NUMPY RANDOM SEED EXPLAINED

by Sharp Sight | May 6, 2019

In this tutorial, I'll explain how to use the NumPy random seed function, which is also called np.random.seed or numpy.random.seed.

The function itself is extremely easy to use.

However, the *reason* that we need to use it is a little complicated. To understand *why* we need to use NumPy random seed, you actually need to know a little bit about pseudo-random numbers.

That being the case, this tutorial wil
random numbers, and will then mo
numpy.random.seed itself.

CONTENTS

The tutorial is divided up into severa

You can click on any of the above links, and it will take you directly to that section.

However, I strongly recommend that you read the whole tutorial.

As I said earlier, numpy.random.seed is very easy to use, but it's not that easy to understand. Understanding *why* we use it requires some background. That being the case, it's much better if you actually read the tutorial.

Ok ... let's get to it.

NUMPY RANDOM SEED IS FOR PSEUDO-RANDOM NUMBERS IN PYTHON

So what exactly is NumPy random seed?

NumPy random seed is simply a fu
the NumPy pseudo-random numb
input that enables NumPy to gener
random processes.

Does that make sense? Probably nc

Unless you have a background in c
wrote is probably a little confusing.

Honestly, in order to understand "se
you need to know a little bit about p

A QUICK INTRODUCTION TO PSEUDO-RANDOM NUMBERS

Here, I want to give you a very quick overview of pseudo-random numbers and why we need them.

Once you understand pseudo-random numbers, numpy.random.seed will make more sense.

WTF IS A PSEUDO-RANDOM NUMBER?

At the risk of being a bit of a smart-ass, I think the name "pseudo-random number" is fairly self explanatory, and it gives us some insight into what pseudo-random numbers actually are.

Let's just break down the name a little.

A pseudo-random number is a *number*. A number that's sort-of random. *Pseudo*-random.

So essentially, a pseudo-random nu
random, but not really random.

It might sound like I'm being a bit s
what they are. Pseudo-random nun
random, but are not actually rando

In the interest of clarity though, let's
little more precise.

A PROPER DEFINITION OF F

> ... a computer-generated random number.

The definition goes on to explain that ....

> The prefix pseudo- is used to distinguish this type of number from a "truly" random number generated by a random physical process such as radioactive decay.

A separate article at random.org notes that pseudo-random numbers "appear random, but they are really predetermined".

Got that? Pseudo-random numbers are computer generated numbers that appear random, but are actually predetermined.

I think that these definitions help quite a bit, and they are a great starting point for understanding why we need them

WHY WE NEED PSEUDO-RA

I swear to god, I'm going to bring th

But, we still need to understand wh
required.

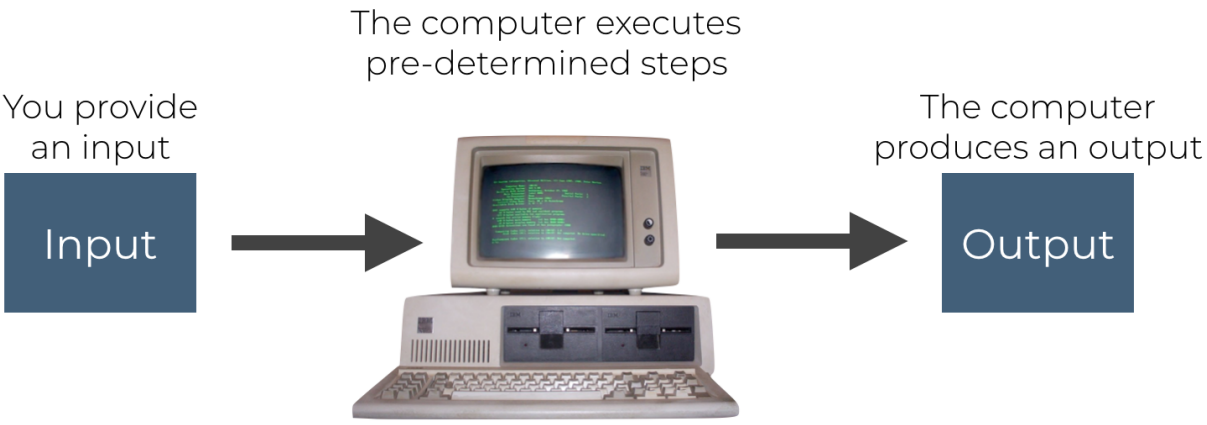
Really. Just bear with me. This will n

A PROBLEM ... COMPUTERS AR RANDOM

Computers are completely deterministic, not random.

Setting aside some rare exceptions, computers are deterministic by their very design. To quote an article at MIT's School of Engineering "if you ask the same question you'll get the same answer every time."

Another way of saying this is that if you give a computer a certain input, it will precisely follow instructions to produce an output.



... And if you later give a computer t
same output.

If the input is the same, then the ou

THAT'S HOW COMPUTERS WORK.

The behavior of computers is *deter*

Essentially, the behavior of compute

This introduces a problem: how can
produce random numbers?

GENERATED BY ALGORITHMS

Computers solve the problem of generating "random" numbers the same way that they solve essentially everything: with an algorithm.

Computer scientists have created a set of algorithms for creating psuedo random numbers, called "pseudo-random number generators."

These algorithms can be executed on a computer.

As such, they are completely deterministic. However, the numbers that they produce have properties that *approximate* the properties of random numbers.

PSEUDO-RANDOM NUMBERS APPEAR TO BE RANDOM

That is to say, the numbers generated by pseudo-random number generators *appear* to be random.

Even though the numbers they are
algorithm, when you examine them
pattern.

For example, here we'll create some
NumPy randint function:

```
np.random.seed(1)

np.random.randint(low = 1, high
```

OUT:

```
2, 4, 5, 9, 2, 5, 1, 4, 3, 1, 5, 3, 8, 8, 9, 7]
```

See any pattern here?

Me neither.

I can assure you though, that these numbers are not random, and are in fact completely determined by the algorithm. If you run the same code again, you'll get the exact same numbers.

PSEUDO-RANDOM NUMBERS CAN BE RE-CREATED EXACTLY

Importantly, because pseudo-random number generators are deterministic, they are also repeatable.

What I mean is that if you run the algorithm with the same input, it will produce the same output.

So you can use pseudo-random nu
re-create the exact same set of pseu

Let me show you.

GENERATE PSEUDO-RANDOM INTE

Here, we'll create a list of 5 pseudo-
using numpy.random.randint.

(And notice that we're using np.ran

```
np.random.randint(10, size = 5)
```

This produces the following output:

```
array([5, 0, 3, 3, 7])
```

Simple. The algorithm produced an array with the values `[5, 0, 3, 3, 7]`.

GENERATE PSEUDO-RANDOM INTEGERS *AGAIN*

Ok.

Now, let's run the same code again.

... and notice that we're using np.random.seed in exactly the same way ...

```
np.random.seed(0)
np.random.randint(10, size = 5)
```

OUTPUT:

```
array([5, 0, 3, 3, 7])
```

Well take a look at that ...

The. numbers. are. the. same.

We ran the exact same code, and it

I will repeat what I said earlier: pseu

produce numbers that look random

Remember what I wrote earlier: computers and algorithms process inputs into outputs. The outputs of computers depend on the inputs.

So just like any output produced by a computer, pseudo-random numbers are dependent on the *input*.

THIS is where numpy.random.seed comes in ...

The numpy.random.seed function provides the input (i.e., the seed) to the algorithm that generates pseudo-random numbers in NumPy.

HOW AND WHY WE USE NUMPY RANDOM SEED

Ok, you got this far.

You're ready now.

Now you can learn about NumPy random seed.

NUMPY.RANDOM.SEED PRO
TO THE PSEUDO-RANDOM I

What I wrote in the previous sectior

The "random" numbers generated l
They are pseudo-random ... they ap
100% determined by the input and
algorithm.

The np.random.seed function provi
number generator in Python.

It allows you to provide a "seed" value to NumPy's random number generator.

WE USE NUMPY.RANDOM.SEED IN CONJUNCTION WITH OTHER NUMPY FUNCTIONS

Importantly, numpy.random.seed doesn't exactly work all on its own.

The numpy.random.seed function works in *conjunction* with other functions from NumPy.

Specifically, numpy.random.seed works with other function from the `numpy.random` namespace.

So for example, you might use numpy.random.seed along with numpy.random.randint. This will enable you to create random integers with NumPy.

You can also use numpy.random.seed with numpy.random.normal to create normally distributed numbe

... or you can use it with numpy.ranc sample from an input.

In fact, there are several dozen Num you to generate random numbers, specific probability distributions.

I'll show you a few examples of som section of this tutorial.

NUMPY RANDOM SEED IS D

What this means is that if you provide the same seed, you will get the same output.

And if you change the seed, you will get a different output.

The output that you get depends on the input that you give it.

I'll show you examples of this behavior in the examples section.

NUMPY RANDOM SEED MAKES YOUR CODE REPEATABLE

The important thing about using a seed for a pseudo-random number generator is that it makes the code *repeatable*.

Remember what I said earlier?

... pseudo-random number generators operate by a deterministic process.

If you give a pseudo-random numb
get the same output.

This can actually be a good thing!

There are times when you really wa
repeatable.

Code that has well defined, repeata

Essentially, we use NumPy random
pseudo-random numbers in a repe

SHARE

The fact that np.random.seed makes your code repeatable also makes is easier to *share*.

Take for example the tutorials that I post here at Sharp Sight.

I post detailed tutorials about how to perform various data science tasks, and I show how code works, step by step.

When I do this, it's important that people who read the tutorials and run the code get the same result. If a student reads the tutorial, and copy-and-pastes the code exactly, I want them to get the exact same result. This just helps them check their work! If they type in the code exactly as I show it in a tutorial, getting the exact same result gives them confidence that they ran the code properly.

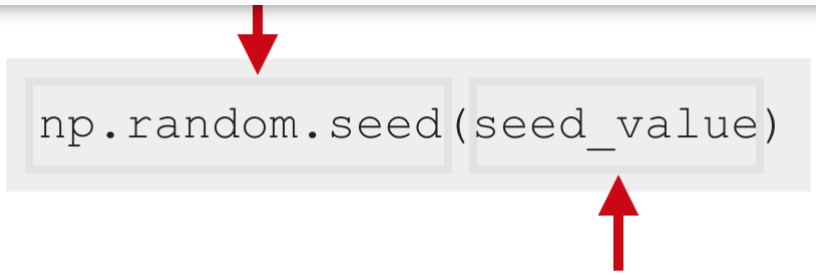
Again, in order to get repeatable results when we are using "random" functions in NumPy, we need to use numpy.random.seed.

Ok ... now that you understand wha
we use it), let's take a look at the act

THE SYNTAX OF NUMPY RANDOM

The syntax of NumPy random seed

There's essentially only one parame



So essentially, to use the function, you just call the function by name and then pass in a "seed" value inside the parenthesis.

Note that in this syntax explanation, I'm using the abbreviation "`np`" to refer to NumPy. This is a common convention, but it requires you to import NumPy with the code "`import numpy as np`." I'll explain more about this soon in the examples section.

EXAMPLES OF NUMPY.RANDOM.S

Let's take a look at some examples
numpy.random.seed.

Before we look at the examples tho

RUN THIS CODE FIRST

To get the following examples to ru
NumPy with the appropriate "nickn

You can do that by executing the fo

Running this code will enable us to use the alias `np` in our syntax to refer to `numpy`.

This is a common convention in NumPy. When you read NumPy code, it is extremely common to see NumPy referred to as `np`. If you're a beginner you might not realize that you need to import NumPy with the code `import numpy as np`, otherwise the examples won't work properly!

Now that we've imported NumPy properly, let's start with a simple example. We'll generate a single random number between 0 and 1 using NumPy random random.

GENERATE A RANDOM NUMBER WITH NUMPY.RANDOM.RANDOM

Here, we're going to use NumPy to generate a random number between zero and one. To do this, we're going to use the NumPy random random function (AKA, np.random.random).

Ok, here's the code:

```
np.random.seed(0)

np.random.random()
```

OUTPUT:

```
0.5488135039273248
```

Note that the output is a float. It's a

probabilities.

RERUN THE CODE

Now that I've shown you how to use np.random.random, let's just run it again with the same seed.

Here, I just want to show you what happens when you use np.random.seed before running np.random.random.

```
np.random.seed(0)
np.random.random()
```

OUTPUT:

```
0.5488135039273248
```

Notice that the number is exactly the same as the first time we ran the code.

Essentially, if you execute a NumPy

get the same result.

GENERATE A RANDOM
NUMPY.RANDOM.RAN

Next, we're going to use np.random

before using NumPy random randi

```
np.random.seed(74)
np.random.randint(low = 0, high = 100, size = 5)
```

OUTPUT:

```
array([30, 91,  9, 73, 62])
```

This is pretty simple.

NumPy random seed sets the seed for the pseudo-random number generator, and then NumPy random randint selects 5 numbers between 0 and 99.

RUN THE CODE AGAIN

Let's just run the code so you can see that it reproduces the same output if you have the same seed.

```
np.random.seed(74)
np.random.randint(low = 0, high
```

OUTPUT:

```
array([30, 91,  9, 73, 62])
```

Once again, as you can see, the cod
use the same seed. As noted previo
randint doesn't exactly produce "ra

SELECT A RANDOM SAMPLE FROM AN INPUT ARRAY

It's also common to use the NP random seed function when you're doing random sampling.

Specifically, if you need to generate a reproducible random sample from an input array, you'll need to use numpy.random.seed.

Let's take a look.

Here, we're going to use numpy.random.seed before we use numpy.random.choice. The NumPy random choice function will then create a random sample from a list of elements.

```
np.random.seed(0)
np.random.choice(a = [1,2,3,4,5,6], size = 5)
```

OUTPUT:

```
array([5, 6, 1, 4, 4])
```

As you can see, we've basically gene
of input elements ... the numbers 1 t

In the output, you can see that som
because np.random.choice is using
For more information about how to
read our tutorial about np.random.c

Let's quickly re-run the code.

I want to re-run the code just so you can see, once again, that the primary reason we use NumPy random seed is to create results that are completely repeatable.

Ok, here is the exact same code that we just ran (with the same seed).

```
np.random.seed(0)
np.random.choice(a = [1,2,3,4,5,6], size = 5)
```

OUTPUT:

```
array([5, 6, 1, 4, 4])
```

Once again, we used the same seed, and this produced the same output.

FREQUENTLY ASKED QUESTIONS

Now that we've taken a look at som
seed to set a random seed in Pytho
asked questions.

WHAT DOES NP.RAND

Dude. I just wrote 2000 words expla
function does ... which basically exp

Ok, ok ... I get it. You're probably in a

We use np.random.seed when we need to generate random numbers or mimic random processes in NumPy.

Computers are generally deterministic, so it's very difficult to create truly "random" numbers on a computer. Computers get around this by using pseudo-random number generators.

These pseudo-random number generators are algorithms that produce numbers that appear random, but are not really random.

In order to work properly, pseudo-random number generators require a starting input. We call this starting input a "seed."

The code `np.random.seed(0)` enables you to provide a seed (i.e., the starting input) for NumPy's pseudo-random number generator.

NumPy then uses the seed and the pseudo-random number generator in conjunction with other functions from the numpy.random namespace to produce certain types of random

Ultimately, creating pseudo-randon
repeatable output, which is good fo

Having said all of that, to really unde
need to have some understanding
generators.

... so if what I just wrote doesn't mak
the page and read the f*#^ing tutor

Basically, it doesn't matter.

You can use `numpy.random.seed(0)`, or `numpy.random.seed(42)`, or any other number.

For the most part, the number that you use inside of the function doesn't really make a difference.

You just need to understand that using different seeds will cause NumPy to produce different pseudo-random numbers. The output of a `numpy.random` function will depend on the seed that you use.

Here's a quick example. We're going to use NumPy random seed in conjunction with NumPy random randint to create a set of integers between 0 and 99.

In the first example, we'll set the seed value to 0.

```
np.random.seed(0)
np.random.randint(99, size = 5)
```

Which produces the following outp

```
array([44, 47, 64, 67, 67])
```

Basically, np.random.randint genera
and 99. Note that if you run this coc
0), you'll get the same integers from

Next, let's run the code with a *differ*

```
np.random.randint(99, size=5)
```

OUTPUT:

```
array([37, 12, 72,  9, 75])
```

Here, the code for np.random.randint is exactly the same … we only changed the seed value. Here, the seed is `1`.

With a *different* seed, NumPy random randint created a *different* set of integers. Everything else is the same. The code for np.random.randint is the same. But with a different seed, it produces a different output.

Ultimately, I want you to understand that the output of a numpy.random function ultimately depends on the value of np.random.seed, but the choice of seed value is sort of arbitrary.

DO I ALWAYS NEED TO USE NUMPY RANDOM SEED?

The short answer is, no.

If you use a function from the `numpy` np.random.randint, np.random.norm random see first, Python will actual background. NumPy will generate a computer system (like `/urandom` on

So essentially, if you don't set a seed will set one for you.

If you don't explicitly set a seed, your code will not have repeatable outputs. NumPy will generate a seed on its own, but that seed might change moment to moment. This will make your outputs different every time you run it.

So to summarize: you don't absolutely have to use numpy.random.seed, but you *should* use it if you want your code to have repeatable outputs.

WHAT'S THE DIFFERENCE BETWEEN NP.RANDOM.SEED AND NP.RANDOM.RANDOMSTATE?

Ok.

We're really getting into the weeds here.

Essentially, numpy.random.seed sets a seed value for the global instance of the numpy.random namespace.

On the other hand, np.random.Ran
RandomState and does not effect t

Confused?

That's okay .... this answer is a little t
little about how NumPy is structure
you to know a little bit about progra
variables." If you're a relative data sc
need to know might be over your h

However, if you're building software systems that need to be secure, NumPy random seed is probably not the right tool.

To summarize, np.random.seed is probably fine if you're just doing simple analytics, data science, and scientific computing, but you need to learn more about RandomState if you want to use the NumPy pseudo-random number generator in systems where security is a consideration.

APPLICATIONS OF NP.RANDOM.SEED

Now that I've explained the basics of NumPy random seed, I want to tell you a few applications ...

Here's where you might see the np.random.seed function.

PROBABILITY AND STATISTICS

It's possible to do probability and st

Almost by definition, probability inv
As such, if you use Python and Num
you'll need to use np.random.seed t
(or a similar tool in Python).

RANDOM SAMPLING

More specifically, if you're doing ran
need to use numpy.random.seed.

random choice.

In almost every case, when you use one of these functions, you'll need to use it in conjunction with numpy random seed if you want to create reproducible outputs.

MONTE CARLO METHODS

Monte Carlo methods are a class of computational methods that rely on repeatedly drawing random samples.

I won't go into the details here, since Monte Carlo methods are a little complicated, and beyond the scope of this post.

Essentially though, Monte Carlo methods are a powerful computational tool used in science and engineering. In fact, Monte Carlo methods were initially used at the Manhattan Project!

Monte Carlo methods require rando
these methods are used, they actua
instead of true random numbers.

MACHINE LEARNING

Interested in machine learning?

Great ... it's a powerful toolset, and it
21st century.

Broadly speaking, pseudo-random
learning.

requires pseudo-random numbers.

So if you're doing machine learning in Python, you'll almost certainly need to use NumPy random seed ...

DEEP LEARNING

More specifically, you'll also probably use pseudo-random numbers if you want to do deep learning.

For example, if you want to do deep learning in Python, you'll often need to split datasets into training and test sets (just like with other machine learning techniques). Again, this requires pseudo-random numbers.

... so when people do deep learning in Python, you'll frequently see at least a few uses of numpy.random.seed.

LEARN NUMPY TO LEARN DATA S

I've really only touched on a few app
Python. There are many more.

Speaking generally, if you want to u
this little function.

But even though we focused on Nu
there are many other NumPy funct

If you want to learn how to do data
important ...

our email list.

Here at Sharp Sight, we teach data science.

... and we regularly post FREE data science tutorials just like this one.

If you want to get our free tutorials delivered directly to you email inbox, then sign up now.

If you sign up for our email list, you'll get tutorials about:

- NumPy
- Pandas
- Matplotlib
- Seaborn
- Sci-kit learn
- Machine learning
- Deep learning
- ... and more

We also teach data science in R, so i
both languages.

So if you want to learn more data sc

7 COMMENTS

Adinarayana on May 8, 2019 at 11:30 PM

It's really a very nice tutorial for the beginners.

Reply

Naeem on October 28, 20

Very clear and helpfu

3akram on November 29,

Great f*^&ing tutorial

Sharp Sight on N

Chen on December 25, 2019 at 8:29 PM

so patient, thank you

Reply

Abhay on January 12, 2020 at 9:17 AM

Awesome insights on Seed. I got really clear about it after this explanation.

Reply

Sharp Sight on January 12, 2020 at 11:10 AM

👍👍👍