a number is to truncate the number to a given number of digits. When you truncate a number, you replace each digit after a given position with 0. Here are some examples:

A Menagerie of Methods

There are a plethora of rounding strategies, each with advantages and disadvantages. In this section, you'll learn about some of the most common techniques, and how they can influence your data.

Truncation

The simplest, albeit crudest, method for rounding a number is to truncate the number to a given number of digits. When you truncate a number, you replace each digit after a given position with 0. Here are some examples:

A Menagerie of Methods

There are a plethora of rounding strategies, each with advantages and disadvantages. In this section, you'll learn about some of the most common techniques, and how they can influence your data.

Truncation

The simplest, albeit crudest, method for rounding a number is to truncate the number to a given number of digits. When you truncate a number, you replace each digit after a given position with 0. Here are some examples:

Remove ads

A Menagerie of Methods

There are a plethora of rounding strategies, each with advantages and disadvantages. In this section, you'll learn about some of the most common techniques, and how they can influence your data.

Truncation

The simplest, albeit crudest, method for rounding a number is to truncate the number to a given number of digits. When you truncate a number, you replace each digit after a given position with 0. Here are some examples:

Remove ads

A Menagerie of Methods

There are a plethora of rounding strategies, each with advantages and disadvantages. In this section, you'll learn about some of the most common techniques, and how they can influence your data.

Truncation

The simplest, albeit crudest, method for rounding a number is to truncate the number to a given number of digits. When you truncate a number, you replace each digit after a given position with 0. Here are some examples:

Remove ads

A Menagerie of Methods

There are a plethora of rounding strategies, each with advantages and disadvantages. In this section, you'll learn about some of the most common techniques, and how they can influence your data.

Truncation

The simplest, albeit crudest, method for rounding a number is to truncate the number to a given number of digits. When you truncate a number, you replace each digit after a given position with 0. Here are some examples:

Remove ads

A Menagerie of Methods

There are a plethora of rounding strategies, each with advantages and disadvantages. In this section, you'll learn about some of the most common techniques, and how they can influence your data.

Truncation

The simplest, albeit crudest, method for rounding a number is to truncate the number to a given number of digits. When you truncate a number, you replace each digit after a given position with 0. Here are some examples:

A Menagerie of Methods

There are a plethora of rounding strategies, each with advantages and disadvantages. In this section, you'll learn about some of the most common techniques, and how they can influence your data.

Truncation

The simplest, albeit crudest, method for rounding a number is to truncate the number to a given number of digits. When you truncate a number, you replace each digit after a given position with 0. Here are some examples:

Remove ads

A Menagerie of Methods

There are a plethora of rounding strategies, each with advantages and disadvantages. In this section, you'll learn about some of the most common techniques, and how they can influence your data.

Truncation

The simplest, albeit crudest, method for rounding a number is to truncate the number to a given number of digits. When you truncate a number, you replace each digit after a given position with 0. Here are some examples:

Remove ads

A Menagerie of Methods

There are a plethora of rounding strategies, each with advantages and disadvantages. In this section, you'll learn about some of the most common techniques, and how they can influence your data.

Truncation

The simplest, albeit crudest, method for rounding a number is to truncate the number to a given number of digits. When you truncate a number, you replace each digit after a given position with 0. Here are some examples:

Remove ads

A Menagerie of Methods

There are a plethora of rounding strategies, each with advantages and disadvantages. In this section, you'll learn about some of the most common techniques, and how they can influence your data.

Truncation

The simplest, albeit crudest, method for rounding a number is to truncate the number to a given number of digits. When you truncate a number, you replace each digit after a given position with 0. Here are some examples:

A Menagerie of Methods

There are a plethora of rounding strategies, each with advantages and disadvantages. In this section, you'll learn about some of the most common techniques, and how they can influence your data.

Truncation

The simplest, albeit crudest, method for rounding a number is to truncate the number to a given number of digits. When you truncate a number, you replace each digit after a given position with 0. Here are some examples:

strategy has a **round towards negative infinity bias**.

The "truncation" strategy exhibits a round towards negative infinity bias on positive values and a round towards positive infinity for negative values. Rounding functions with this behavior are said to have a **round towards zero bias**, in general.

Let's see how this works in practice. Consider the following list of floats:

Python >>>

```
>>> data = [1.25, -2.67, 0.43, -1.79, 4.32, -8.19]
```

Let's compute the mean value of the values in `data` using the `statistics.mean()` function:

Python >>>

```
>>> import statistics

>>> statistics.mean(data)
-1.1083333333333332
```

Now apply each of `round_up()`, `round_down()`, and `truncate()` in a list comprehension to round each number in `data` to one decimal place and calculate the new mean:

Python >>>

```
>>> ru_data = [round_up(n, 1) for n in data]
>>> ru_data
[1.3, -2.6, 0.5, -1.7, 4.4, -8.1]
>>> statistics.mean(ru_data)
-1.0333333333333332

>>> rd_data = [round_down(n, 1) for n in data]
>>> statistics.mean(rd_data)
-1.1333333333333333

>>> tr_data = [truncate(n, 1) for n in data]
>>> statistics.mean(tr_data)
-1.0833333333333333
```

After every number in `data` is rounded up, the new mean is about `-1.033`, which is greater than the actual mean of about `1.108`. Rounding down shifts the mean downwards to about `-1.133`. The mean of the truncated values is about `-1.08` and is the closest to the actual mean.

This example *does not* imply that you should always truncate when you need to round individual values while preserving a mean value as closely as possible. The `data` list contains an equal number of positive and negative values. The `truncate()` function would behave just like `round_up()` on a list of all positive values, and just like `round_down()` on a list of all negative values.

What this example does illustrate is the effect rounding bias has on values computed from data that has been rounded. You will need to keep these effects in mind when drawing conclusions from data that has been rounded.

Typically, when rounding, you are interested in rounding to the nearest number with some specified precision, instead of just rounding everything up or down.

specified precision, instead of just rounding everything up or down.

For example, if someone asks you to round the numbers `1.23` and `1.28` to one decimal place, you would probably respond quickly with `1.2` and `1.3`. The `truncate()`, `round_up()`, and `round_down()` functions don't do anything like this.

individual values while preserving a mean value as closely as possible. The data list contains an equal number of positive and negative values. The `truncate()` function would behave just like `round_up()` on a list of all positive values, and just like `round_down()` on a list of all negative values.

What this example does illustrate is the effect rounding bias has on values computed from data that has been rounded. You will need to keep these effects in mind when drawing conclusions from data that has been rounded.

Typically, when rounding, you are interested in rounding to the nearest number with some specified precision, instead of just rounding everything up or down.

For example, if someone asks you to round the numbers `1.23` and `1.28` to one decimal place, you would probably respond quickly with `1.2` and `1.3`. The `truncate()`, `round_up()`, and `round_down()` functions don't do anything like this.

individual values while preserving a mean value as closely as possible. The data list contains an equal number of positive and negative values. The `truncate()` function would behave just like `round_up()` on a list of all positive values, and just like `round_down()` on a list of all negative values.

What this example does illustrate is the effect rounding bias has on values computed from data that has been rounded. You will need to keep these effects in mind when drawing conclusions from data that has been rounded.

Typically, when rounding, you are interested in rounding to the nearest number with some specified precision, instead of just rounding everything up or down.

For example, if someone asks you to round the numbers `1.23` and `1.28` to one decimal place, you would probably respond quickly with `1.2` and `1.3`. The `truncate()`, `round_up()`, and `round_down()` functions don't do anything like this.

individual values while preserving a mean value as closely as possible. The data list contains an equal number of positive and negative values. The `truncate()` function would behave just like `round_up()` on a list of all positive values, and just like `round_down()` on a list of all negative values.

What this example does illustrate is the effect rounding bias has on values computed from data that has been rounded. You will need to keep these effects in mind when drawing conclusions from data that has been rounded.

Typically, when rounding, you are interested in rounding to the nearest number with some specified precision, instead of just rounding everything up or down.

For example, if someone asks you to round the numbers `1.23` and `1.28` to one decimal place, you would probably respond quickly with `1.2` and `1.3`. The `truncate()`, `round_up()`, and `round_down()` functions don't do anything like this.

individual values while preserving a mean value as closely as possible. The data list contains an equal number of positive and negative values. The `truncate()` function would behave just like `round_up()` on a list of all positive values, and just like `round_down()` on a list of all negative values.

What this example does illustrate is the effect rounding bias has on values computed from data that has been rounded. You will need to keep these effects in mind when drawing conclusions from data that has been rounded.

Typically, when rounding, you are interested in rounding to the nearest number with some specified precision, instead of just rounding everything up or down.

For example, if someone asks you to round the numbers `1.23` and `1.28` to one decimal place, you would probably respond quickly with `1.2` and `1.3`. The `truncate()`, `round_up()`, and `round_down()` functions don't do anything like this.

individual values while preserving a mean value as closely as possible. The data list contains an equal number of positive and negative values. The `truncate()` function would behave just like `round_up()` on a list of all positive values, and just like `round_down()` on a list of all negative values.

What this example does illustrate is the effect rounding bias has on values computed from

data that has been rounded. You will need to keep these effects in mind when drawing conclusions from data that has been rounded.

Typically, when rounding, you are interested in rounding to the nearest number with some specified precision, instead of just rounding everything up or down.

For example, if someone asks you to round the numbers `1.23` and `1.28` to one decimal place, you would probably respond quickly with `1.2` and `1.3`. The `truncate()`, `round_up()`, and `round_down()` functions don't do anything like this.

an equal number of positive and negative values. The `truncate()` function would behave just like `round_up()` on a list of all positive values, and just like `round_down()` on a list of all negative values.

What this example does illustrate is the effect rounding bias has on values computed from data that has been rounded. You will need to keep these effects in mind when drawing conclusions from data that has been rounded.

Typically, when rounding, you are interested in rounding to the nearest number with some specified precision, instead of just rounding everything up or down.

For example, if someone asks you to round the numbers `1.23` and `1.28` to one decimal place, you would probably respond quickly with `1.2` and `1.3`. The `truncate()`, `round_up()`, and `round_down()` functions don't do anything like this.

an equal number of positive and negative values. The `truncate()` function would behave just like `round_up()` on a list of all positive values, and just like `round_down()` on a list of all negative values.

What this example does illustrate is the effect rounding bias has on values computed from data that has been rounded. You will need to keep these effects in mind when drawing conclusions from data that has been rounded.

Typically, when rounding, you are interested in rounding to the nearest number with some specified precision, instead of just rounding everything up or down.

For example, if someone asks you to round the numbers `1.23` and `1.28` to one decimal place, you would probably respond quickly with `1.2` and `1.3`. The `truncate()`, `round_up()`, and `round_down()` functions don't do anything like this.

an equal number of positive and negative values. The `truncate()` function would behave just like `round_up()` on a list of all positive values, and just like `round_down()` on a list of all negative values.

What this example does illustrate is the effect rounding bias has on values computed from data that has been rounded. You will need to keep these effects in mind when drawing conclusions from data that has been rounded.

Typically, when rounding, you are interested in rounding to the nearest number with some specified precision, instead of just rounding everything up or down.

For example, if someone asks you to round the numbers `1.23` and `1.28` to one decimal place, you would probably respond quickly with `1.2` and `1.3`. The `truncate()`, `round_up()`, and `round_down()` functions don't do anything like this.

an equal number of positive and negative values. The `truncate()` function would behave just like `round_up()` on a list of all positive values, and just like `round_down()` on a list of all negative values.

What this example does illustrate is the effect rounding bias has on values computed from data that has been rounded. You will need to keep these effects in mind when drawing conclusions from data that has been rounded.

Typically, when rounding, you are interested in rounding to the nearest number with some specified precision, instead of just rounding everything up or down.

For example, if someone asks you to round the numbers `1.23` and `1.28` to one decimal place, you would probably respond quickly with `1.2` and `1.3`. The `truncate()`, `round_up()`, and `round_down()` functions don't do anything like this.

an equal number of positive and negative values. The `truncate()` function would behave

just like `round_up()` on a list of all positive values, and just like `round_down()` on a list of all negative values.

What this example does illustrate is the effect rounding bias has on values computed from data that has been rounded. You will need to keep these effects in mind when drawing conclusions from data that has been rounded.

Typically, when rounding, you are interested in rounding to the nearest number with some specified precision, instead of just rounding everything up or down.

For example, if someone asks you to round the numbers `1.23` and `1.28` to one decimal place, you would probably respond quickly with `1.2` and `1.3`. The `truncate()`, `round_up()`, and `round_down()` functions don't do anything like this.

individual values while preserving a mean value as closely as possible. The `data` list contains an equal number of positive and negative values. The `truncate()` function would behave

just like `round_up()` on a list of all positive values, and just like `round_down()` on a list of all negative values.

What this example does illustrate is the effect rounding bias has on values computed from data that has been rounded. You will need to keep these effects in mind when drawing conclusions from data that has been rounded.

Typically, when rounding, you are interested in rounding to the nearest number with some specified precision, instead of just rounding everything up or down.

For example, if someone asks you to round the numbers `1.23` and `1.28` to one decimal place, you would probably respond quickly with `1.2` and `1.3`. The `truncate()`, `round_up()`, and `round_down()` functions don't do anything like this.

individual values while preserving a mean value as closely as possible. The `data` list contains an equal number of positive and negative values. The `truncate()` function would behave just like `round_up()` on a list of all positive values, and just like `round_down()` on a list of all negative values.

What this example does illustrate is the effect rounding bias has on values computed from data that has been rounded. You will need to keep these effects in mind when drawing conclusions from data that has been rounded.

Typically, when rounding, you are interested in rounding to the nearest number with some specified precision, instead of just rounding everything up or down.

For example, if someone asks you to round the numbers `1.23` and `1.28` to one decimal place, you would probably respond quickly with `1.2` and `1.3`. The `truncate()`, `round_up()`, and `round_down()` functions don't do anything like this.

individual values while preserving a mean value as closely as possible. The `data` list contains an equal number of positive and negative values. The `truncate()` function would behave just like `round_up()` on a list of all positive values, and just like `round_down()` on a list of all negative values.

What this example does illustrate is the effect rounding bias has on values computed from data that has been rounded. You will need to keep these effects in mind when drawing conclusions from data that has been rounded.

Typically, when rounding, you are interested in rounding to the nearest number with some specified precision, instead of just rounding everything up or down.

For example, if someone asks you to round the numbers `1.23` and `1.28` to one decimal place, you would probably respond quickly with `1.2` and `1.3`. The `truncate()`, `round_up()`, and `round_down()` functions don't do anything like this.

individual values while preserving a mean value as closely as possible. The `data` list contains an equal number of positive and negative values. The `truncate()` function would behave just like `round_up()` on a list of all positive values, and just like `round_down()` on a list of all negative values.

What this example does illustrate is the effect rounding bias has on values computed from data that has been rounded. You will need to keep these effects in mind when drawing conclusions from data that has been rounded.

Typically, when rounding, you are interested in rounding to the nearest number with some specified precision, instead of just rounding everything up or down.

For example, if someone asks you to round the numbers `1.23` and `1.28` to one decimal place, you would probably respond quickly with `1.2` and `1.3`. The `truncate()`, `round_up()`, and `round_down()` functions don't do anything like this.

individual values while preserving a mean value as closely as possible. The `data` list contains an equal number of positive and negative values. The `truncate()` function would behave just like `round_up()` on a list of all positive values, and just like `round_down()` on a list of all negative values.

What this example does illustrate is the effect rounding bias has on values computed from data that has been rounded. You will need to keep these effects in mind when drawing conclusions from data that has been rounded.

Typically, when rounding, you are interested in rounding to the nearest number with some specified precision, instead of just rounding everything up or down.

For example, if someone asks you to round the numbers `1.23` and `1.28` to one decimal place, you would probably respond quickly with `1.2` and `1.3`. The `truncate()`, `round_up()`, and `round_down()` functions don't do anything like this.

individual values while preserving a mean value as closely as possible. The `data` list contains an equal number of positive and negative values. The `truncate()` function would behave just like `round_up()` on a list of all positive values, and just like `round_down()` on a list of all negative values.

What this example does illustrate is the effect rounding bias has on values computed from data that has been rounded. You will need to keep these effects in mind when drawing conclusions from data that has been rounded.

Typically, when rounding, you are interested in rounding to the nearest number with some specified precision, instead of just rounding everything up or down.

For example, if someone asks you to round the numbers `1.23` and `1.28` to one decimal place, you would probably respond quickly with `1.2` and `1.3`. The `truncate()`, `round_up()`, and `round_down()` functions don't do anything like this.

individual values while preserving a mean value as closely as possible. The `data` list contains an equal number of positive and negative values. The `truncate()` function would behave just like `round_up()` on a list of all positive values, and just like `round_down()` on a list of all negative values.

What this example does illustrate is the effect rounding bias has on values computed from data that has been rounded. You will need to keep these effects in mind when drawing conclusions from data that has been rounded.

Typically, when rounding, you are interested in rounding to the nearest number with some specified precision, instead of just rounding everything up or down.

For example, if someone asks you to round the numbers `1.23` and `1.28` to one decimal place, you would probably respond quickly with `1.2` and `1.3`. The `truncate()`, `round_up()`, and `round_down()` functions don't do anything like this.

individual values while preserving a mean value as closely as possible. The `data` list contains an equal number of positive and negative values. The `truncate()` function would behave just like `round_up()` on a list of all positive values, and just like `round_down()` on a list of all negative values.

What this example does illustrate is the effect rounding bias has on values computed from data that has been rounded. You will need to keep these effects in mind when drawing conclusions from data that has been rounded.

Typically, when rounding, you are interested in rounding to the nearest number with some specified precision, instead of just rounding everything up or down.

For example, if someone asks you to round the numbers `1.23` and `1.28` to one decimal place, you would probably respond quickly with `1.2` and `1.3`. The `truncate()`, `round_up()`, and `round_down()` functions don't do anything like this.

individual values while preserving a mean value as closely as possible. The `data` list contains an equal number of positive and negative values. The `truncate()` function would behave just like `round_up()` on a list of all positive values, and just like `round_down()` on a list of all negative values.

What this example does illustrate is the effect rounding bias has on values computed from data that has been rounded. You will need to keep these effects in mind when drawing

conclusions from data that has been rounded.

Typically, when rounding, you are interested in rounding to the nearest number with some specified precision, instead of just rounding everything up or down.

For example, if someone asks you to round the numbers `1.23` and `1.28` to one decimal place, you would probably respond quickly with `1.2` and `1.3`. The `truncate()`, `round_up()`, and `round_down()` functions don't do anything like this.

individual values while preserving a mean value as closely as possible. The `data` list contains an equal number of positive and negative values. The `truncate()` function would behave just like `round_up()` on a list of all positive values, and just like `round_down()` on a list of all negative values.

What this example does illustrate is the effect rounding bias has on values computed from data that has been rounded. You will need to keep these effects in mind when drawing conclusions from data that has been rounded.

Typically, when rounding, you are interested in rounding to the nearest number with some specified precision, instead of just rounding everything up or down.

For example, if someone asks you to round the numbers `1.23` and `1.28` to one decimal place, you would probably respond quickly with `1.2` and `1.3`. The `truncate()`, `round_up()`, and `round_down()` functions don't do anything like this.

individual values while preserving a mean value as closely as possible. The `data` list contains an equal number of positive and negative values. The `truncate()` function would behave just like `round_up()` on a list of all positive values, and just like `round_down()` on a list of all negative values.

What this example does illustrate is the effect rounding bias has on values computed from data that has been rounded. You will need to keep these effects in mind when drawing conclusions from data that has been rounded.

Typically, when rounding, you are interested in rounding to the nearest number with some specified precision, instead of just rounding everything up or down.

For example, if someone asks you to round the numbers `1.23` and `1.28` to one decimal place, you would probably respond quickly with `1.2` and `1.3`. The `truncate()`, `round_up()`, and `round_down()` functions don't do anything like this.

individual values while preserving a mean value as closely as possible. The `data` list contains an equal number of positive and negative values. The `truncate()` function would behave just like `round_up()` on a list of all positive values, and just like `round_down()` on a list of all negative values.

What this example does illustrate is the effect rounding bias has on values computed from data that has been rounded. You will need to keep these effects in mind when drawing conclusions from data that has been rounded.

Typically, when rounding, you are interested in rounding to the nearest number with some specified precision, instead of just rounding everything up or down.

For example, if someone asks you to round the numbers `1.23` and `1.28` to one decimal place, you would probably respond quickly with `1.2` and `1.3`. The `truncate()`, `round_up()`, and `round_down()` functions don't do anything like this.

individual values while preserving a mean value as closely as possible. The `data` list contains an equal number of positive and negative values. The `truncate()` function would behave just like `round_up()` on a list of all positive values, and just like `round_down()` on a list of all negative values.

What this example does illustrate is the effect rounding bias has on values computed from data that has been rounded. You will need to keep these effects in mind when drawing conclusions from data that has been rounded.

Typically, when rounding, you are interested in rounding to the nearest number with some specified precision, instead of just rounding everything up or down.

For example, if someone asks you to round the numbers `1.23` and `1.28` to one decimal place, you would probably respond quickly with `1.2` and `1.3`. The `truncate()`, `round_up()`, and `round_down()` functions don't do anything like this.

individual values while preserving a mean value as closely as possible. The `data` list contains an equal number of positive and negative values. The `truncate()` function would behave just like `round_up()` on a list of all positive values, and just like `round_down()` on a list of all

negative values.

What this example does illustrate is the effect rounding bias has on values computed from data that has been rounded. You will need to keep these effects in mind when drawing conclusions from data that has been rounded.

Typically, when rounding, you are interested in rounding to the nearest number with some specified precision, instead of just rounding everything up or down.

For example, if someone asks you to round the numbers `1.23` and `1.28` to one decimal place, you would probably respond quickly with `1.2` and `1.3`. The `truncate()`, `round_up()`, and `round_down()` functions don't do anything like this.

individual values while preserving a mean value as closely as possible. The data list contains an equal number of positive and negative values. The `truncate()` function would behave just like `round_up()` on a list of all positive values, and just like `round_down()` on a list of all negative values.

What this example does illustrate is the effect rounding bias has on values computed from data that has been rounded. You will need to keep these effects in mind when drawing conclusions from data that has been rounded.

Typically, when rounding, you are interested in rounding to the nearest number with some specified precision, instead of just rounding everything up or down.

For example, if someone asks you to round the numbers `1.23` and `1.28` to one decimal place, you would probably respond quickly with `1.2` and `1.3`. The `truncate()`, `round_up()`, and `round_down()` functions don't do anything like this.

individual values while preserving a mean value as closely as possible. The data list contains an equal number of positive and negative values. The `truncate()` function would behave just like `round_up()` on a list of all positive values, and just like `round_down()` on a list of all negative values.

What this example does illustrate is the effect rounding bias has on values computed from data that has been rounded. You will need to keep these effects in mind when drawing conclusions from data that has been rounded.

Typically, when rounding, you are interested in rounding to the nearest number with some specified precision, instead of just rounding everything up or down.

For example, if someone asks you to round the numbers `1.23` and `1.28` to one decimal place, you would probably respond quickly with `1.2` and `1.3`. The `truncate()`, `round_up()`, and `round_down()` functions don't do anything like this.

individual values while preserving a mean value as closely as possible. The data list contains an equal number of positive and negative values. The `truncate()` function would behave just like `round_up()` on a list of all positive values, and just like `round_down()` on a list of all negative values.

What this example does illustrate is the effect rounding bias has on values computed from data that has been rounded. You will need to keep these effects in mind when drawing conclusions from data that has been rounded.

Typically, when rounding, you are interested in rounding to the nearest number with some specified precision, instead of just rounding everything up or down.

For example, if someone asks you to round the numbers `1.23` and `1.28` to one decimal place, you would probably respond quickly with `1.2` and `1.3`. The `truncate()`, `round_up()`, and `round_down()` functions don't do anything like this.

individual values while preserving a mean value as closely as possible. The data list contains an equal number of positive and negative values. The `truncate()` function would behave just like `round_up()` on a list of all positive values, and just like `round_down()` on a list of all negative values.

What this example does illustrate is the effect rounding bias has on values computed from data that has been rounded. You will need to keep these effects in mind when drawing conclusions from data that has been rounded.

Typically, when rounding, you are interested in rounding to the nearest number with some specified precision, instead of just rounding everything up or down.

For example, if someone asks you to round the numbers `1.23` and `1.28` to one decimal place,

you would probably respond quickly with `1.2` and `1.3`. The `truncate()`, `round_up()`, and `round_down()` functions don't do anything like this.

individual values while preserving a mean value as closely as possible. The data list contains an equal number of positive and negative values. The `truncate()` function would behave just like `round_up()` on a list of all positive values, and just like `round_down()` on a list of all negative values.

What this example does illustrate is the effect rounding bias has on values computed from data that has been rounded. You will need to keep these effects in mind when drawing conclusions from data that has been rounded.

Typically, when rounding, you are interested in rounding to the nearest number with some specified precision, instead of just rounding everything up or down.

For example, if someone asks you to round the numbers `1.23` and `1.28` to one decimal place, you would probably respond quickly with `1.2` and `1.3`. The `truncate()`, `round_up()`, and `round_down()` functions don't do anything like this.

individual values while preserving a mean value as closely as possible. The data list contains an equal number of positive and negative values. The `truncate()` function would behave just like `round_up()` on a list of all positive values, and just like `round_down()` on a list of all negative values.

What this example does illustrate is the effect rounding bias has on values computed from data that has been rounded. You will need to keep these effects in mind when drawing conclusions from data that has been rounded.

Typically, when rounding, you are interested in rounding to the nearest number with some specified precision, instead of just rounding everything up or down.

For example, if someone asks you to round the numbers `1.23` and `1.28` to one decimal place, you would probably respond quickly with `1.2` and `1.3`. The `truncate()`, `round_up()`, and `round_down()` functions don't do anything like this.

individual values while preserving a mean value as closely as possible. The data list contains an equal number of positive and negative values. The `truncate()` function would behave just like `round_up()` on a list of all positive values, and just like `round_down()` on a list of all negative values.

What this example does illustrate is the effect rounding bias has on values computed from data that has been rounded. You will need to keep these effects in mind when drawing conclusions from data that has been rounded.

Typically, when rounding, you are interested in rounding to the nearest number with some specified precision, instead of just rounding everything up or down.

For example, if someone asks you to round the numbers `1.23` and `1.28` to one decimal place, you would probably respond quickly with `1.2` and `1.3`. The `truncate()`, `round_up()`, and `round_down()` functions don't do anything like this.

individual values while preserving a mean value as closely as possible. The data list contains an equal number of positive and negative values. The `truncate()` function would behave just like `round_up()` on a list of all positive values, and just like `round_down()` on a list of all negative values.

What this example does illustrate is the effect rounding bias has on values computed from data that has been rounded. You will need to keep these effects in mind when drawing conclusions from data that has been rounded.

Typically, when rounding, you are interested in rounding to the nearest number with some specified precision, instead of just rounding everything up or down.

For example, if someone asks you to round the numbers `1.23` and `1.28` to one decimal place, you would probably respond quickly with `1.2` and `1.3`. The `truncate()`, `round_up()`, and `round_down()` functions don't do anything like this.

individual values while preserving a mean value as closely as possible. The data list contains an equal number of positive and negative values. The `truncate()` function would behave just like `round_up()` on a list of all positive values, and just like `round_down()` on a list of all negative values.

What this example does illustrate is the effect rounding bias has on values computed from data that has been rounded. You will need to keep these effects in mind when drawing conclusions from data that has been rounded.

Typically, when rounding, you are interested in rounding to the nearest number with some specified precision, instead of just rounding everything up or down.

For example, if someone asks you to round the numbers `1.23` and `1.28` to one decimal place, you would probably respond quickly with `1.2` and `1.3`. The `truncate()`, `round_up()`, and `round_down()` functions don't do anything like this.

individual values while preserving a mean value as closely as possible. The data list contains an equal number of positive and negative values. The `truncate()` function would behave just like `round_up()` on a list of all positive values, and just like `round_down()` on a list of all negative values.

What this example does illustrate is the effect rounding bias has on values computed from data that has been rounded. You will need to keep these effects in mind when drawing conclusions from data that has been rounded.

Typically, when rounding, you are interested in rounding to the nearest number with some specified precision, instead of just rounding everything up or down.

For example, if someone asks you to round the numbers `1.23` and `1.28` to one decimal place, you would probably respond quickly with `1.2` and `1.3`. The `truncate()`, `round_up()`, and `round_down()` functions don't do anything like this.

individual values while preserving a mean value as closely as possible. The data list contains an equal number of positive and negative values. The `truncate()` function would behave just like `round_up()` on a list of all positive values, and just like `round_down()` on a list of all negative values.

What this example does illustrate is the effect rounding bias has on values computed from data that has been rounded. You will need to keep these effects in mind when drawing conclusions from data that has been rounded.

Typically, when rounding, you are interested in rounding to the nearest number with some specified precision, instead of just rounding everything up or down.

For example, if someone asks you to round the numbers `1.23` and `1.28` to one decimal place, you would probably respond quickly with `1.2` and `1.3`. The `truncate()`, `round_up()`, and `round_down()` functions don't do anything like this.

individual values while preserving a mean value as closely as possible. The data list contains an equal number of positive and negative values. The `truncate()` function would behave just like `round_up()` on a list of all positive values, and just like `round_down()` on a list of all negative values.

What this example does illustrate is the effect rounding bias has on values computed from data that has been rounded. You will need to keep these effects in mind when drawing conclusions from data that has been rounded.

Typically, when rounding, you are interested in rounding to the nearest number with some specified precision, instead of just rounding everything up or down.

For example, if someone asks you to round the numbers `1.23` and `1.28` to one decimal place, you would probably respond quickly with `1.2` and `1.3`. The `truncate()`, `round_up()`, and `round_down()` functions don't do anything like this.

individual values while preserving a mean value as closely as possible. The data list contains an equal number of positive and negative values. The `truncate()` function would behave just like `round_up()` on a list of all positive values, and just like `round_down()` on a list of all negative values.

What this example does illustrate is the effect rounding bias has on values computed from data that has been rounded. You will need to keep these effects in mind when drawing conclusions from data that has been rounded.

Typically, when rounding, you are interested in rounding to the nearest number with some specified precision, instead of just rounding everything up or down.

For example, if someone asks you to round the numbers `1.23` and `1.28` to one decimal place, you would probably respond quickly with `1.2` and `1.3`. The `truncate()`, `round_up()`, and `round_down()` functions don't do anything like this.

individual values while preserving a mean value as closely as possible. The data list contains an equal number of positive and negative values. The `truncate()` function would behave just like `round_up()` on a list of all positive values, and just like `round_down()` on a list of all negative values.

What this example does illustrate is the effect rounding bias has on values computed from data that has been rounded. You will need to keep these effects in mind when drawing conclusions from data that has been rounded.

Typically, when rounding, you are interested in rounding to the nearest number with some specified precision, instead of just rounding everything up or down.

For example, if someone asks you to round the numbers `1.23` and `1.28` to one decimal place, you would probably respond quickly with `1.2` and `1.3`. The `truncate()`, `round_up()`, and `round_down()` functions don't do anything like this.

individual values while preserving a mean value as closely as possible. The data list contains an equal number of positive and negative values. The `truncate()` function would behave just like `round_up()` on a list of all positive values, and just like `round_down()` on a list of all negative values.

What this example does illustrate is the effect rounding bias has on values computed from data that has been rounded. You will need to keep these effects in mind when drawing conclusions from data that has been rounded.

Typically, when rounding, you are interested in rounding to the nearest number with some specified precision, instead of just rounding everything up or down.

For example, if someone asks you to round the numbers `1.23` and `1.28` to one decimal place, you would probably respond quickly with `1.2` and `1.3`. The `truncate()`, `round_up()`, and `round_down()` functions don't do anything like this.

individual values while preserving a mean value as closely as possible. The data list contains an equal number of positive and negative values. The `truncate()` function would behave just like `round_up()` on a list of all positive values, and just like `round_down()` on a list of all negative values.

What this example does illustrate is the effect rounding bias has on values computed from data that has been rounded. You will need to keep these effects in mind when drawing conclusions from data that has been rounded.

Typically, when rounding, you are interested in rounding to the nearest number with some specified precision, instead of just rounding everything up or down.

For example, if someone asks you to round the numbers `1.23` and `1.28` to one decimal place, you would probably respond quickly with `1.2` and `1.3`. The `truncate()`, `round_up()`, and `round_down()` functions don't do anything like this.

individual values while preserving a mean value as closely as possible. The data list contains an equal number of positive and negative values. The `truncate()` function would behave just like `round_up()` on a list of all positive values, and just like `round_down()` on a list of all negative values.

What this example does illustrate is the effect rounding bias has on values computed from data that has been rounded. You will need to keep these effects in mind when drawing conclusions from data that has been rounded.

Typically, when rounding, you are interested in rounding to the nearest number with some specified precision, instead of just rounding everything up or down.

For example, if someone asks you to round the numbers `1.23` and `1.28` to one decimal place, you would probably respond quickly with `1.2` and `1.3`. The `truncate()`, `round_up()`, and `round_down()` functions don't do anything like this.

individual values while preserving a mean value as closely as possible. The data list contains an equal number of positive and negative values. The `truncate()` function would behave just like `round_up()` on a list of all positive values, and just like `round_down()` on a list of all negative values.

What this example does illustrate is the effect rounding bias has on values computed from data that has been rounded. You will need to keep these effects in mind when drawing conclusions from data that has been rounded.

Typically, when rounding, you are interested in rounding to the nearest number with some specified precision, instead of just rounding everything up or down.

For example, if someone asks you to round the numbers `1.23` and `1.28` to one decimal place, you would probably respond quickly with `1.2` and `1.3`. The `truncate()`, `round_up()`, and

`round_down()` functions don’t do anything like this.
individual values while preserving a mean value as closely as possible. The data list contains an equal number of positive and negative values. The `truncate()` function would behave just like `round_up()` on a list of all positive values, and just like `round_down()` on a list of all negative values.

What this example does illustrate is the effect rounding bias has on values computed from data that has been rounded. You will need to keep these effects in mind when drawing conclusions from data that has been rounded.

Typically, when rounding, you are interested in rounding to the nearest number with some specified precision, instead of just rounding everything up or down.

For example, if someone asks you to round the numbers `1.23` and `1.28` to one decimal place, you would probably respond quickly with `1.2` and `1.3`. The `truncate()`, `round_up()`, and

`round_down()` functions don’t do anything like this.
individual values while preserving a mean value as closely as possible. The data list contains an equal number of positive and negative values. The `truncate()` function would behave just like `round_up()` on a list of all positive values, and just like `round_down()` on a list of all negative values.

What this example does illustrate is the effect rounding bias has on values computed from data that has been rounded. You will need to keep these effects in mind when drawing conclusions from data that has been rounded.

Typically, when rounding, you are interested in rounding to the nearest number with some specified precision, instead of just rounding everything up or down.

For example, if someone asks you to round the numbers `1.23` and `1.28` to one decimal place, you would probably respond quickly with `1.2` and `1.3`. The `truncate()`, `round_up()`, and

`round_down()` functions don’t do anything like this.
individual values while preserving a mean value as closely as possible. The data list contains an equal number of positive and negative values. The `truncate()` function would behave just like `round_up()` on a list of all positive values, and just like `round_down()` on a list of all negative values.

What this example does illustrate is the effect rounding bias has on values computed from data that has been rounded. You will need to keep these effects in mind when drawing conclusions from data that has been rounded.

Typically, when rounding, you are interested in rounding to the nearest number with some specified precision, instead of just rounding everything up or down.

For example, if someone asks you to round the numbers `1.23` and `1.28` to one decimal place, you would probably respond quickly with `1.2` and `1.3`. The `truncate()`, `round_up()`, and

`round_down()` functions don’t do anything like this.
individual values while preserving a mean value as closely as possible. The data list contains an equal number of positive and negative values. The `truncate()` function would behave just like `round_up()` on a list of all positive values, and just like `round_down()` on a list of all negative values.

What this example does illustrate is the effect rounding bias has on values computed from data that has been rounded. You will need to keep these effects in mind when drawing conclusions from data that has been rounded.

Typically, when rounding, you are interested in rounding to the nearest number with some specified precision, instead of just rounding everything up or down.

For example, if someone asks you to round the numbers `1.23` and `1.28` to one decimal place, you would probably respond quickly with `1.2` and `1.3`. The `truncate()`, `round_up()`, and

`round_down()` functions don’t do anything like this.
individual values while preserving a mean value as closely as possible. The data list contains an equal number of positive and negative values. The `truncate()` function would behave just like `round_up()` on a list of all positive values, and just like `round_down()` on a list of all negative values.

What this example does illustrate is the effect rounding bias has on values computed from data that has been rounded. You will need to keep these effects in mind when drawing conclusions from data that has been rounded.

Typically, when rounding, you are interested in rounding to the nearest number with some

specified precision, instead of just rounding everything up or down.

For example, if someone asks you to round the numbers 1.23 and 1.28 to one decimal place, you would probably respond quickly with 1.2 and 1.3. The truncate(), round_up(), and round_down() functions don't do anything like this.

individual values while preserving a mean value as closely as possible. The data list contains an equal number of positive and negative values. The truncate() function would behave just like round_up() on a list of all positive values, and just like round_down() on a list of all negative values.

What this example does illustrate is the effect rounding bias has on values computed from data that has been rounded. You will need to keep these effects in mind when drawing conclusions from data that has been rounded.

Typically, when rounding, you are interested in rounding to the nearest number with some specified precision, instead of just rounding everything up or down.

For example, if someone asks you to round the numbers 1.23 and 1.28 to one decimal place, you would probably respond quickly with 1.2 and 1.3. The truncate(), round_up(), and round_down() functions don't do anything like this.

individual values while preserving a mean value as closely as possible. The data list contains an equal number of positive and negative values. The truncate() function would behave just like round_up() on a list of all positive values, and just like round_down() on a list of all negative values.

What this example does illustrate is the effect rounding bias has on values computed from data that has been rounded. You will need to keep these effects in mind when drawing conclusions from data that has been rounded.

Typically, when rounding, you are interested in rounding to the nearest number with some specified precision, instead of just rounding everything up or down.

For example, if someone asks you to round the numbers 1.23 and 1.28 to one decimal place, you would probably respond quickly with 1.2 and 1.3. The truncate(), round_up(), and round_down() functions don't do anything like this.

individual values while preserving a mean value as closely as possible. The data list contains an equal number of positive and negative values. The truncate() function would behave just like round_up() on a list of all positive values, and just like round_down() on a list of all negative values.

What this example does illustrate is the effect rounding bias has on values computed from data that has been rounded. You will need to keep these effects in mind when drawing conclusions from data that has been rounded.

Typically, when rounding, you are interested in rounding to the nearest number with some specified precision, instead of just rounding everything up or down.

For example, if someone asks you to round the numbers 1.23 and 1.28 to one decimal place, you would probably respond quickly with 1.2 and 1.3. The truncate(), round_up(), and round_down() functions don't do anything like this.

individual values while preserving a mean value as closely as possible. The data list contains an equal number of positive and negative values. The truncate() function would behave just like round_up() on a list of all positive values, and just like round_down() on a list of all negative values.

What this example does illustrate is the effect rounding bias has on values computed from data that has been rounded. You will need to keep these effects in mind when drawing conclusions from data that has been rounded.

Typically, when rounding, you are interested in rounding to the nearest number with some specified precision, instead of just rounding everything up or down.

For example, if someone asks you to round the numbers 1.23 and 1.28 to one decimal place, you would probably respond quickly with 1.2 and 1.3. The truncate(), round_up(), and round_down() functions don't do anything like this.

individual values while preserving a mean value as closely as possible. The data list contains an equal number of positive and negative values. The truncate() function would behave just like round_up() on a list of all positive values, and just like round_down() on a list of all negative values.

What this example does illustrate is the effect rounding bias has on values computed from

What this example does illustrate is the effect rounding bias has on values computed from data that has been rounded. You will need to keep these effects in mind when drawing conclusions from data that has been rounded.

Typically, when rounding, you are interested in rounding to the nearest number with some specified precision, instead of just rounding everything up or down.

For example, if someone asks you to round the numbers `1.23` and `1.28` to one decimal place, you would probably respond quickly with `1.2` and `1.3`. The `truncate()`, `round_up()`, and `round_down()` functions don't do anything like this.

individual values while preserving a mean value as closely as possible. The `data` list contains an equal number of positive and negative values. The `truncate()` function would behave just like `round_up()` on a list of all positive values, and just like `round_down()` on a list of all negative values.

What this example does illustrate is the effect rounding bias has on values computed from data that has been rounded. You will need to keep these effects in mind when drawing conclusions from data that has been rounded.

Typically, when rounding, you are interested in rounding to the nearest number with some specified precision, instead of just rounding everything up or down.

For example, if someone asks you to round the numbers `1.23` and `1.28` to one decimal place, you would probably respond quickly with `1.2` and `1.3`. The `truncate()`, `round_up()`, and `round_down()` functions don't do anything like this.

individual values while preserving a mean value as closely as possible. The `data` list contains an equal number of positive and negative values. The `truncate()` function would behave just like `round_up()` on a list of all positive values, and just like `round_down()` on a list of all negative values.

What this example does illustrate is the effect rounding bias has on values computed from data that has been rounded. You will need to keep these effects in mind when drawing conclusions from data that has been rounded.

Typically, when rounding, you are interested in rounding to the nearest number with some specified precision, instead of just rounding everything up or down.

For example, if someone asks you to round the numbers `1.23` and `1.28` to one decimal place, you would probably respond quickly with `1.2` and `1.3`. The `truncate()`, `round_up()`, and `round_down()` functions don't do anything like this.

individual values while preserving a mean value as closely as possible. The `data` list contains an equal number of positive and negative values. The `truncate()` function would behave just like `round_up()` on a list of all positive values, and just like `round_down()` on a list of all negative values.

What this example does illustrate is the effect rounding bias has on values computed from data that has been rounded. You will need to keep these effects in mind when drawing conclusions from data that has been rounded.

Typically, when rounding, you are interested in rounding to the nearest number with some specified precision, instead of just rounding everything up or down.

For example, if someone asks you to round the numbers `1.23` and `1.28` to one decimal place, you would probably respond quickly with `1.2` and `1.3`. The `truncate()`, `round_up()`, and `round_down()` functions don't do anything like this.

individual values while preserving a mean value as closely as possible. The `data` list contains an equal number of positive and negative values. The `truncate()` function would behave just like `round_up()` on a list of all positive values, and just like `round_down()` on a list of all negative values.

What this example does illustrate is the effect rounding bias has on values computed from data that has been rounded. You will need to keep these effects in mind when drawing conclusions from data that has been rounded.

Typically, when rounding, you are interested in rounding to the nearest number with some specified precision, instead of just rounding everything up or down.

For example, if someone asks you to round the numbers `1.23` and `1.28` to one decimal place, you would probably respond quickly with `1.2` and `1.3`. The `truncate()`, `round_up()`, and `round_down()` functions don't do anything like this.

individual values while preserving a mean value as closely as possible. The `data` list contains

an equal number of positive and negative values. The `truncate()` function would behave just like `round_up()` on a list of all positive values, and just like `round_down()` on a list of all negative values.

What this example does illustrate is the effect rounding bias has on values computed from data that has been rounded. You will need to keep these effects in mind when drawing conclusions from data that has been rounded.

Typically, when rounding, you are interested in rounding to the nearest number with some specified precision, instead of just rounding everything up or down.

For example, if someone asks you to round the numbers `1.23` and `1.28` to one decimal place, you would probably respond quickly with `1.2` and `1.3`. The `truncate()`, `round_up()`, and `round_down()` functions don't do anything like this.

an equal number of positive and negative values. The `truncate()` function would behave just like `round_up()` on a list of all positive values, and just like `round_down()` on a list of all negative values.

What this example does illustrate is the effect rounding bias has on values computed from data that has been rounded. You will need to keep these effects in mind when drawing conclusions from data that has been rounded.

Typically, when rounding, you are interested in rounding to the nearest number with some specified precision, instead of just rounding everything up or down.

For example, if someone asks you to round the numbers `1.23` and `1.28` to one decimal place, you would probably respond quickly with `1.2` and `1.3`. The `truncate()`, `round_up()`, and `round_down()` functions don't do anything like this.

an equal number of positive and negative values. The `truncate()` function would behave just like `round_up()` on a list of all positive values, and just like `round_down()` on a list of all negative values.

What this example does illustrate is the effect rounding bias has on values computed from data that has been rounded. You will need to keep these effects in mind when drawing conclusions from data that has been rounded.

Typically, when rounding, you are interested in rounding to the nearest number with some specified precision, instead of just rounding everything up or down.

For example, if someone asks you to round the numbers `1.23` and `1.28` to one decimal place, you would probably respond quickly with `1.2` and `1.3`. The `truncate()`, `round_up()`, and `round_down()` functions don't do anything like this.

an equal number of positive and negative values. The `truncate()` function would behave just like `round_up()` on a list of all positive values, and just like `round_down()` on a list of all negative values.

What this example does illustrate is the effect rounding bias has on values computed from data that has been rounded. You will need to keep these effects in mind when drawing conclusions from data that has been rounded.

Typically, when rounding, you are interested in rounding to the nearest number with some specified precision, instead of just rounding everything up or down.

For example, if someone asks you to round the numbers `1.23` and `1.28` to one decimal place, you would probably respond quickly with `1.2` and `1.3`. The `truncate()`, `round_up()`, and `round_down()` functions don't do anything like this.

an equal number of positive and negative values. The `truncate()` function would behave just like `round_up()` on a list of all positive values, and just like `round_down()` on a list of all negative values.

What this example does illustrate is the effect rounding bias has on values computed from data that has been rounded. You will need to keep these effects in mind when drawing conclusions from data that has been rounded.

Typically, when rounding, you are interested in rounding to the nearest number with some specified precision, instead of just rounding everything up or down.

For example, if someone asks you to round the numbers `1.23` and `1.28` to one decimal place, you would probably respond quickly with `1.2` and `1.3`. The `truncate()`, `round_up()`, and `round_down()` functions don't do anything like this.

an equal number of positive and negative values. The `truncate()` function would behave just like `round_up()` on a list of all positive values, and just like `round_down()` on a list of all negative values.

What this example does illustrate is the effect rounding bias has on values computed from data that has been rounded. You will need to keep these effects in mind when drawing conclusions from data that has been rounded.

Typically, when rounding, you are interested in rounding to the nearest number with some specified precision, instead of just rounding everything up or down.

For example, if someone asks you to round the numbers `1.23` and `1.28` to one decimal place, you would probably respond quickly with `1.2` and `1.3`. The `truncate()`, `round_up()`, and `round_down()` functions don't do anything like this.

Python >>>

```
>>> import pandas as pd

>>> # Re-seed np.random if you closed your REPL since the last example
>>> np.random.seed(444)

>>> series = pd.Series(np.random.randn(4))
>>> series
0    0.357440
1    0.377538
2    1.382338
3    1.175549
dtype: float64

>>> series.round(2)
0    0.36
1    0.38
2    1.38
3    1.18
dtype: float64

>>> df = pd.DataFrame(np.random.randn(3, 3), columns=["A", "B", "C"])
>>> df
          A         B         C
0 -0.939276 -1.143150 -0.542440
1 -0.548708  0.208520  0.212690
2  1.268021 -0.807303 -3.303072

>>> df.round(3)
       A      B      C
0 -0.939 -1.143 -0.542
1 -0.549  0.209  0.213
2  1.268 -0.807 -3.303
```

The `DataFrame.round()` method can also accept a dictionary or a `Series`, to specify a different precision for each column. For instance, the following examples show how to round the first column of `df` to one decimal place, the second to two, and the third to three decimal places:

Python >>>

```
>>> # Specify column-by-column precision with a dictionary
>>> df.round({"A": 1, "B": 2, "C": 3})
     A     B      C
0 -0.9 -1.14 -0.542
1 -0.5  0.21  0.213
2  1.3 -0.81 -3.303

>>> # Specify column-by-column precision with a Series
>>> decimals = pd.Series([1, 2, 3], index=["A", "B", "C"])
>>> df.round(decimals)
     A     B      C
0 -0.9 -1.14 -0.542
1 -0.5  0.21  0.213
2  1.3 -0.81 -3.303
```

If you need more rounding flexibility, you can apply NumPy's `floor()`, `ceil()`, and `rint()` functions to Pandas `Series` and `DataFrame` objects:

```
Python                                                    >>>
>>> np.floor(df)
     A    B    C
0 -1.0 -2.0 -1.0
1 -1.0  0.0  0.0
2  1.0 -1.0 -4.0

>>> np.ceil(df)
     A    B    C
0 -0.0 -1.0 -0.0
1 -0.0  1.0  1.0
2  2.0 -0.0 -3.0

>>> np.rint(df)
     A    B    C
0 -1.0 -1.0 -1.0
1 -1.0  0.0  0.0
2  1.0 -1.0 -3.0
```

The modified `round_half_up()` function from the previous section will also work here:

```
Python                                                    >>>
>>> round_half_up(df, decimals=2)
      A     B     C
0 -0.94 -1.14 -0.54
1 -0.55  0.21  0.21
2  1.27 -0.81 -3.30
```

Congratulations, you're well on your way to rounding mastery! You now know that there are more ways to round a number than there are taco combinations. (Well… maybe not!) You can implement numerous rounding strategies in pure Python, and you have sharpened your skills on rounding NumPy arrays and Pandas `Series` and `DataFrame` objects.

There's just one more step: knowing when to apply the right strategy.

Applications and Best Practices

The last stretch on your road to rounding virtuosity is understanding when to apply your newfound knowledge. In this section, you'll learn some best practices to make sure you round your numbers the right way.

Store More and Round Late

When you deal with large sets of data, storage can be an issue. In most relational databases, each column in a table is designed to store a specific data type, and numeric data types are often assigned precision to help conserve memory.

For example, a temperature sensor may report the temperature in a long-running industrial oven every ten seconds accurate to eight decimal places. The readings from this are used to detect abnormal fluctuations in temperature that could indicate the failure of a heating element or some other component. So, there might be a Python script running that compares each incoming reading to the last to check for large fluctuations.

The readings from this sensor are also stored in a SQL database so that the daily average temperature inside the oven can be computed each day at midnight. The manufacturer of the heating element inside the oven recommends replacing the component whenever the daily average temperature drops `.05` degrees below normal.

For this calculation, you only need three decimal places of precision. But you know from the incident at the Vancouver Stock Exchange that removing too much precision can drastically affect your calculation.

If you have the space available, you should store the data at full precision. If storage is an issue, a good rule of thumb is to store at least two or three more decimal places of precision

than you need for your calculation.

Finally, when you compute the daily average temperature, you should calculate it to the full precision available and round the final answer.

Obey Local Currency Regulations

When you order a cup of coffee for $2.40 at the coffee shop, the merchant typically adds a required tax. The amount of that tax depends a lot on where you are geographically, but for the sake of argument, let's say it's 6%. The tax to be added comes out to $0.144. Should you round this up to $0.15 or down to $0.14? The answer probably depends on the regulations set forth by the local government!

Situations like this can also arise when you are converting one currency to another. In 1999, the European Commission on Economical and Financial Affairs codified the use of the "rounding half away from zero" strategy when converting currencies to the Euro, but other currencies may have adopted different regulations.

Another scenario, "Swedish rounding", occurs when the minimum unit of currency at the accounting level in a country is smaller than the lowest unit of physical currency. For example, if a cup of coffee costs $2.54 after tax, but there are no 1-cent coins in circulation, what do you do? The buyer won't have the exact amount, and the merchant can't make exact change.

How situations like this are handled is typically determined by a country's government. You can find a list of rounding methods used by various countries on Wikipedia.

If you are designing software for calculating currencies, you should always check the local laws and regulations in your users' locations.

When In Doubt, Round Ties To Even

When you are rounding numbers in large datasets that are used in complex computations, the primary concern is limiting the growth of the error due to rounding.

Of all the methods we've discussed in this article, the "rounding half to even" strategy minimizes rounding bias the best. Fortunately, Python, NumPy, and Pandas all default to this strategy, so by using the built-in rounding functions you're already well protected!

Summary

Whew! What a journey this has been!

In this article, you learned that:

- There are various rounding strategies, which you now know how to implement in pure Python.
- Every rounding strategy inherently introduces a rounding bias, and the "rounding half to even" strategy mitigates this bias well, most of the time.
- The way in which computers store floating-point numbers in memory naturally introduces a subtle rounding error, but you learned how to work around this with the `decimal` module in Python's standard library.
- You can round NumPy arrays and Pandas `Series` and `DataFrame` objects.
- There are best practices for rounding with real-world data.

progress over time:

Take the Quiz »

If you are interested in learning more and digging into the nitty-gritty details of everything we've covered, the links below should keep you busy for quite a while.

At the very least, if you've enjoyed this article and learned something new from it, pass it on to a friend or team member! Be sure to share your thoughts with us in the comments. We'd love to hear some of your own rounding-related battle stories!

Happy Pythoning!

Additional Resources

Rounding strategies and bias:

- Rounding, Wikipedia
- Rounding Numbers without Adding a Bias, from ZipCPU

Floating-point and decimal specifications:

- IEEE-754, Wikipedia
- IBM's General Decimal Arithmetic Specification

Interesting Reads:

- What Every Computer Scientist Should Know About Floating-Point Arithmetic, David Goldberg, ACM Computing Surveys, March 1991
- Floating Point Arithmetic: Issues and Limitations, from python.org
- Why Python's Integer Division Floors, by Guido van Rossum

Mark as Completed

About **David Amos**

David is a writer, programmer, and mathematician passionate about exploring mathematics through code.

» More about David

Each tutorial at Real Python is created by a team of developers so that it meets our high quality standards. The team members who worked on this tutorial are:



 Adriana  Geir Arne  Joanna

What Do You Think?

Tweet Share Email

Real Python Comment Policy: The most useful comments are those written with the goal of learning from or helping out other readers—after reading the whole article and all the earlier comments. Complaints and insults generally won't make the cut here.

What's your #1 takeaway or favorite thing you learned? How are you going to put your newfound skills to use? Leave a comment below and let us know.

Keep Learning

Related Tutorial Categories: best-practices intermediate python