assignment2020

November 21, 2020

1 Higher Diploma in Science in Computing (Data Analytics)

1.1 #### Programme Module: Programming for Data Analysis (COMP08050)

1.1.1 Assignment 2020 - numpy.random

The following assignment concerns the numpy.random package in Python. You are required to create a Jupyter notebook explaining the use of the package, including detailed explanations of at least five of the distributions provided for in the package.

There are four distinct tasks to be carried out in your Jupyter notebook. 1. Explain the overall purpose of the package. 2. Explain the use of the "Simple random data" and "Permutations" functions. 3. Explain the use and purpose of at least five "Distributions" functions. 4. Explain the use of seeds in generating pseudorandom numbers.

1.1.2 Table of contents

2.1.2 Random

2.1.3 Choice

2.1.4 Bytes

2.2 Permutations

3. Explain the use and purpose of at least five "Distributions" functions

3.1 Uniform distribution

3.2 Normal distribution

3.3 Exponential distribution

3.4 Binomial distribution

3.5 Poisson distribution

4. Explain the use of seeds in generating pseudorandom numbers.

5. References

1.1.3 1. Explain the overall purpose of the package

Numerical Python more commonly referred to as NumPy is an open source Python library created in 2005 by Travis Oliphant. It contains multi-dimensional array and matrix data structures. Multi-dimensional arrays have more than one column (dimension), consider it like an excel spreadsheet. (Malik, 2020,(5)).

NumPy also has a large collection of high-level mathematical functions to operate on these arrays. One of the main uses of NumPy is its use in data analysis *"as a container for data to be passed between algorithms and libraries".*(McKinney, 2018,(13)). These capabilities mean *"many numerical computing tools for Python either assume NumPy arrays as a primary data structure or else target seamless interoperability with NumPy".*(McKinney, 2018,(13)). Examples include Scikit-Learn, Scipy and Keras which make extensive use of NumPy.

Before explaining the overall purpose of the numpy.random package let's have a quick overview of what random numbers are. Random refers to something that cannot be predicted logically. Randomness is useful in many areas such as simulating the impact of chance on stock markets or in the selection of representative samples of patients when testing new drugs. (Matthews, 2020,(6)). However, there is a problem when using randomness for making unbiased choices and that comes down to bias. *"The lack of bias only really appears in an infinitely long set of random numbers. In any given collection, there can be astonishingly long patterns".* (Matthews, 2020,(6))

There are two types of random number.

1. True-Random: Truly random number sequences are generated by chance that contain no recognisable pattern or regularity. (Spacey, 2016,(8)).
2. Pseudo-Random: generated by computers, are not random as they are deterministic devices i.e., they are predictable by design. *"So, to create something unpredictable, they use mathematical algorithms to produce numbers that are random enough".* (ComputerHope, 2019,(14))

The numpy.random package can perform a few different tasks: you can use it to generate random numbers, randomize lists, or choose elements from a list at random.

The package is also perfect for generating passwords or producing test datasets. It can also be integrated into Python for or if loops to change the outcome of a function at random.

The NumPy random module adds to the already built in Python random *"with functions for efficiently generating whole arrays of sample values from many kinds of probability distributions"*. (McKinney, 2018,(13)).

Numpy random and Python random although sharing the same algorithm work in different ways. In terms of efficiency, NumPy is most likely to perform better because arrays can be created without the need of a loop. (DiTect, 2020,(7)).

Note: The algorithm used by NumPy has now changed. Previously NumPy used the Mersenne Twister(MT19937) as the core generator but with the introduction of the latest version 1.19 the core generator is now the Permuted Congruential Generator (64bit,PCG64).

The Mersenne Twister was developed in 1997 by Makoto Matsumoto and Takuji Nishimura to try and rectify issues in older PRNG's. However, it is now considered to be not particularly fast and predictable, it is also not very space efficient. The PCG64 generator is considered faster and space efficient not requiring excessive memory to run. It is also challenging to predict but this does not mean it should be considered cryptographically secure.

1.1 Import required libraries Below are the libraries I use in creating arrays and visualisations throught this notebook.

[1]:
```
# import required libraries
import numpy as np

# import seaborn
import seaborn as sns

#import matplotlib
import matplotlib.pyplot as plt

# used so that the output of plotting commands is displayed inline with␣
 ↪frontend of Jupyter Notebook
%matplotlib inline
```

1.2 Notes Referencing: For references I have used the following style in this document. (Author, Year(Reference number)). For example (Matthews, 2020,(6)). 6 in this case refers to number 6 in the References section.

NumPy RNG: numpy.random(functions) are now a legacy function as of NumPy 1.17. NumPy 1.17 introduced a new random number generation system. The old functions in the numpy.random namespace will continue to work, but they are considered "frozen", with no ongoing development.

Syntax: For the code sections in this notebook I will be using the syntax in NumPy 1.19.0 which supports Python versions 3.6-3.8.

Seeding: In the examples I use I have set the seed in the numpy.random number generator, which is explained in section 4 of this notebook.

Plotting: Where I have used plots to visualize code outputs, I have used Seaborn distplots. This function combines the matplotlib hist function (with automatic calculation of a good default bin size) with the seaborn kdeplot() and rugplot() functions. In some cases I have not used the hist function and only used the kernel density estimate and rugplot for stylistic reasons.

Note: Where you see a green note box this refers to real world examples or usage of the concepts in this notebook.

Note: Throughout this notebook you will see yellow note boxes, these will refer you to the NumPy help sections where you can find additional information. Just run the piece of code in your own code cells to get the information.

1.1.4 2. Explain the use of the "Simple random data" and "Permutations" functions

2.1 Simple random data There are various ways of creating arrays with random data in NumPy. The random.module provides convenient methods for doing this with the desired shape and distribution. "*You will find that most of the random functions have several variants that do more or less the same thing. They might vary in minor ways - parameter order, whether the value range is inclusive or exclusive etc. The basic set described below should be enough to do everything you need, but if you prefer to use the other variants, they will deliver the same results*".(PythonInformer, 2019,(35))

2.1.1 Integers Replaces `RandomState.randint` and `RandomState.random_integers`. It creates an array of integers. At its most basic it creates values in the range 0 to high i.e. integers from 0 up to but not including high. In the below example with a value of 4, it creates values in the range 0 to 3. Note that size is passed in as a named paramter.

Syntax: integers(low, high=None, size=None, dtype=np.int64, endpoint=False)

[2]:
```
rng = np.random.default_rng(7)
rng.integers(4, size=(3, 4))
```

[2]:
```
array([[3, 2, 2, 3],
       [2, 3, 3, 0],
       [0, 1, 1, 3]], dtype=int64)
```

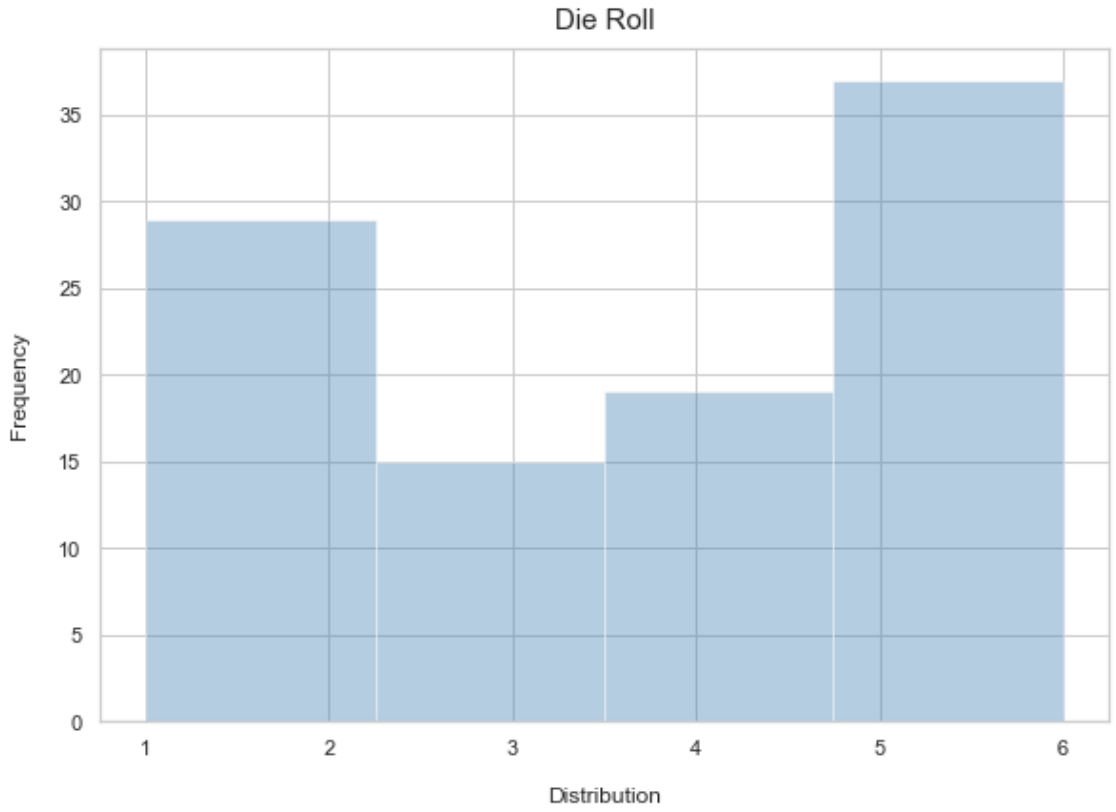
You can also pass in two values, low and high, resulting in numbers in the range (low, high). For example to simulate a dice (output values 1 to 6 inclusive), you would use values 1 and 7:

[3]:
```
rng = np.random.default_rng(7)
x = rng.integers(1, 7, size=(100))
```

Lets visualise the above where we roll a die 100 with a histogram to review the results.

[4]:
```
# visualise styling code
sns.set(rc={'figure.figsize': (9.5, 6.5)})
```

```
sns.set_context('notebook')
sns.set_style("whitegrid")

# plotting
sns.distplot(x, kde=False, hist=True, kde_kws={
             "shade": True}, color='steelblue', rug=False)
plt.xlabel("Distribution", labelpad=15)
plt.ylabel("Frequency", labelpad=15)
plt.title("Die Roll", fontsize=15, y=1.012);
```



Further information: np.info(rng.integers)

2.1.2 Random This function produces an array of floating point random numbers in the range 0.0 up to but not including 1.0. Remember this means the range will include anything from 0.0 up to the largest float that is less than 1 for example 0.99999999…. it will never include 1. The values will also be in a uniform distribution, meaning every value is equally likely to occur.

Syntax: random(size=None, dtype=np.float64, out=None)

[5]:
```
rng = np.random.default_rng(7)
rng.random((3, 2))
```

[5]:
```
array([[0.62509547, 0.8972138 ],
       [0.77568569, 0.22520719],
       [0.30016628, 0.87355345]])
```

Further information: np.info(rng.random)

2.1.3 Choice TheNumPy random choice function accepts an input of elements, chooses randomly from those elements, and outputs the random selections as a NumPy array. Because the output of numpy. random.

Syntax: choice(a, size=None, replace=True, p=None, axis=0)

[6]:
```
array = np.arange(start=0, stop=10)
```

Lets see what this does.

[7]:
```
print(array)
```

```
[0 1 2 3 4 5 6 7 8 9]
```

As expected it generates a list of integers between 0 and 9. Now lets select a random number from here.

[8]:
```
rng = np.random.default_rng(7)
rng.choice(a=array)
```

[8]:
```
9
```

In this instance it returned 9. You can also produce this outcome by using a shorter syntax.

[9]:
```
rng = np.random.default_rng(7)
rng.choice(10)
```

[9]:
```
9
```

In this example, when we ran the code a specific NumPy array was not provided as an input. Instead, the number 10 was provided. "*When we provide a number this way, it will automatically create a NumPy array using NumPy arange. Effectively, the code is identical to the code (array = np.arange(start = 0, stop = 10). So by running rng.choice this way, it will create a new numpy array of values from 0 to 9 and pass that as the input to rng.choice. This is essentially a shorthand way to both create an array of input values and then select from those values using the NumPy random choice function.*" (Ebner, 2019,(23))

Further information: np.info(rng.choice)

2.1.4 Bytes This function generates pseudorandom bytes. It only has one parameter (length).

Syntax: bytes(length)

[10]:
```
rng = np.random.default_rng(7)
b = rng.bytes(10)
print(b)
```

```
b'\x8bJ\xe5\xf1\xa9A\x06\xa0\x95j'
```

Further information: np.info(rng.bytes)

2.2 Permutations Permutation is a mathematical term and permutation of a set is defined as the arrangement of it's elements in a sequence or a linear order. If it is already arranged then permutation is the rearrangement of its elements in another sequence. The number of permutations of a specified data set can be calculated using a mathematical formula however NumPy provides two built in functions to do this namely the `permutation()` function and the `shuffle()` function. They are very similar; the functions are the same but they are different.

In `numpy.random` the permutation as already stated is built in. The permutation provides an array as an output. However, it doesn't offer all the permutations of the array *"but only one in which we can find that the elements of the array have been re-arranged"*. (Sourcecodester, 2020,(16)). The permutation function returns a re-arranged array and leaves the original unchanged. So, the original array is intact and will return a shuffled array. If we have an array x=[1, 4, 2, 8] as the original the permutation may return a re-arranged array, say [8, 2, 1, 4]. Let's look at an example to explain this further.

Syntax: permutation(x, axis=0)

[11]:
```
# create a numpy array
a = np.arange(12)
print("Original array:\n", a)

# permute the results
rng = np.random.default_rng(7)
b = rng.permutation(a)
print("Permuted array:\n", b)
print("Original array:\n", a)
```

```
Original array:
 [ 0  1  2  3  4  5  6  7  8  9 10 11]
Permuted array:
 [ 4  6 10  0  1  3  8  7  2  5  9 11]
Original array:
 [ 0  1  2  3  4  5  6  7  8  9 10 11]
```

Further information: np.info(rng.permutation)

In the above array you can see that the original array is unchanged. Now lets look at the how the `shuffle()` function works.

Syntax: shuffle(x, axis=0)

[12]:
```
# create a numpy array
a = np.arange(12)
print("Original array:\n", a)

# shuffle the results
rng = np.random.default_rng(7)
```

```
rng.shuffle(a)
print("Shuffled array:\n", a)
```

```
Original array:
 [ 0  1  2  3  4  5  6  7  8  9 10 11]
Shuffled array:
 [ 4  6 10  0  1  3  8  7  2  5  9 11]
```

`Permutation` does not directly operate on the original array, but returns a new array of scrambled orders, without changing the original array whereas the `shuffle` function directly operates on the original array, changing the order of the original array, no return value.(ProgrammerSought, 2020, (18))

Further information: np.info(rng.shuffle)

1.1.5 3. Explain the use and purpose of at least five "Distributions" functions

3.1 Uniform distribution Notation:

$$p(x) = \frac{1}{b-a}$$

Uniform distribution is a type of probability distribution in which all outcomes are equally likely. For example, a coin toss has a uniform distribution, because the probability of getting heads or tails is the same. A uniform distribution can also be discrete or continuous.

A discrete uniform distribution has a finite number of outcomes, for example the probability of landing on each side of a die. (Investopedia, 2020,(19)). Alternatively, the continuous uniform distribution (also referred to as a rectangular distribution) is a statistical distribution with an infinite number of equally likely measurable values. (CFI,2020,(20)).

For example 0.0 and 1.0, every point in the continuous range between these has an equal opportunity of appearing, yet there are an infinite number of points between them.

Real world usage: From a theoretical perspective, uniform distribution is a key one in risk analysis; many Monte Carlo software algorithms use a sample from this distribution (between zero and one) to generate random samples from other distributions (by inversion of the cumulative form of the respective distribution).

Discrete uniform distributions can be valuable for businesses such as in inventory management in the study of the frequency of inventory sales. It can provide a probability distribution that can guide the business on how to properly allocate the inventory for the best use of square footage.

The syntax of np.random.default_rng().uniform is;

np.random.default_rng().uniform(low= , high= , size =)

1. `np.random.default_rng().uniform` is the function name.
2. `low` is the minimum.
3. `high` is the maximum.

4. `size` is the shape of the output NumPy array.

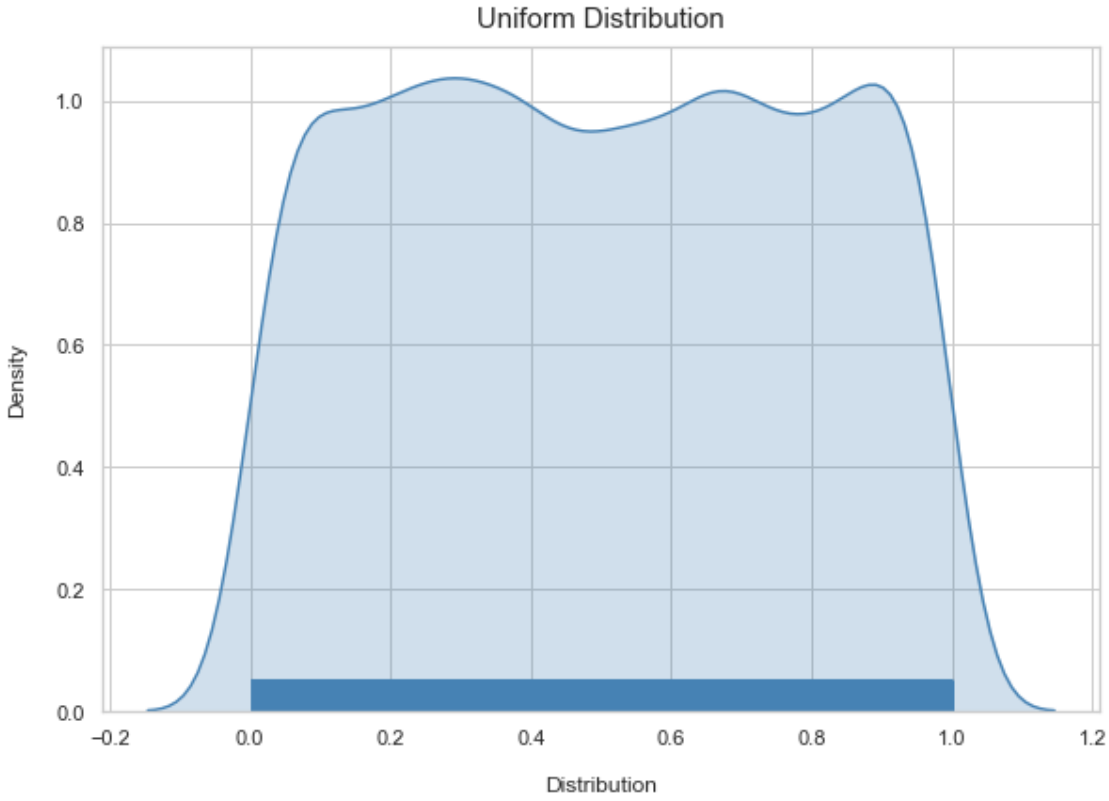
The below examples illustrate the creation of uniform distributions.

[13]:
```
# draw 10 values from the range [0,1)
example1 = np.random.default_rng(7).uniform(0, 1, 10)

# create a 2-dimensional array of uniformly distributed numbers
example2 = np.random.default_rng(7).uniform(0, 1, (10, 1000))

# create an array of values from a specific range
example3 = np.random.default_rng(7).uniform(50, 60, 10000)

# print the arrays
print("Example1 Uniform distribution - values from a range:\n", example1)
print("Example2 Uniform distribution - 2 dimensional array:\n", example2)
print("Example3 Uniform distribution - specific range:\n", example3)
```

```
Example1 Uniform distribution - values from a range:
 [0.62509547 0.8972138  0.77568569 0.22520719 0.30016628 0.87355345
 0.0052653  0.82122842 0.79706943 0.46793495]
Example2 Uniform distribution - 2 dimensional array:
 [[0.62509547 0.8972138  0.77568569 … 0.17795331 0.97015595 0.2027232 ]
 [0.86904976 0.87787367 0.62190832 … 0.85235258 0.12209217 0.06502189]
 [0.23157976 0.49258688 0.94215871 … 0.62854247 0.44125798 0.95921486]
 …
 [0.08814085 0.30635689 0.62623328 … 0.27680629 0.82155355 0.61154437]
 [0.74428164 0.73879575 0.90550973 … 0.5418332  0.9300833  0.50630729]
 [0.26153447 0.40107793 0.65992693 … 0.87604993 0.47684568 0.73355636]]
Example3 Uniform distribution - specific range:
 [56.25095467 58.97213801 57.7568569  … 58.7604993  54.76845678
 57.33556364]
```

Now lets visualise the distribution of example3.

[14]:
```
# visualise styling code
sns.set(rc={'figure.figsize': (9.5, 6.5)})
sns.set_context('notebook')
sns.set_style("whitegrid")

# plotting
sns.distplot(example2, kde=True, hist=False, kde_kws={
             "shade": True}, color='steelblue', rug=True)
plt.xlabel("Distribution", labelpad=15)
plt.ylabel("Density", labelpad=15)
plt.title("Uniform Distribution", fontsize=15, y=1.012);
```



3.2 Normal distribution Notation:

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Normal distribution (Gaussian) in probability theory is a type of continuous probability distribution for a real valued random variable. The data can be distributed or spread out in different ways i.e. it can be more to the left, the right or it can be jumbled up. However, there are numerous instances "*where the data tends to be around a central value with no bias to the left or right and it gets close to a normal distribution*". (MathsIsFun, 2020,(22)). This normally distributed data is shaped sort of like a "bell curve".

Real world usage: The normal distribution is the most important probability distribution in statistics because it fits many natural phenomena. For example, heights, blood pressure, measurement error, and IQ scores follow the normal distribution. It is also known as the Gaussian distribution and the bell curve

The syntax for np.random.default_rng().normal is is below

np.random.default_rng().normal(loc= , scale = , size=)

1. `np.random.default_rng().normal` is the function name.
2. `loc` is the mean ("centre") of the distribution.
3. `scale` is the standard deviation (spread or "width") of the distribution. Must be non-negative.

4. `size` is the shape of the output NumPy array.

The below examples illustrate the creation of a normal distribution. I have added `.mean()` to example 1 to show that results generated from choosing a `loc = 50` are in or around the mean.

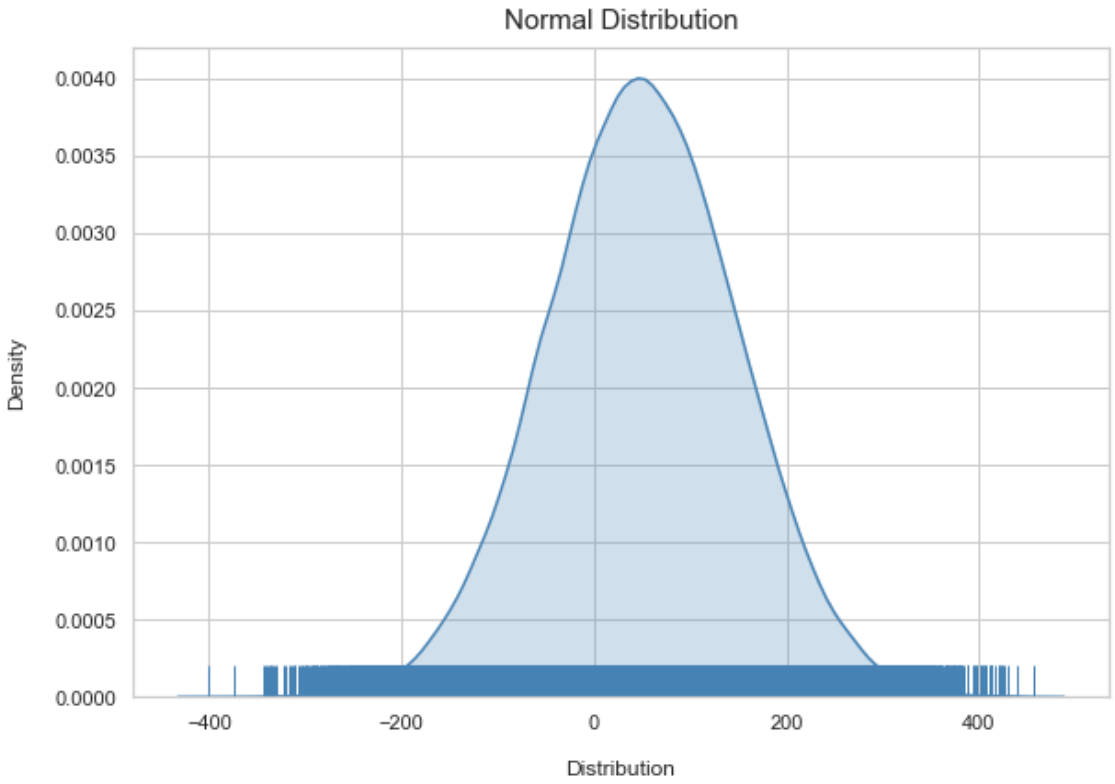
For example 2 I added `.std()` to show results are calculated based on a `scale = 100` and are in or around 100.

[15]:
```
# generate normally distributed values with a specific mean
example1 = np.random.default_rng(7).normal(loc=50, size=100000).mean()

# generate normally distributed values with a specific standard deviation
example2 = np.random.default_rng(7).normal(scale=100, size=100000).std()

# lets combine all the elements of the syntax
example3 = np.random.default_rng(7).normal(loc=50, scale=100, size=100000)

# print the arrays
print("Example1 Normal distribution - distributed values with a specific mean:
 ↪\n", example1)
print("Example2 Normal distribution - distributed values with a specific␣
 ↪standard deviation:\n", example2)
print("Example3 Normal distribution:\n", example3)
```

```
Example1 Normal distribution - distributed values with a specific mean:
 49.99867368091261
Example2 Normal distribution - distributed values with a specific standard
deviation:
 99.82966974374132
Example3 Normal distribution:
 [ 50.12301534  79.87455375  22.58621446 …  11.55742105 -28.97114874
 135.18898525]
```

Now lets visualise the distribution.

[16]:
```
# visualisation styling code
sns.set(rc={'figure.figsize': (9.5, 6.5)})
sns.set_context('notebook')
sns.set_style("whitegrid")

# plotting
sns.distplot(example3, kde=True, hist=False, kde_kws={
             "shade": True}, color='steelblue', rug=True)
plt.xlabel("Distribution", labelpad=15)
plt.ylabel("Density", labelpad=15)
plt.title("Normal Distribution", fontsize=15, y=1.012);
```



In a normal distribution, 68% of the data set will lie within ±1 standard deviation of the mean. 95% of the data set will lie within ±2 standard deviations of the mean. And 99.7% of the data set will lie within ±3 standard deviations of the mean. This is called the 68–95–99.7 rule.

3.3 Exponential distribution Notation:

$$f\left(x; \frac{1}{\beta}\right) = \frac{1}{\beta} \exp\left(-\frac{x}{\beta}\right)$$

Exponential distribution is a widely used continuous distribution. It is often used to model the time elapsed between events. For example, the amount of time (beginning now) until an earthquake might occur. In physics it is used to measure radioactive decay or in engineering measuring the time associated with receiving a defective part on the production line. In finance it is used to "*measure the likelihood of incurring a specified number of defaults within a specified time period*". (Science Direct, 2017,(26)).

Real world usage: The exponential distribution can be used when studying failure rates and the reliability of production lines in factories.

The syntax of random.default_rng().exponential is below

np.random.default_rng().exponential(scale= , size =)

1. `np.random.exponential` is the function name.
2. `scale` is the inverse of the rate which is by default set to 1.0.

3. `size` is the number of values drawn from an exponential distribution.

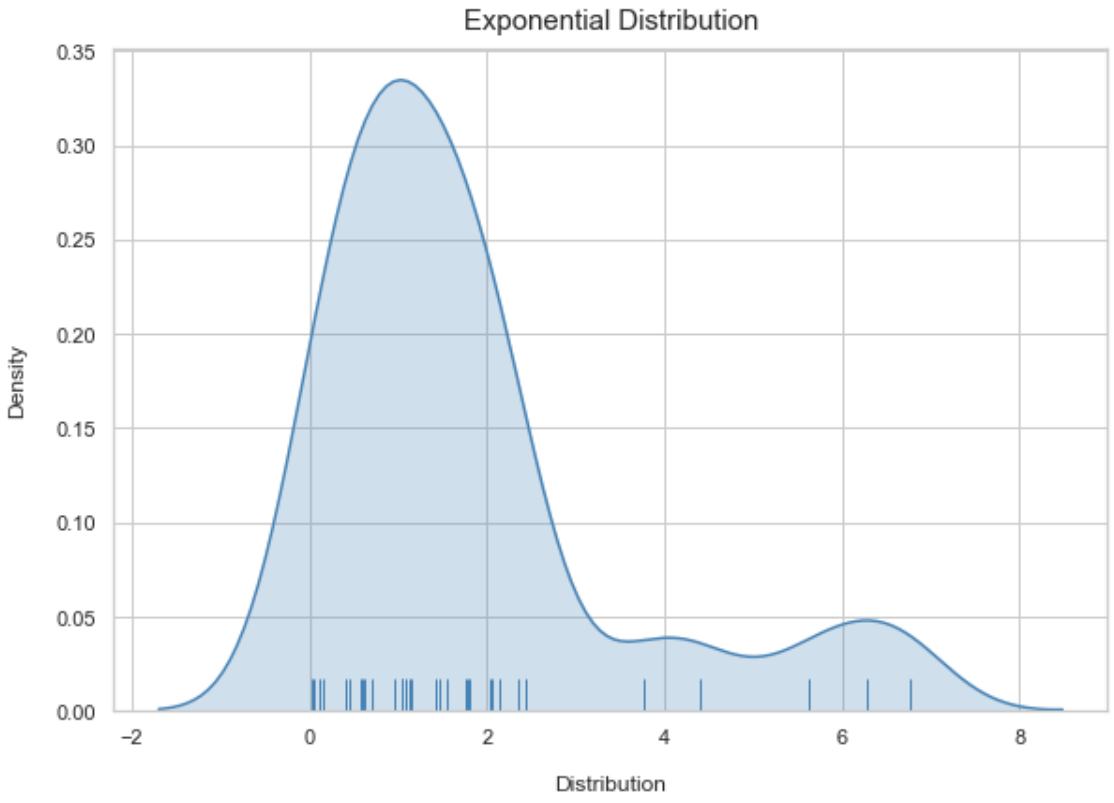
We can use random.exponential() to get a exponential distribution.

[17]:
```
# create a 2-dimensional array of exponentially distributed numbers
example1 = np.random.default_rng(7).exponential(scale=2, size=(4, 8))

# print the array
print("Exponential distribution:\n", example1)
```

```
Exponential distribution:
 [[1.41505851 2.0504067  1.13709731 1.79021973 0.41306551 6.7672747
  0.01950725 5.61843153]
 [1.15066551 0.60106803 1.08227179 0.62429122 1.79954031 2.14740132
  3.7685001  0.44414252]
 [6.28934664 1.47171455 0.6967458  1.76713024 0.15012301 0.12009269
  2.44713284 1.5458113 ]
 [4.3919296  0.95030781 1.04263637 2.36151039 1.05307088 0.04433827
  0.58655753 2.02851733]]
```

Time to visualise how this looks.

[18]:
```
# visualisation styling code
sns.set(rc={'figure.figsize': (9.5, 6.5)})
sns.set_context('notebook')
sns.set_style("whitegrid")

# plotting
sns.distplot(example1, kde=True, hist=False, kde_kws={
             "shade": True}, color='steelblue', rug=True)
plt.xlabel("Distribution", labelpad=15)
plt.ylabel("Density", labelpad=15)
plt.title("Exponential Distribution", fontsize=15, y=1.012);
```



3.4 Binomial distribution Notation:

$$P(N) = \begin{pmatrix} n \\ N \end{pmatrix} p^N (1-p)^{n-N}$$

Binomial distribution is the probability distribution of a sequence of experiments, where each experiment produces a binary outcome and where each of the outcomes is independent of all others. (Towards Data Science, 2019). In simple terms it is the probability of a of a success or failure outcome. (Statistics How to, 2020,(29)). The distribution of these binary scenarios is obtained by performing a number of "Bernolli" trials, which are assumed to meet each of the following criteria: (a) only two possible outcomes (b) each outcome has a fixed probability of occurring and (c) each trial is completely independent of all others. (GeeksforGeeks, 2020,(27)).

Real world usage: In real life binomial distributions are found everywhere. For example, if a new drug is introduced to cure a disease, it either cures the disease (it's successful) or it doesn't cure the disease (it's a failure). If you purchase a lottery ticket, you're either going to win money, or you aren't. Basically, anything you can think of that can only be a success or a failure can be represented by a binomial distribution. (Statistics How to, 2020(29)).

The syntax of np.random.default_rng().binomial is

np.random.default_rng().binomial (n= , p = , size=)

1. `np.random.default_rng().binomial` is the function name.

2. `n` is the number of trials.
3. `p` is the probability of occurrence.
4. `size` is the shape of returned array.

We can use random.default_rng().binomial to get a binomial distribution.

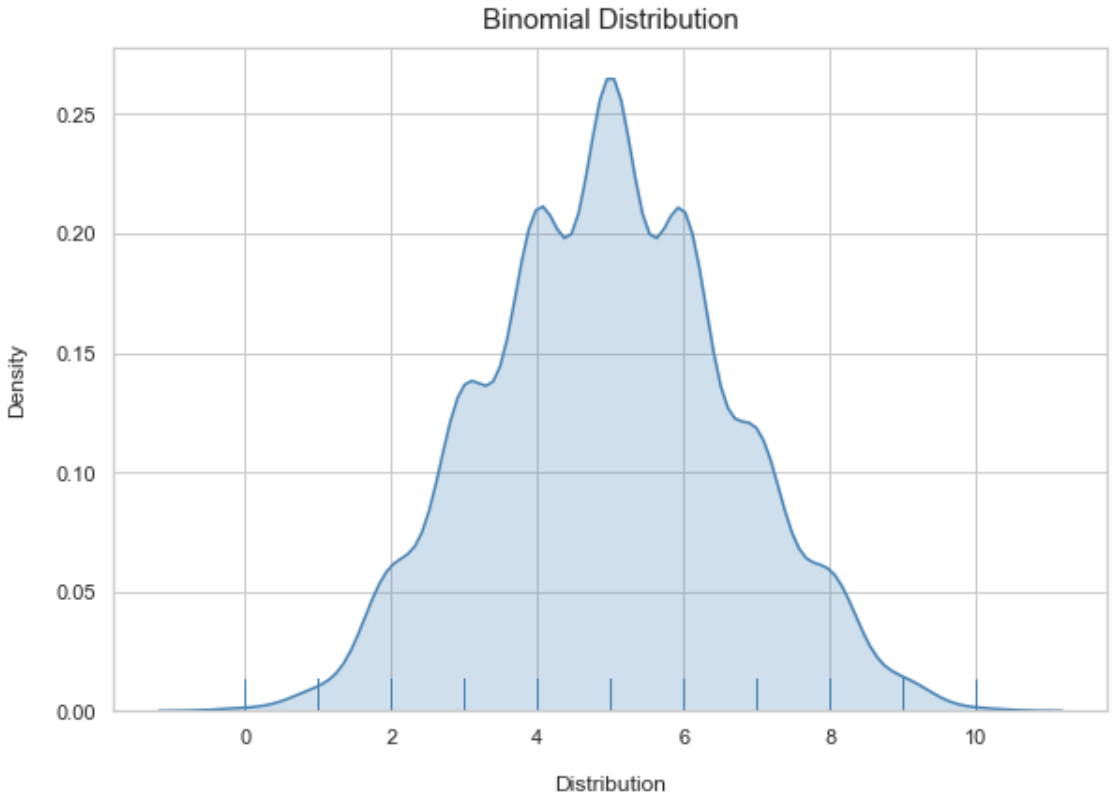
[19]:
```
# create the variables and assign them
example1 = np.random.default_rng().binomial(10, 0.5, 1000)

# print the array
print("Binomial distribution:\n", example1)
```

```
Binomial distribution:
 [ 5  4  6  8  4  6  6  5  6  2  4  6  2  4  5  6  3  5  3  8  5  5  3  4
  7  4  6  5  6  7  3  4  2  3  6  5  4  6  3  2  6  4  4  3  5  4  5  6
  3  5  6  4  3  4  5  8  6  3  2  5  5  4  4  7  4  7  4  5  3  4  6  9
  6  4  7  6  4  4  5  5  5  3  6  7  5  6  5  3  5  7  6  1  4  6  2  4
  5  4  6  4  7  2  5  2  5  4  6  2  5  4  1  9  4  3  5  6  7  5  5  7
  4  5  6  4  7  4  4  4  5  7  6  4  5  7  4  3  5  5  4  4  7  6  5  6
  9  3  4  1  4  7  6  4  3  4  5  3  5  5  4  7  4  4  4  6  5  3  6  5
  8  6  5  5  8  4  4  5  6  6  5  2  2  6  5  5  5  6  6  8  8  5  5  6
  6  6  5  7  7  5  7  3  7  4  5  2  5  5  2  5  6  6  4  6  8  3  5  2
  4  7  6  6  6  3  6  5  6  5  6  8  5  6  4  6  3  7  6  5  6  6  9  7
  8  5  8  3  5  4  6  4 10  5  3  5  4  4  4  2  6  5  5  3  8  6  4  4
  7  6  6  4  8  4  6  4  6  8  6  4  6  2  6  5  2  3  6  8  4  5  3  6
  5  6  3  5  5  3  2  5  3  6  5  8  6  4  5  8  6  6  4  5  4  2  7  5
  7  5  5  3  3  4  2  4  6  5  5  3  4  5  7  4  5  3  4  5  6  4  5  4
  5  3  5  6  5  3  4  6  6  6  4  4  5  6  5  4  7  8  3  5  6  5  5  3
  4  6  9  6  8  8  6  4  5  6  6  5  7  5  6  7  4  6  3  3  4  4  4  5
  4  4  6  4  5  8  5  8  7  4  6  3  3  6  3  7  5  6  8  7  4  4  4  5
  6  3  3  3  7  6  4  3  3  6  5  2  5  6  3  3  7  4  5  5  5  7  3  7
  4  5  7  4  2  4  3  5  3  4  2  7  6  5  5  8  3  5  3  5  6  3  5  3
  5  4  2  4  5  1  6  6  8  8  7  6  8  5  2  5  4  7  4  7  2  4  7  7
  3  2  5  6  5  8  4  4  6  6  6  7  4  3  5  7  6  3  5  3  3  6  6  8
  5  5  4  7  2  3  5  4  5  6  6  4  9  5  9  7  2  7  5  4  5  6  3  4
  5  8  7  6  5  4  3  4  3  6  2  5  5  4  9  5  4  5  4  6  7  6  4  5
  6  4  5  6  4  3  5  5  6  4  4  3  3  3  2  7  5  6  5  5  5  5  4  0
  4  6  4  7  6  4  5  4  5  5  6  6  4  3  5  5  3  5  4  5  5  6  5  2
  8  7  4  5  3  6  7  5  7  6  2  6  7  3  5  6  2  5  7  6  3  6  4  7
  4  3  4  4  6  5  6  6  4  6  3  5  5  8  4  5  2  5  2  6  6  7  3  6
  4  5  3  4  3  5  6  7  5  7  6  5  7  4  4  5  6  5  5  2  6  7  7  7
  3  7  2  4  6  6  5  5  6  6  5  5  1  4  7  6  2  2  1  5  7  4  7  4
  2  3  5  3  3  8  4  4  6  4  8  2  3  4  6  5  5  5  3  6  6  5  7  5
  6  5  6  4  4  5  6  5  7  5  7  4  4  8  4  3  4  8  8  5  3  3  5  4
  4  5  5  7  2  9  6  5  4  4  3  6  5  6  7  8  5  8  5  6  6  8  6  7
  5  3  6  6  8  4  3  2  5  8  5  7  3  5  3  3  5  3  6  5  7  4  5  6
  4  7  5  7  4  6  4  5  6  5  4  4  8  6  4  5  7  3  5  6  8  9  3  8
  5  4  7  5  8  9  4  7  7  6  4  2  3  5  6  3  6  3  5  5  4  6  5  7
  6  5  7  7  3  6  5  4  5  6  5  1  4  5  6  6  4  4  3  4  3  7  6  5
```

```
 5  5  5  5  7  2  5  2  7  4  5  5  4  4  4  3  4  5  6  7  3  7  4  2
 7  4  7  4  8  5  2  6  4  4  5  3  6  9  3  6  7  1  4  5  8  4  5  8
 7  3  8  5  2  3  6  7  3  4  5  5  3  3  5  6  5  8  3  6  2  7  7  5
 7  6  8  4  4  2  3  6  5  6  4  5  8  5  5  5  6  2  4  7  7  5  7  3
 4  3  5  6  5  5  4  3  6  5  5  7  7  4  3  3  8  6  6  4  5  6  5  7
 2  6  6  3  3  3  5  7  2  3  4  5  6  5  5  5]
```

Lets see what this looks like.

[20]:
```
# visualisation styling code
sns.set(rc={'figure.figsize': (9.5, 6.5)})
sns.set_context('notebook')
sns.set_style("whitegrid")

# plotting
sns.distplot(example1, kde=True, hist=False, kde_kws={
             "shade": True}, color='steelblue', rug=True,)
plt.xlabel("Distribution", labelpad=15)
plt.ylabel("Density", labelpad=15)
plt.title("Binomial Distribution", fontsize=15, y=1.012);
```



3.5 Poisson distribution Notation:

$$f(k;\lambda) = \frac{\lambda^k e^{-\lambda}}{k!}$$

A Poisson distribution is a probability distribution which indicates how often an event is likely to occur within a specific time period. It is a discrete function "meaning that the event can only be measured in whole numbers". (Investopedia, 2019)

Real world usage: Common examples of Poisson processes are customers calling a help center, visitors to a website, radioactive decay in atoms, photons arriving at a space telescope, and movements in a stock price. Poisson processes are generally associated with time, but they do not have to be. In the stock case, we might know the average movements per day (events per time), but we could also have a Poisson process for the number of trees in an acre (events per area).

The syntax of random.default_rng().poisson is

np.random.default_rng().poisson(lam= , size=)

1. `np.random.default_rng().poisson` is the function name.
2. `lam` is the xpectation of interval, must be >= 0. A sequence of expectation intervals must be broadcastable over the requested size.
3. `size` is the Output shape.

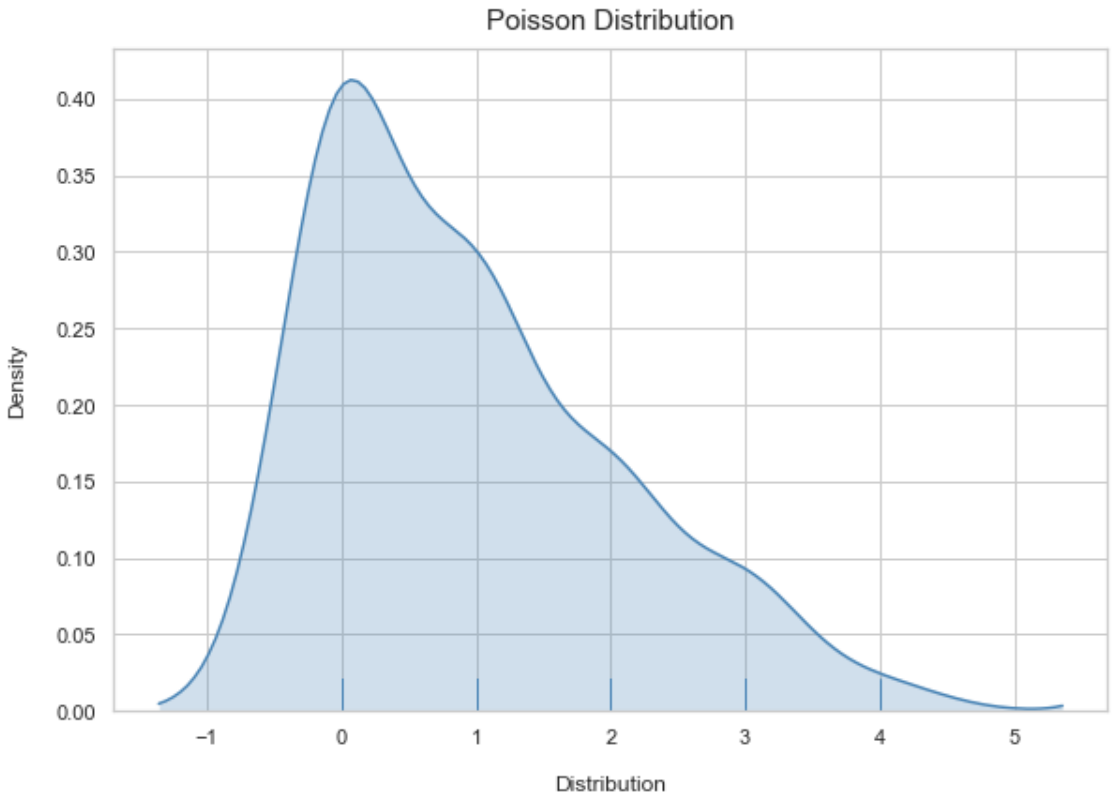
We can use random.default_rng().poisson to get a binomial distribution.

[21]:
```
# create the variables and assign them
example1 = np.random.default_rng().poisson(1, 100)

# print the array
print("Poisson distribution:\n", example1)
```

```
Poisson distribution:
 [0 1 0 3 1 1 0 0 0 0 0 1 0 3 2 1 0 0 3 4 2 0 0 1 0 0 0 1 0 1 2 0 0 1 3 0 1
 1 1 1 0 2 1 0 1 2 1 1 1 3 0 0 0 2 0 0 2 0 2 2 2 1 3 0 0 2 0 1 2 1 0 2 1 0
 0 3 0 0 1 2 2 0 0 1 1 1 0 0 0 4 0 0 1 3 2 3 0 1 0 1]
```

Finally lets have a look at the plot of the distribution.

[22]:
```
# visualisation styling code
sns.set(rc={'figure.figsize': (9.5, 6.5)})
sns.set_context('notebook')
sns.set_style("whitegrid")

# plotting
sns.distplot(example1, kde=True, hist=False, kde_kws={
             "shade": True}, color='steelblue', rug=True,)
plt.xlabel("Distribution", labelpad=15)
plt.ylabel("Density", labelpad=15)
plt.title("Poisson Distribution", fontsize=15, y=1.012);
```



1.1.6 4. Explain the use of seeds in generating pseudorandom numbers.

As stated previously in section one of this notebook pseudo-random numbers are generated by algorithms with deterministic behaviour. Integral to creating them is a pseudo-random number generator or (PRNG). It may also be referred to as a DRNG (digital random number generator) or a DRBG (deterministic random bit generator). They are programs or functions which use mathematics to simulate randomness. For data scientists this is useful, because they can take advantage of this to produce code that is both "random" and "reproducible". This is done by using "seeds". (Practical Data Science, 2020,(34)).

The (PRNG) uses a random "seed" to specify the "*start point when a computer generates a random number sequence. It can be any number, but it usually comes from seconds on a computer system's clock*". (Statistics How to, 2020,(32)). In `numpy.random` "seed" is simply a function that sets the random "seed". (Sharp Sight, 2019,(33)).

Note: A random seed (or seed state, or just seed) is a number (or vector) used to initialize a pseudorandom number generator. For a seed to be used in a pseudorandom number generator, it does not need to be random. Because of the nature of number generating algorithms, so long as the original seed is ignored, the rest of the values that the algorithm generates will follow probability distribution in a pseudorandom manner. A pseudorandom number generator's number sequence is completely determined by the seed: thus, if a pseudorandom number generator is reinitialized

with the same seed, it will produce the same sequence of numbers. The choice of a good random seed is crucial in the field of computer security. When a secret encryption key is pseudorandomly generated, having the seed will allow one to obtain the key. High entropy is important for selecting good random seed data. If the same random seed is deliberately shared, it becomes a secret key, so two or more systems using matching pseudorandom number algorithms and matching seeds can generate matching sequences of non-repeating numbers which can be used to synchronize remote systems, such as GPS satellites and receivers. Random seeds are often generated from the state of the computer system (such as the time), a cryptographically secure pseudorandom number generator or from a hardware random number generator. (Wikipedia,2020,(37))

Now lets look at explaing this. We won't use the seed element first, we will just generate 5 random numbers.

[23]:
```
rng = np.random.default_rng()
rng.integers(0, 100, 5)
```

[23]:
```
array([16, 53, 29, 68,  1], dtype=int64)
```

Lets try that again and see what we get.

[24]:
```
rng = np.random.default_rng()
rng.integers(0, 100, 5)
```

[24]:
```
array([ 6, 34, 65, 10, 62], dtype=int64)
```

As you can see we get different results because the seed hasn't been initiated.

Lets see what happens when we provide a "seed" value to the random number generator (RNG). In the examples below I entered 7 in the RNG.

[25]:
```
rng = np.random.default_rng(7)
rng.integers(0, 100, 5)
```

[25]:
```
array([94, 62, 68, 89, 57], dtype=int64)
```

[26]:
```
rng = np.random.default_rng(7)
rng.integers(0, 100, 5)
```

[26]:
```
array([94, 62, 68, 89, 57], dtype=int64)
```

The random generated numbers are the same. So if you start from the same seed, you get the very same sequence. This can be quite useful for debugging, simulations, training and testing.

1.1.7 5. References

[1] DataCamp Community. 2020. *(Tutorial) Random Number Generator Using Numpy.* [online] Available at: https://www.datacamp.com/community/tutorials/numpy-random [Accessed 3 November 2020].

[2] Bhattacharjya, D., 2020. *Numpy.Random.Seed(101) Explained*. [online] Medium. Available at: https://medium.com/@debanjana.bhattacharyya9818/numpy-random-seed-101-explained-2e96ee3fd90b [Accessed 4 November 2020].

[3] MLK - Machine Learning Knowledge. 2020. *Complete Numpy Random Tutorial - Rand, Randn, Randint, Normal, Uniform, Binomial And More | MLK - Machine Learning Knowledge*. [online] Available at: https://machinelearningknowledge.ai/numpy-random-rand-randn-randint-normal-uniform-binomial-poisson-sample-choice/ [Accessed 6 November 2020].

[4] Cui, Y., 2020. *A Cheat Sheet On Generating Random Numbers In Numpy*. [online] Medium. Available at: https://towardsdatascience.com/a-cheat-sheet-on-generating-random-numbers-in-numpy-5fe95ec2286 [Accessed 4 November 2020].

[5] Malik, U., 2020. *Numpy Tutorial: A Simple Example-Based Guide*. [online] Stack Abuse. Available at: https://stackabuse.com/numpy-tutorial-a-simple-example-based-guide/#therandommethod [Accessed 4 November 2020].

[6] Matthews, R., 2020. *Is Anything Truly Random?*. [online] BBC Science Focus Magazine. Available at: https://www.sciencefocus.com/science/is-anything-truly-random/ [Accessed 8 November 2020].

[7] Iditect.com. 2020. *Performance Difference Between Numpy.Random And Random.Random In Python*. [online] Available at: https://www.iditect.com/how-to/57220804.html [Accessed 8 November 2020].

[8] Spacey, J., 2016. *Pseudorandom Vs Random*. [online] Simplicable. Available at: https://simplicable.com/new/pseudorandom-vs-random [Accessed 6 November 2020].

[9] Tamilselvan, S., 2020. *Random Numbers In Numpy*. [online] Medium. Available at: https://medium.com/analytics-vidhya/random-numbers-in-numpy-29e929f16c70 [Accessed 4 November 2020].

[10] Technology, F., 2020. *Can A Computer Generate A Truly Random Number?*. [online] BBC Science Focus Magazine. Available at: https://www.sciencefocus.com/future-technology/can-a-computer-generate-a-truly-random-number/ [Accessed 8 November 2020].

[11] Thomas, A., 2020. *Good Practices With Numpy Random Number Generators*. [online] Albert Thomas. Available at: https://albertcthomas.github.io/good-practices-random-number-generators/ [Accessed 5 November 2020].

[12] Tutorial Links. 2020. *What Is Numpy Random Intro | Numpy Tutorial*. [online] Available at: https://tutorialslink.com/Articles/What-is-NumPy-Random-Intro-NumPy-Tutorial/1924 [Accessed 6 November 2020].

[13] McKinney, W., 2018. *Python For Data Analysis*. 2nd ed. O'Reilly.

[14] Computerhope.com. 2019. *What Is Pseudorandom?*. [online] Available at: https://www.computerhope.com/jargon/p/pseudo-random.htm [Accessed 8 November 2020].

[15] Ebner, J., 2019. *How To Create Random Samples With Python's Numpy.Random.Choice*. [online] Sharp Sight. Available at: https://www.sharpsightlabs.com/blog/numpy-random-choice/ [Accessed 7 November 2020].

[16] Sourcecodester. 2020. *Numpy Permutations | Free Source Code & Tutorials*. [online] Available at: https://www.sourcecodester.com/book/python/14297/numpy-permutations.html [Accessed 10 November 2020].

[17] Stack Overflow. 2020. *Shuffle Vs Permute Numpy*. [online] Available at: https://stackoverflow.com/questions/15474159/shuffle-vs-permute-numpy [Accessed 10 November 2020].

[18] Programmersought.com. 2020. *The Difference Between Shuffle And Permutation In Numpy.Random*. [online] Available at: https://www.programmersought.com/article/32541516928/ [Accessed 10 November 2020].

[19] Chen, J., 2020. *Uniform Distribution Definition*. [online] Investopedia. Available at: https://www.investopedia.com/terms/u/uniform-distribution.asp [Accessed 10 November 2020].

[20] Corporate Finance Institute. 2020. *Uniform Distribution - Overview, Examples, And Types*. [online] Available at: https://corporatefinanceinstitute.com/resources/knowledge/other/uniform-distribution/ [Accessed 10 November 2020].

[21] Taylor, C., 2019. *What Is A Uniform Distribution?*. [online] ThoughtCo. Available at: https://www.thoughtco.com/uniform-distribution-3126573 [Accessed 10 November 2020].

[22] Mathsisfun.com. 2020. *Normal Distribution*. [online] Available at: https://www.mathsisfun.com/data/standard-normal-distribution.html [Accessed 11 November 2020].

[23] Ebner, J., 2020. *How To Use Numpy Random Normal In Python*. [online] Sharp Sight. Available at: https://www.sharpsightlabs.com/blog/numpy-random-normal/ [Accessed 11 November 2020].

[24] Learningaboutelectronics.com. 2018. *How To Create A Normal Distribution Plot In Python With The Numpy And Matplotlib Modules*. [online] Available at: http://www.learningaboutelectronics.com/Articles/How-to-create-a-normal-distribution-plot-in-Python-with-numpy-and-matplotlib.php [Accessed 11 November 2020].

[25] En.wikipedia.org. 2020. *68–95–99.7 Rule*. [online] Available at: https://en.wikipedia.org/wiki/68%E2%80%9395%E2%80%9399.7_rule [Accessed 11 November 2020].

[26] Sciencedirect.com. 2017. *Exponential Distribution - An Overview* | Sciencedirect Topics. [online] Available at: https://www.sciencedirect.com/topics/mathematics/exponential-distribution [Accessed 11 November 2020].

[27] GeeksforGeeks. 2020. *Python - Binomial Distribution* - Geeksforgeeks. [online] Available at: https://www.geeksforgeeks.org/python-binomial-distribution/ [Accessed 12 November 2020].

[28] Yiu, T., 2019. *Fun With The Binomial Distribution*. [online] Medium. Available at: https://towardsdatascience.com/fun-with-the-binomial-distribution-96a5ecabf65b [Accessed 12 November 2020].

[29] Statistics How To. 2020. *Binomial Distribution: Formula, What It Is, And How To Use It In Simple Steps*. [online] Available at: https://www.statisticshowto.com/probability-and-statistics/binomial-theorem/binomial-distribution-formula/ [Accessed 12 November 2020].

[30] Sharma, V., 2020. *Probability For Data Science With Numpy*. [online] Medium. Available at: https://levelup.gitconnected.com/probability-for-data-science-with-numpy-7e76e5e65910 [Accessed 12 November 2020].

[31] Python and R Tips. 2018. *Simulating Coin Toss Experiment In Python With Numpy - Python And R Tips*. [online] Available at: https://cmdlinetips.com/2018/12/simulating-coin-toss-experiment-with-binomial-random-numbers-using-numpy/ [Accessed 12 November 2020].

[32] Statistics How To. 2020. *Random Seed: Definition - Statistics How To*. [online] Available at: https://www.statisticshowto.com/random-seed-definition/ [Accessed 15 November 2020].

[33] Ebner, J., 2019. *Numpy Random Seed Explained*. [online] Sharp Sight. Available at: https://www.sharpsightlabs.com/blog/numpy-random-seed/ [Accessed 15 November 2020].

[34] Practicaldatascience.org. 2020. *Numbers In Computers — Practical Data Science*. [online] Available at: https://www.practicaldatascience.org/html/ints_and_floats.html [Accessed 15 November 2020].

[35] McBride, M., 2019. *Creating Random Data In Numpy*. [online] Pythoninformer.com. Available at: https://www.pythoninformer.com/python-libraries/numpy/creating-random-data/ [Accessed 18 November 2020].

[36] Kaggle.com. 2020. *Different Types Of Distributions And Visualizations With Real Life Examples*| Data Science And Machine Learning. [online] Available at: https://www.kaggle.com/questions-and-answers/175147 [Accessed 19 November 2020].

[37] En.wikipedia.org. 2020. *Random Seed*. [online] Available at: https://en.wikipedia.org/wiki/Random_seed [Accessed 21 November 2020].