Contents

[NUMPY 2](#_Toc187141606)

[WHAT IS NUMPY 2](#_Toc187141607)

[NUMPY ARRAYS 2](#_Toc187141608)

[NUMPY FUNCTIONS 2](#_Toc187141609)

[arange 2](#_Toc187141610)

[EXAMPLE 3](#_Toc187141611)

[np.ones() and np.zeros() 3](#_Toc187141612)

[EXAMPLE 3](#_Toc187141613)

[np.linspace() 3](#_Toc187141614)

[EXAMPLE 4](#_Toc187141615)

# NUMPY

## PROBABILITY THEORY AND STATISTICS

### MODE

* The mode represents the value or values that occur most frequently in a dataset. It is a measure of central tendency that highlights the most commonly occurring value(s) in a dataset.
* Unlike the mean and median, which focus on the average or middle values, the mode provides insight into the most prevalent or dominant value(s) in the dataset.

|  |
| --- |
| Let's consider the following dataset: [2, 4, 4, 6, 8, 8, 8, 10]. **MODE**   * The mode is the value that appears most frequently in the dataset. * In this case, the value 8 appears three times, which is more frequently than any other value in the dataset. **Therefore, the mode of this dataset is 8**. |

* The mode can be useful in various scenarios. For example:
  + In a survey or poll, the mode can help identify the most common response or opinion among the respondents.
  + In analyzing sales data, the mode can indicate the most popular product or item.
  + In studying a distribution of data, the mode can provide information about the peak or highest point of frequency.

It is important to note that

* a dataset can have multiple modes if more than one value occurs with the same highest frequency. In such cases, the dataset is referred to as multimodal.
* If no value repeats and each value occurs with the same frequency, the dataset is said to have no mode.

### VARIANCE

* **Variance represents the spread or variability of a set of data points from the mean (average) value. It quantifies how much the individual values in a dataset deviate from the mean**.
* A higher variance indicates that the data points are more spread out from the mean, while a lower variance suggests that the data points are closer to the mean.
* Variance is calculated by taking the average of the squared differences between each data point and the mean. The squared differences are used to ensure that positive and negative deviations from the mean do not cancel each other out.

|  |
| --- |
| **VARIANCE**: Let's consider the following dataset: [2, 4, 6, 8, 10].  To find the variance, follow these steps:  1. Step 1: Calculate the mean of the dataset: Mean = 6.  2. Step 2: Subtract the mean from each value, square the result, and sum up all the squared values.  (**2 - 6)^2 + (4 - 6)^2 + (6 - 6)^2 + (8 - 6)^2 + (10 - 6)^2 = 16 + 4 + 0 + 4 + 16 = 40.**  3. Step 3: Divide the sum by the number of values in the dataset.  **Variance = 40 / 5 = 8.**  So, the variance of this dataset is 8. |

* Variance is commonly used in statistics and probability theory to analyze and compare the dispersion or variability of datasets. It allows us to understand the distribution of data points and the extent to which they deviate from the average.
* In practical terms, variance helps to:
  + Assess the volatility or variability of financial returns.
  + Evaluate the performance of models or predictions by comparing the variance of observed data and predicted values.
  + Compare the spread of data in different datasets or groups.
  + Understand the variability in experimental or observational data.
* Variance is an important statistical concept that provides valuable insights into the variability and spread of data, enabling us to make informed decisions and draw meaningful conclusions.

### CALCULATION OF MEAN, MEDIAN AND MODE

Let's consider the following dataset: [2, 4, 6, 8, 10].

|  |
| --- |
| **MEAN:**  To find the mean (average), we sum up all the values in the dataset and divide it by the number of values.  Mean = (2 + 4 + 6 + 8 + 10) / 5 = 30 / 5 = 6.  So, the mean of this dataset is 6. |
| **MEDIAN**:  To find the median, we arrange the values in ascending order and locate the middle value. If there is an even number of values, we take the average of the two middle values.  Sorted dataset: [2, 4, 6, 8, 10].  Since there is an odd number of values, the middle value is the median.  So, the median of this dataset is 6. |
| Let's consider the following dataset: [2, 4, 4, 6, 8, 8, 8, 10]. **MODE**   * The mode is the value that appears most frequently in the dataset. * In this case, the value 8 appears three times, which is more frequently than any other value in the dataset. **Therefore, the mode of this dataset is 8**. |
| **VARIANCE**: Let's consider the following dataset: [2, 4, 6, 8, 10].  To find the variance, follow these steps:  1. Step 1: Calculate the mean of the dataset: Mean = 6.  2. Step 2: Subtract the mean from each value, square the result, and sum up all the squared values.  (**2 - 6)^2 + (4 - 6)^2 + (6 - 6)^2 + (8 - 6)^2 + (10 - 6)^2 = 16 + 4 + 0 + 4 + 16 = 40.**  3. Step 3: Divide the sum by the number of values in the dataset.  **Variance = 40 / 5 = 8.**  So, the variance of this dataset is 8. |

## WHAT IS NUMPY

NumPy is a Python library that stands for "Numerical Python". It is a fundamental package for scientific computing in Python, providing support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays efficiently.

## NUMPY ARRAYS

* A NumPy array, also known as ndarray, is a central data structure in NumPy library. It is a multi-dimensional grid of elements of the same type. NumPy arrays can have any number of dimensions, but most commonly they are 1D (one-dimensional), 2D (two-dimensional), or 3D (three-dimensional).
* NumPy arrays are widely used in scientific computing, data analysis, machine learning, and numerical computations due to their efficiency and versatility.

|  |  |
| --- | --- |
| **import numpy as np**    **# Creating a 1D array**  **arr1 = np.array([1, 2, 3, 4, 5])**  **print(arr1) # Output: [1 2 3 4 5]**    **# Creating a 2D array**  **arr2 = np.array([[1, 2, 3], [4, 5, 6]])**  **print(arr2)**  **# Output:**  **# [[1 2 3]**  **# [4 5 6]]**    **# Creating a 3D array**  **arr3 = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])**  **print(arr3)**  **# Output:**  **# [[[1 2]**  **# [3 4]]**    **# [[5 6]**  **# [7 8]]]** | Key features and characteristics of NumPy arrays:   1. **Homogeneous Data**: NumPy arrays contain elements of the **same data type** 2. **Fixed Size**: Once a NumPy array is created, its size is fixed and cannot be changed. To modify the size, a new array needs to be created. 3. **Fast and Efficient**: NumPy arrays are implemented in C, making them faster and more efficient compared to Python lists. They allow for vectorized operations, which can perform computations on entire arrays without the need for loops. 4. **Powerful Indexing and Slicing**: NumPy arrays support advanced indexing and slicing operations, allowing easy access to elements or sub-arrays based on specific conditions or criteria. 5. **Mathematical Operations**: NumPy arrays provide a wide range of mathematical functions and operations to perform calculations efficiently on arrays**, such as element-wise operations, linear algebra operations, statistical functions.** |

## NUMPY FUNCTIONS

## arange

* The `**np.arange()`** function in NumPy is used to create an array with regularly spaced values within a specified range.
* **It is similar to the Python built-in `range()` function but returns an array instead of a list**.
* **SYNTAX**

**np.arange(start, stop, step, dtype=None)**

|  |  |
| --- | --- |
| `start` (optional): | The starting value of the sequence. If not provided, the default value is 0. |
| stop | The end value of the sequence. It is exclusive, so the generated sequence will stop before reaching this value. |
| step | The step size between consecutive values in the sequence. If not provided, the default value is 1 |
| dtype | The data type of the elements in the resulting array. If not specified, NumPy will determine it based on the other input arguments |

### EXAMPLE

|  |
| --- |
| **import numpy as np**    **# Example 1: Generate a sequence of numbers from 0 to 9**  **arr1 = np.arange(10)**  **print(arr1) # Output: [0 1 2 3 4 5 6 7 8 9]** |
| **# Example 2: Generate a sequence of even numbers from 2 to 10**  **arr2 = np.arange(2, 11, 2)**  **print(arr2) # Output: [2 4 6 8 10]** |
| **# Example 3: Generate a sequence of floating-point numbers from 0 to 1 with a step of 0.1**  **arr3 = np.arange(0, 1, 0.1)**  **print(arr3) # Output: [0. 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9]** |
| **print(np.arange(0, 10, 2, dtype=float)) #OUTPUT [0. 2. 4. 6. 8.]** |

## np.ones() and np.zeros()

* In NumPy, the functions `np.ones()` and `np.zeros()` are used to create arrays filled with ones and zeros, respectively. These functions allow we to easily create arrays of desired shapes and sizes filled with the specified values.
* **SYNTAX** - This function creates an array of ones with the specified shape

**np.ones(shape, dtype=None, order='C')**

|  |  |
| --- | --- |
| shape | The shape of the array, specified as a tuple of integers. For example, `(3, 4)` creates a 2D array with 3 rows and 4 columns |
| dtype(optional): | The data type of the elements in the array. If not specified, the default data type is `float64`. |
| order(optional): | The order in which the array is stored in memory. It can be `'C'` for row-major (C-style) or `'F'` for column-major (Fortran-style). The default is `'C' |

* **SYNTAX** - This function creates an array of zeros with the specified shape

**np.zeros(shape, dtype=None, order='C')**

### EXAMPLE

|  |  |
| --- | --- |
| **# Example 1: Create a 1D array of ones with 5 elements**  **arr1 = np.ones(5)**  **print(arr1)**  **# Output: [1. 1. 1. 1. 1.] 🡪 “.”(dot) represents the float** | **# Example 3: Create a 3D array of ones with dimensions 2x3x2**  **arr3 = np.ones((2, 3, 2))**  **print(arr3)**  **# Output:**  **# [[[1. 1.]**  **# [1. 1.]**  **# [1. 1.]]**    **# [[1. 1.]**  **# [1. 1.]**  **# [1. 1.]]]** |
| **# Example 2: Create a 2D array of zeros with 3 rows and 4 columns**  **arr2 = np.zeros((3, 4)) 🡨 THE ROWS AND COLUMNS NEED TO BE PASSED AS TUPLE**  **print(arr2)**  **# Output:**  **# [[0. 0. 0. 0.]**  **# [0. 0. 0. 0.]**  **# [0. 0. 0. 0.]]** |

## np.linspace()

* the `np.linspace()` function is used to create an array with evenly spaced values over a specified interval. It is particularly useful when we want to generate a sequence of numbers with a specific number of elements.
* `np.linspace()` is commonly used in various scientific and numerical computing applications, such as plotting graphs, creating test datasets, and generating time series data.

**np.linspace(start, stop, num=50, endpoint=True, retstep=False, dtype=None)**

|  |  |
| --- | --- |
| **start** | The starting value of the sequence |
| **stop** | The end value of the sequence |
| **num(optional):** | The number of equally spaced values to generate between `start` and `stop`. The default value is 50 i.e How many numbers we want between start and stop (including stop) |
| **endpoint(optional)** | Whether or not to include the `stop` value in the sequence. **If `True`, the sequence will include `stop`. The default value is `True** |
| **retstep(optional)** | Whether or not to return the spacing between consecutive values. If `True`, the function will return a tuple containing the array and the step value. The default value is `False`. |
| **dtype(optional)** | The data type of the elements in the resulting array. If not specified, NumPy will determine it based on the other input arguments. |

### EXAMPLE

|  |
| --- |
| **# Example 1: Generate a sequence of 10 numbers from 0 to 1 (including 1)**  **arr1 = np.linspace(0, 1, num=10)**  **print(arr1)**  **# Output: [0. 0.11111111 0.22222222 0.33333333 0.44444444 0.55555556**  **# 0.66666667 0.77777778 0.88888889 1. ]** |
| **# Example 2: Generate a sequence of 5 numbers from -2 to 2 (including -2 and 2)**  **arr2 = np.linspace(-2, 2, num=5)**  **print(arr2)**  **# Output: [-2. -1. 0. 1. 2.]** |
| **# Example 3: Generate a sequence of 6 numbers from 1 to 10 (excluding 10) and return the step value**  **arr3, step = np.linspace(1, 10, num=6, retstep=True)**  **print(arr3) # [ 1. 3.8 6.6 9.4 12.2 15. ]**  **print(step) 🡺 # 3.2 🡸 THIS IS THE SPACING BETWEEN THE ELEMENTS** |

## NUMPY RANDOM LIBRARY

### UNIFORM DISTRIBUTION

* In probability theory and statistics, a uniform distribution refers to a probability distribution where all outcomes within a specified range are equally likely. In other words, each value within the range has the same probability of occurring. The uniform distribution is often represented graphically as a rectangular shape, where the probability density function (PDF) is constant over the range.
* For example, consider rolling a fair six-sided die. Each outcome (1, 2, 3, 4, 5, or 6) has an equal probability of 1/6, making it a uniform distribution.

STANDARD NORMAL DISTRIBUTION:

The standard normal distribution, also known as the Gaussian distribution or the bell curve, is a specific type of probability distribution. It is characterized by its symmetric bell-shaped curve and has a mean of 0 and a standard deviation of 1.

The standard normal distribution is often denoted as N(0, 1), where N represents the normal distribution, 0 represents the mean, and 1 represents the standard deviation. The PDF of the standard normal distribution is given by the mathematical formula:

f(x) = (1 / √(2π)) \* e^(-x^2 / 2)

where e is the base of the natural logarithm and π is a mathematical constant.

The standard normal distribution is widely used in statistics and probability theory. It is particularly useful because many natural phenomena and statistical processes tend to follow a normal distribution. Moreover, the standard normal distribution serves as a foundation for various statistical tests and techniques, such as hypothesis testing and confidence intervals.

NumPy's `random` module is a part of the NumPy library that provides functions for generating random numbers and arrays. It is widely used in scientific computing, simulations, and data analysis tasks. The `random` module offers various functions to generate random values from different probability distributions.

Here are some commonly used functions from the `random` module:

1. `random.rand()`: Generates random numbers from a uniform distribution over the range [0, 1). It takes in an optional shape parameter to create an array of random values.

```python

import numpy as np

# Generate a random float between 0 and 1

rand\_num = np.random.rand()

print(rand\_num)

# Generate a 2x3 array of random floats between 0 and 1

rand\_array = np.random.rand(2, 3)

print(rand\_array)

```

2. `random.randint()`: Generates random integers from a specified range. It takes in the lower bound, upper bound (exclusive), and optional shape parameters.

```python

import numpy as np

# Generate a random integer between 0 and 9

rand\_int = np.random.randint(10)

print(rand\_int)

# Generate a 1D array of 5 random integers between 0 and 9

rand\_array = np.random.randint(0, 10, 5)

print(rand\_array)

# Generate a 2x3 array of random integers between 1 and 100

rand\_matrix = np.random.randint(1, 101, (2, 3))

print(rand\_matrix)

```

3. `random.randn()`: Generates random samples from a standard normal distribution (mean=0, standard deviation=1). It takes in an optional shape parameter.

```python

import numpy as np

# Generate a random number from a standard normal distribution

rand\_num = np.random.randn()

print(rand\_num)

# Generate a 1D array of 5 random numbers from a standard normal distribution

rand\_array = np.random.randn(5)

print(rand\_array)

# Generate a 2x3 array of random numbers from a standard normal distribution

rand\_matrix = np.random.randn(2, 3)

print(rand\_matrix)

```

4. `random.choice()`: Generates random samples from an input array or list. It takes in the input array/list and an optional size parameter to specify the number of random samples to generate.

```python

import numpy as np

# Generate a random element from the given array

rand\_choice = np.random.choice([1, 2, 3, 4, 5])

print(rand\_choice)

# Generate a 1D array of 3 random elements from the given array

rand\_array = np.random.choice([1, 2, 3, 4, 5], size=3)

print(rand\_array)

# Generate a 2x2 array of random elements from the given array

rand\_matrix = np.random.choice([1, 2, 3, 4, 5], size=(2, 2))

print(rand\_matrix)

```

These are just a few examples of the functions available in the `random` module of NumPy. There are many more functions and options available to generate random numbers and arrays from different probability distributions, such as normal, exponential, and more. NumPy's `random` module is a powerful tool for generating random values and arrays for various applications.