### WEEK-1:

#### Write a python program to compute Central Tendency Measures: Mean, Median, Mode Measure of Dispersion: Variance, Standard Deviation

* 1. **Mean**

# Python program to print # mean of elements

# list of elements to calculate mean n\_num = [1, 2, 3, 4, 5]

n = len(n\_num)

get\_sum = sum(n\_num) mean = get\_sum / n

print("Mean / Average is: " + str(mean))

**OUTPUT**:

# Mean / Average is: 3.0

#### Median

# Python program to print # median of elements

# list of elements to calculate median n\_num = [1, 2, 3, 4, 5]

n = len(n\_num) n\_num.sort()

if n % 2 == 0:

median1 = n\_num[n//2] median2 = n\_num[n//2 - 1]

median = (median1 + median2)/2

else:

median = n\_num[n//2]

print("Median is: " + str(median))

**OUTPUT**:

# Median is: 3

#### Mode

# Python program to print # mode of elements

from collections import Counter

# list of elements to calculate mode

n\_num = [1, 2, 3, 4, 5, 5]

n = len(n\_num)

data = Counter(n\_num) get\_mode = dict(data)

mode = [k for k, v in get\_mode.items() if v == max(list(data.values()))]

if len(mode) == n:

get\_mode = "No mode found"

else:

get\_mode = "Mode is / are: " + ', '.join(map(str, mode))

print(get\_mode)

**OUTPUT**:

# Mode is / are: 5

#### Variance:

# Python program to get variance of a list # Importing the NumPy module

import numpy as np

# Taking a list of elements list = [2, 4, 4, 4, 5, 5, 7, 9]

# Calculating variance using var() print(np.var(list))

**OUTPUT**:

# 4.0

#### Standard Deviation

# Python program to get

# standard deviation of a list

# Importing the NumPy module import numpy as np

# Taking a list of elements list = [2, 4, 4, 4, 5, 5, 7, 9]

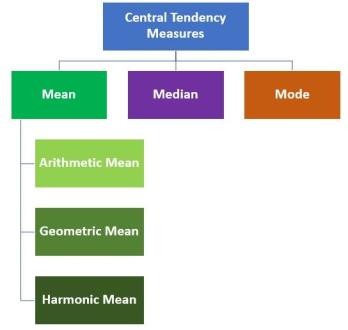
# Calculating standard # deviation using var() print(np.std(list))

**OUTPUT**:

# 2.0

**Sample Viva Questions** :

* 1. **What are Central Tendency Measures: Mean, Median, Mode**



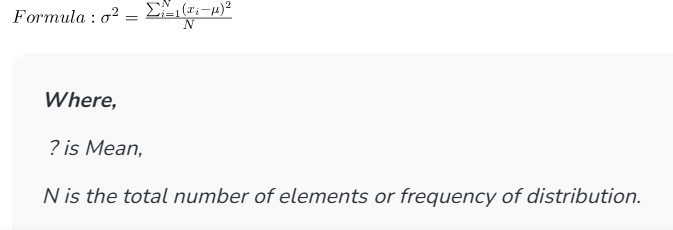
* 1. **Define Mean, Mode and Median?**

The mean is the average of all numbers and is sometimes called the arithmetic mean. The median is the middle number in a group of numbers

The mode is the number that occurs most often within a set of numbers.

#### Define Variance, Standard Deviation?

Variance is the sum of squares of differences between all numbers and means. The mathematical formula for variance is as follows,



Standard Deviation is the square root of variance. It is a measure of the extent to which data varies from the mean. The mathematical formula for calculating standard deviation is as follows,



#### What is the syntax for input and output in python?

In Python, we use the print() function to output data to the screen. Sometimes we might want to take input from the user. We can do so by using the input() function. Python takes all the input as a string input by default.

#### Which python version your using?

To check your Python version, run python ‐‐version in your command line (Windows), shell (Mac), or terminal (Linux/Ubuntu).

### WEEK-2:

1. **Study of Python Basic Libraries such as Statistics, Math, Numpy and Scipy**

# Statistics

Statistics is use to calculate mathematical statistics of numeric data. it is an area of applied mathematics concerned with data collection analysis, interpretation, and presentation. Python's statistics is a built-in Python library for descriptive statistics. You can use it if your datasets are not too large or if you can't rely on importing other libraries.

Descriptive statistics generally means describing the data with the help of some representative methods like charts, tables, Excel files, etc. The data is described in such a way that it can express some meaningful information that can also be used to find some future trends. Describing and summarizing a single variable is called **univariate analysis.**Describing a statistical relationship between two variables is called **bivariate analysis.**Describing the statistical relationship between multiple variables is called **multivariate analysis.**

**Statistics Methods**

|  |  |
| --- | --- |
| **Method** | **Description** |
| [statistics.harmonic\_mean()](https://www.w3schools.com/python/ref_stat_harmonic_mean.asp) | Calculates the harmonic mean (central location) of the given data |
| [statistics.mean()](https://www.w3schools.com/python/ref_stat_mean.asp) | Calculates the mean (average) of the given data |
| [statistics.median()](https://www.w3schools.com/python/ref_stat_median.asp) | Calculates the median (middle value) of the given data |
| [statistics.median\_grouped()](https://www.w3schools.com/python/ref_stat_median_grouped.asp) | Calculates the median of grouped continuous data |
| [statistics.median\_high()](https://www.w3schools.com/python/ref_stat_median_high.asp) | Calculates the high median of the given data |
| [statistics.median\_low()](https://www.w3schools.com/python/ref_stat_median_low.asp) | Calculates the low median of the given data |
| [statistics.mode()](https://www.w3schools.com/python/ref_stat_mode.asp) | Calculates the mode (central tendency) of the given numeric or nominal data |
| [statistics.pstdev()](https://www.w3schools.com/python/ref_stat_pstdev.asp) | Calculates the standard deviation from an entire population |
| [statistics.stdev()](https://www.w3schools.com/python/ref_stat_stdev.asp) | Calculates the standard deviation from a sample of data |
| [statistics.pvariance()](https://www.w3schools.com/python/ref_stat_pvariance.asp) | Calculates the variance of an entire population |
| [statistics.variance()](https://www.w3schools.com/python/ref_stat_variance.asp) | Calculates the variance from a sample of data |

# Math

To carry out calculations with real numbers, the Python language contains many additional functions collected in a library (module) called math.

To use these functions at the beginning of the program, you need to connect the math library, which is done by the command

#### import math

Python provides various operators for performing basic calculations, such as \* for multiplication,% for a module, and / for the division. If you are developing a program in Python to perform certain tasks, you need to work with trigonometric functions, as well as complex numbers. Although you cannot use these functions directly, you can access them by turning on the math module [math](https://docs.python.org/3/library/math.html), which gives access to hyperbolic, trigonometric and logarithmic functions for real numbers. To use complex numbers, you can use the math module [cmath](https://docs.python.org/2/library/cmath.html). When comparing *math* vs *numpy*, a *math* library is more lightweight and can be used for extensive computation as well.

The Python Math Library is the foundation for the rest of the math libraries that are written on top of its functionality and functions defined by the C standard. Please refer to the [python math examples](https://docs.python.org/3/library/math.html) for more information.

#### Number-theoretic and representation functions

This part of the mathematical library is designed to work with numbers and their representations. It allows you to effectively carry out the necessary transformations with support for NaN (not a number) and infinity and is one of the most important sections of the Python math library. Below is a short list of features for Python 3rd version. A more detailed description can be found in the documentation for the math library.

**math.ceil(x)** - return the ceiling of x, the smallest integer greater than or equal to x

**math.comb(n, k)** - return the number of ways to choose k items from n items without repetition and without order

**math.copysign(x, y)** - return float with the magnitude (absolute value) of x but the sign of

y. On platforms that support signed zeros, copysign (1.0, -0.0) returns -1.0

**math.fabs(x)** - return the absolute value of x

**math.factorial(x)** - return x factorial as an integer. Raises ValueError if x is not integral or is negative

**math.floor(x)** - return the floor of x, the largest integer less than or equal to x

**math.fmod(x, y)** - return fmod(x, y), as defined by the platform C library

**math.frexp(x)** - return the mantissa and exponent of x as the pair (m, e). m is a float and e is an integer such that x == m \* 2\*\*e exactly

**math.fsum(iterable)** - return an accurate floating-point sum of values in the iterable

**math.gcd(a, b)** - return the greatest common divisor of the integers a and b

**math.isclose(a, b, \*, rel\_tol=1e-09, abs\_tol=0.0)** - return True if the values a and b are close to each other and False otherwise

**math.isfinite(x)** - return True if x is neither infinity nor a NaN, and False otherwise (note that 0.0 is considered finite)

**math.isinf(x)** - return True if x is positive or negative infinity, and False otherwise

**math.isnan(x)** - return True if x is a NaN (not a number), and False otherwise

**math.isqrt(n)** - return the integer square root of the nonnegative integer n. This is the floor of the exact square root of n, or equivalently the greatest integer a such that a² ≤ n

**math.ldexp(x, i) - return x \* (2\*\*i).** This is essentially the inverse of function frexp()

**math.modf(x)** - return the fractional and integer parts of x. Both results carry the sign of x and are floats

**math.perm(n, k=None)** - return the number of ways to choose k items from n items without repetition and with order

**math.prod(iterable, \*, start=1**) - calculate the product of all the elements in the input iterable. The default start value for the product is 1

**math.remainder(x, y)** - return the IEEE 754-style remainder of x with respect to y **math.trunc(x)** - return the Real value x truncated to an Integral (usually an integer) **Power and logarithmic functions**

The power and logarithmic functions section are responsible for exponential calculations, which is important in many areas of mathematics, engineering, and statistics. These

functions can work with both natural logarithmic and exponential functions, logarithms modulo two, and arbitrary bases.

**math.exp(x)** - return e raised to the power x, where e = 2.718281… is the base of natural logarithms

**math.expm1(x)** - return e raised to the power x, minus 1. Here e is the base of natural logarithms. math.log(x[, base]) - With one argument, return the natural logarithm of x (to base e). With two arguments, return the logarithm of x to the given base, calculated as log(x)/log(base)

**math.log1p(x)** - return the natural logarithm of 1+x (base e). The result is calculated in a way that is accurate for x near zero

**math.log2(x)** - return the base-2 logarithm of x. This is usually more accurate than log(x, 2)

**math.log10(x)** - return the base-10 logarithm of x. This is usually more accurate than log(x, 10)

**math.pow(x, y)** - return x raised to the power y **math.sqrt(x)** - return the square root of x **Trigonometric functions**

Trigonometric functions, direct and inverse, are widely represented in the Python Mathematical Library. They work with radian values, which is important. It is also possible to carry out calculations with Euclidean functions.

**math.acos(x)** - return the arc cosine of x, in radians **math.asin(x)** - return the arc sine of x, in radians **math.atan(x)** - return the arctangent of x, in radians

**math.atan2(y, x) - return atan(y / x),** in radians. The result is between -pi and pi

**math.cos(x)** - return the cosine of x radians

**math.dist(p, q)** - return the Euclidean distance between two points p and q, each given as a sequence (or iterable) of coordinates. The two points must have the same dimension

**math.hypot(\*coordinates)** - return the Euclidean norm, sqrt(sum(x\*\*2 for x in coordinates)). This is the length of the vector from the origin to the point given by the coordinates

**math.sin(x)** - return the sine of x radians

**math.tan(x)** - return the tangent of x radians

#### Angular conversion

Converting degrees to radians and vice versa is a fairly common function and therefore the developers have taken these actions to the Python library. This allows you to write compact and understandable code.

**math.degrees(x)** - convert angle x from radians to degrees **math.radians(x)** - convert angle x from degrees to radians **Hyperbolic functions**

Hyperbolic functions are analogs of trigonometric functions that are based on hyperbolas instead of circles.

**math.acosh(x)** - return the inverse hyperbolic cosine of x **math.asinh(x)** - return the inverse hyperbolic sine of x **math.atanh(x)** - return the inverse hyperbolic tangent of x **math.cosh(x)** - return the hyperbolic cosine of x **math.sinh(x)** - return the hyperbolic sine of x **math.tanh(x)** - return the hyperbolic tangent of x

#### Special functions

The special functions section is responsible for error handling and gamma functions. This is a necessary function and it was decided to implement it in the standard Python mathematical library.

**math.erf(x)** - Return the error function at x

**math.erfc(x)** - Return the complementary error function at x

**math.gamma(x)** - Return the Gamma function at x

**math.lgamma(x)** - Return the natural logarithm of the absolute value of the Gamma function at x

#### Constants

The constant section provides ready-made values for basic constants and writes them with the necessary accuracy for a given hardware platform, which is important for Python's portability as a cross-platform language. Also, the very important values infinity and “not a number” are defined in this section of the Python library.

**math.pi** - the mathematical constant π = 3.141592…, to available precision

**math.e** - the mathematical constant e = 2.718281…, to available precision

**math.tau** - the mathematical constant τ = 6.283185…, to available precision. Tau is a circle constant equal to 2π, the ratio of a circle’s circumference to its radius

**math.inf** - a floating-point positive infinity. (For negative infinity, use -math.inf.) Equivalent to the output of float('inf')

**math.nan** - a floating-point “not a number” (NaN) value. Equivalent to the output of float('nan')

# Scipy

SciPy is a library for the open-source Python programming language, designed to perform scientific and engineering calculations.

The capabilities of this library are quite wide:

* + - Search for minima and maxima of functions
    - Calculation of function integrals
    - Support for special functions
    - Signal processing
    - Image processing
    - Work with genetic algorithms
    - Solving ordinary differential equations

SciPy in Python is a collection of mathematical algorithms and functions built as a Numpy extension. It greatly extends the capabilities of an interactive Python session by providing the user with high-level commands and classes for managing and visualizing data. With SciPy, an interactive Python session becomes a data processing and prototyping system competing with systems such as MATLAB, IDL, Octave, R-Lab, and SciLab.

An additional advantage of Python-based SciPy is that it is also a fairly powerful programming language used in the development of complex programs and specialized applications. Scientific applications also benefit from the development of additional modules in numerous software niches by developers around the world. Everything from parallel programming for the web to routines and database classes is available to the Python programmer. All of these features are available in addition to the SciPy math library.

#### Packages for mathematical methods

SciPy is organized into sub-packages covering various scientific computing areas:

**cluster** - Clustering Algorithms

**constants** - physical and mathematical constants

**fftpack** - Fast Fourier Transform subroutines

**integrate** - integration and solution of ordinary differential equations

**Interpolate** - interpolation and smoothing splines

**io** - input and output

**linalg** - linear algebra

**ndimage** - n-dimensional image processing

**odr** -orthogonal regression distance multiplexing **optimize** - root structure optimization and search **signal** - signal processing

**sparse** - sparse matrices and related procedures **spatial** - spatial Data Structures and Algorithms **special** - special functions

**stats** - statistical Distributions and Functions

**weave** - C / C ++ integration

The SciPy ecosystem includes general and specialized tools for data management and computation, productive experimentation, and high-performance computing. Below, we overview some key packages, though there are many more relevant packages.

Main components of ScyPy

#### Data and computation:

**pandas,** providing high-performance, easy-to-use data structures

**SymPy,** for symbolic mathematics and computer algebra

**scikit-image** is a collection of algorithms for image processing

**scikit-learn** is a collection of algorithms and tools for machine learning **h5py and PyTables** can both access data stored in the HDF5 format **Productivity and high-performance computing:**

IPython, a rich interactive interface, letting you quickly process data and test ideas

The Jupyter notebook provides IPython functionality and more in your web browser, allowing you to document your computation in an easily reproducible form

Cython extends Python syntax so that you can conveniently build C extensions, either to speed up critical code or to integrate with C/C++ libraries

Dask, Joblib or IPyParallel for distributed processing with a focus on numeric data

#### Quality assurance:

nose, a framework for testing Python code, being phased out in preference for pytest numpydoc, a standard, and library for documenting Scientific Python libraries SciPy provides a very wide and sought-after feature set:

Clustering package **(scipy.cluster)**

Constants **(scipy.constants)**

Discrete Fourier transforms **(scipy.fftpack)** Integration and ODEs **(scipy.integrate)** Interpolation **(scipy.interpolate)**

Input and output **(scipy.io)**

Linear algebra **(scipy.linalg)**

Miscellaneous routines **(scipy.misc)**

Multi-dimensional image processing **(scipy.ndimage)** Orthogonal distance regression **(scipy.odr)** Optimization and Root Finding **(scipy.optimize)** Signal processing **(scipy.signal)**

Sparse matrices **(scipy.sparse)**

Sparse linear algebra **(scipy.sparse.linalg)**

Compressed Sparse Graph Routines **(scipy.sparse.csgraph)** Spatial algorithms and data structures **(scipy.spatial)** Special functions **(scipy.special)**

Statistical functions **(scipy.stats)**

Statistical functions for masked arrays **(scipy.stats.mstats)**

Low-level callback functions

An example of how to calculate effectively on SciPy

In this tutorial, Basic functions — SciPy v1.4.1 Reference Guide, you can find how to calculate polynomials, their derivatives, and integrals. Yes, by one line of code SciPy

calculates derivative and integral in symbolic form. Imagine how many lines of code you would need to do this without SciPy. This is why this library is valuable in Python:

>>> p = poly1d([3,4,5])

>>> print(p) 2

3 x + 4 x + 5

>>> print(p\*p)

4 3 2

9 x + 24 x + 46 x + 40 x + 25

>>> print(p.integ(k=6))

3 2

1 x + 2 x + 5 x + 6

>>> print(p.deriv()) 6 x + 4

>>> p([4, 5])

array([ 69, 100])

#### Applications:

* Multidimensional image operations
* Solving differential equations and the Fourier transform
* Optimization algorithms
* Linear algebra

# Numpy

In early 2005, programmer and data scientist Travis Oliphant wanted to unite the community around one project and created the NumPy library to replace the Numeric and NumArray libraries. NumPy was created based on the Numeric code. The Numeric code was rewritten to be easier to maintain, and new features could be added to the library. NumArray features have been added to NumPy. NumPy was originally part of the SciPy library. To allow other projects to use the NumPy library, its code was placed in a separate package.

The source code for NumPy is publicly available. NumPy is licensed under the BSD license.

#### Purpose of the NumPy library

Mathematical algorithms implemented in interpreted languages, for example, Python, often work much slower than the same algorithms implemented in compiled languages (for example, Fortran, C, and Java). The NumPy library provides implementations of computational algorithms in the form of functions and operators, optimized for working with multidimensional arrays. As a result, any algorithm that can be expressed as a sequence of operations on arrays (matrices) and implemented using NumPy works as fast as the equivalent code executed in MATLAB. If we compare *numpy* vs *math, we quickly find thatnumpy* has more advantages for computation methods compared to *math*.

#### Here are some of the features of Numpy:

* + - A powerful N-dimensional array object
    - Sophisticated (broadcasting) functions
    - Tools for integrating C/C++ and Fortran code
    - Useful linear algebra, Fourier transform, and random number capabilities

#### What’s the difference between a Python list and a NumPy array?

As described in the NumPy documentation, “NumPy gives you an enormous range of fast and efficient ways of creating arrays and manipulating numerical data inside them. While a Python list can contain different data types within a single list, all of the elements in a NumPy array should be homogenous. The mathematical operations that are meant to be performed on arrays would be extremely inefficient if the arrays weren’t homogenous.” **Numpy provides the following features to the user:**

* + - * Array objects
      * Constants
      * Universal functions (ufunc)
      * Routine
      * Packaging (numpy.distutils)
      * NumPy Distutils - Users Guide
      * NumPy C-API
      * NumPy internals
      * NumPy and SWIG

#### NumPy basics:

* + - * Data types
      * Array creation
      * I/O with NumPy
      * Indexing
      * Broadcasting
      * Byte-swapping
      * Structured arrays
      * Writing custom array containers
      * Subclassing ndarray

One of the main objects of NumPy is ndarray. It allows you to create multidimensional data arrays of the same type and perform operations on them with great speed. Unlike sequences in Python, arrays in NumPy have a fixed size, the elements of the array must be of the same type. You can apply various mathematical operations to arrays, which are performed more efficiently than for Python sequences. The next [example](https://numpy.org/devdocs/user/quickstart.html#simple-array-operations) shows how to work with linear algebra with NumPy. It is really simple and easy-to-understand for Python users.

>>> import numpy as np

>>> a = np.array([[1.0, 2.0], [3.0, 4.0]])

>>> print(a) [[ 1. 2.]

[ 3. 4.]]

>>> a.transpose() array([[ 1., 3.],

[ 2., 4.]])

>>> np.linalg.inv(a) array([[-2. , 1. ],

[ 1.5, -0.5]])

>>> u = np.eye(2) # unit 2x2 matrix; "eye" represents "I"

>>> u

array([[ 1., 0.],

[ 0., 1.]])

>>> j = np.array([[0.0, -1.0], [1.0, 0.0]])

>>> j @ j # matrix product array([[-1., 0.],

[ 0., -1.]])

>>> np.trace(u) # trace 2.0

>>> y = np.array([[5.], [7.]])

>>> np.linalg.solve(a, y) array([[-3.],

[ 4.]])

>>> np.linalg.eig(j)

(array([ 0.+1.j, 0.-1.j]), array([[ 0.70710678+0.j,0.70710678-0.j], [ 0.00000000-0.70710678j, 0.00000000+0.70710678j]]))

Numpy allows processing information without cycles. Please take a look at this [article](https://realpython.com/numpy-array-programming/) published by Brad Solomon about the advantages of Numpy: “It is sometimes

said that Python, compared to low-level languages such as C++, improves development time at the expense of runtime. Fortunately, there are a handful of ways to speed up operation runtime in Python without sacrificing ease of use. One option suited for fast numerical operations is NumPy, which deservedly bills itself as the fundamental package for scientific computing with Python.” It makes computation in Python really fast.

#### Applications:

* Extensively used in data analysis
* Creates powerful N-dimensional array
* Forms the base of other libraries, such as SciPy and [scikit-learn](https://www.simplilearn.com/tutorials/scikit-learn-tutorial/what-is-scikit-learn-and-how-to-install-it)
* Replacement of MATLAB when used with SciPy and matplotlib