

PYTHON PANDAS

***INDEX***

|  |  |  |
| --- | --- | --- |
| **S.NO** | **Topics** | **Page No** |
| **1** | [Introduction](#Introduction) | [3](#Introduction) |
| **2** | [Installation](#Installation) | [3](#Installation) |
| **3** | [Data structure](#Datastructure) | [3](#Datastructure) |
| **4** | [pandas.Series](#pandasseries) | [4](#pandasseries) |
| **5** | [pandas.DataFrame](#pandasdataframe) | [7](#pandasdataframe) |
| **6** | [Basic Functionality](#Basicfunctionality) | [10](#Basicfunctionality) |
| **7** | [Descriptive Statistics](#Descriptivestatistics) | [12](#Descriptivestatistics) |
| **8** | [Function Application](#Functionapplication) | [15](#Functionapplication) |
| **9** | [Statistical Functions](#statisticalfunction) | [17](#statisticalfunction) |
| **10** | [GroupBy](#groupby) | [19](#groupby) |
| **11** | [Merging/Joining](#Mergingjoining) | [20](#Mergingjoining) |

# **Introduction:**

**Pandas** is a Python package providing ***fast***, ***flexible***, and ***expressive*** data structures designed to make working with **structured** (tabular, multidimensional, potentially heterogeneous) and **time series** data. It aims to be the fundamental ***high-level*** building block for doing practical, real world data analysis in Python. Additionally, it has the broader goal of becoming the most powerful and flexible open source data analysis / manipulation tool available in any language.

Using Pandas, we can accomplish five typical steps in the processing and analysis of data, regardless of the origin of data — load, prepare, manipulate, model, and analyse.

pandas are well suited for different kinds of data:

**>**Tabular data with heterogeneously typed columns, as in an SQL table or Excel

**>**Ordered and unordered (not necessarily fixed frequency) time series data.

**>**Arbitrary matrix data (homogeneously typed or heterogeneous) with row and column labels.

# **1.1 Installation:**

Pandas can be installed using pip command in command prompt by following command.

**Note:** Please ensure that you have installed python >= 3.8.2



# **1.2 Data structure:**

Data structure in pandas gives the flexibility to work with shape or dimension of the data. Pandas has below listed data structure in it. These data structures are built on top of NumPy array hence it will be fast.

All Pandas data structure are value mutable and except series all are size mutable. DataFrame is widely used and one of the most important data structures.

|  |  |  |
| --- | --- | --- |
| **Data Structure** | **Dimensions** | **Explanation** |
| Series | 1 | 1D is same kind array and size is immutable |
| DataFrame | 2 | 2D is different type column and size is mutable |

Handling more than two dimensions of array using these data structure will be helpful to think the of Index (rows) and columns rather than axes 0 and axes 1.

**Series:**

Series are one dimensional array like structure with homogeneous data. For example, A series contains collection of integers 5,6,7,10,29

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 5 | 6 | 7 | 10 | 29 |

**Features:**

**>**Homogeneous data

**>**Size Immutable

**>**Values of Data Mutable

**Data Frame:**

Data Frame is 2-Dimensional array with heterogeneous data. For example,

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Age** | **Gender** | **Height** (in ft) |
| Abc | 20 | M | 6.1 |
| Def | 24 | F | 5.9 |
| Ghi | 27 | M | 5.4 |
| JKL | 10 | M | 4.2 |

This table contains different datatypes like Name (***String***), Age (***Int***), Gender (***String*)** and

Height (***Float***).

**Features:**

**>**Heterogeneous data

**>**Data Mutable

**>**Size Mutable

# **pandas.Series:**

Series are 1-D array which will hold any kind of datatype (***int, string, float***, python ***objects***, etc). The axis labels are collectively called index.

**Syntax:**

Pandas.Series(data, index, dtype, copy)

Parameter Description:

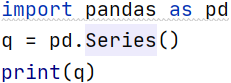
**>*data*** – contains ndarray, list, constants

**>index** – must unique and hashable, same length as data

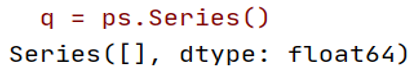
**>dtype** – data type

**>copy** – copy data

Series can be created using array, dict & constants. Below example illustrate creating an empty Pandas Series.



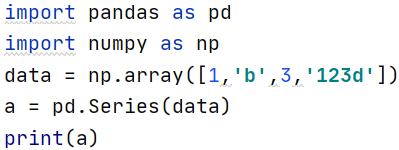
Output:



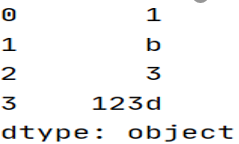
**Creating Series from array [With Implicit index]**:

Creating Pandas Series from array is simple as creating an list in python. we can provide list of input as show below as input to Pandas.Series method. We all know that Pandas data structure holds index with it by default. So we have optional parameter “index” this will allow you to specify index explicitly for your Series object. It can be any of the python data type such as (***int***, ***float***, ***complex***, ***str*** and etc.)

Note: Pandas Series will assign index by default if you are not specifying any index to series object. And the index starts from 0 as usual in python standard. For the below array length is 4 so index will be 0 to 3

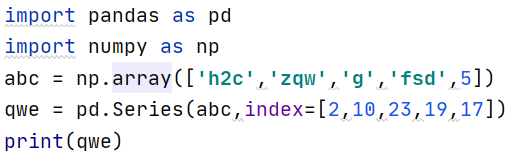


Output:

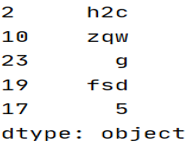


**Creating Series from array [With Explicit index]**:

As we know that the index parameter is an optional parameter when creating Series, which means we can provide explicit index also. As I mentioned earlier it can be any python datatype [**except arbitrary objects such as list, set and dictionary**]. Below example will illustrate how to create and Pandas Series by providing Explicit indexing.



Output:

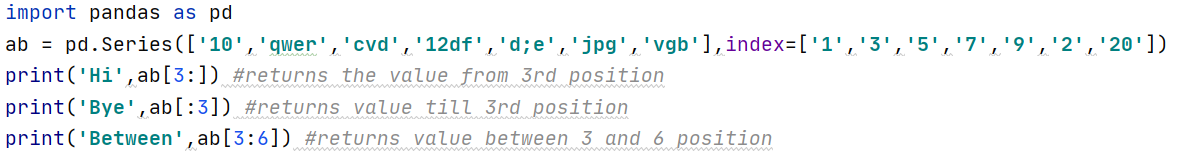


**Slicing & Indexing Pandas Series**:

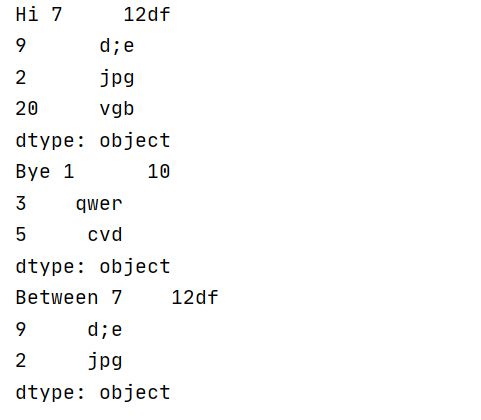
Slicing/Indexing in Pandas series as same as indexing/slicing in *ndarray* or any python sequence object. If you see the below example, we are using the “[:]” operation to slice given Series elements.

**Note:** *you can see a difference here where the index of the series is some random digits but when we specify index number from 3:6 it picks the right values. This is due to a concept called implicit and explicit indexing in Pandas. I know this will be some what confusing will see in detail about this topic later on this session.*

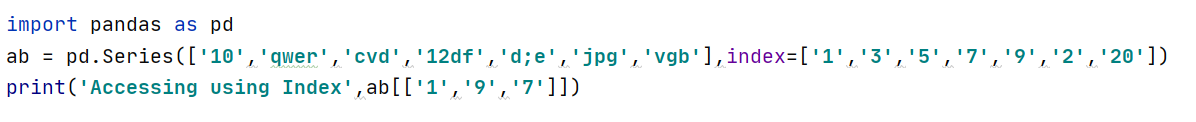
**Slicing in Pandas Object:**



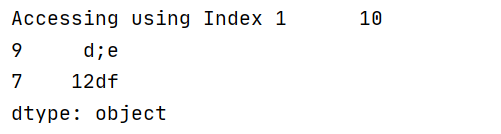
Output:



**Indexing in Pandas Series:**



Output:



Till now what we saw is the basics of Pandas series and will continue to see more in detail in Pandas DataFrame object.

# **pandas.DataFrame:**

Pandas dataframe is 2-Dimensional data structure as that contains multiple series objects associated with it as rows and columns. One important thing in Pandas DataFrame is it shares common index between all associated series objects.

**Syntax:**

pandas.DataFrame(data, index, columns, dtype, copy)

**> Data**: Contains ndarray, series, map, lists, dict, constants and also another DataFrame.

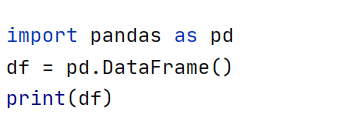
**> index**: For the row labels, the Index to be used for the resulting frame is Optional Default np.arange(n) if no index is passed.

**> columns**: For column labels, the optional default syntax is - np.arange(n). This is only true if no index is passed.

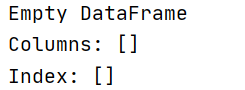
**> dtype**: Data type of each column.

**> copy**: This command (or whatever it is) is used for copying of data, if the default is False.

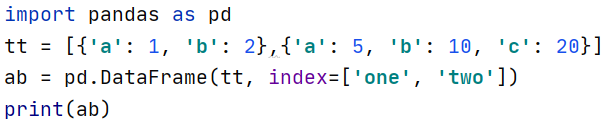
**Creating Empty dataframe Example:**



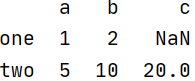
Output:



**Creating Dataframe from dictionary:**

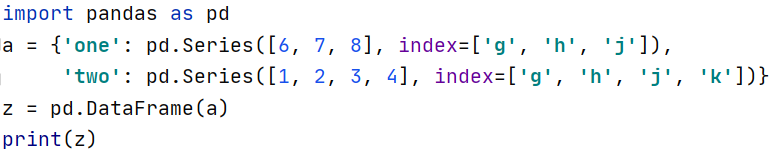


Output:

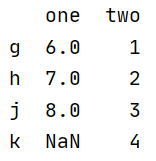


**Note:**  NaN (Not a Number) is appended in missing areas

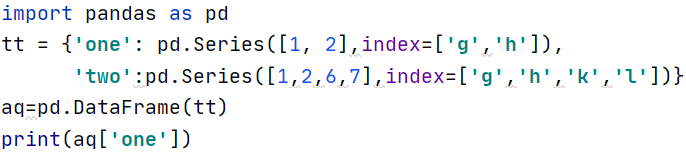
**Create a DataFrame from Dict of Series objects:**



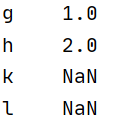
Output:



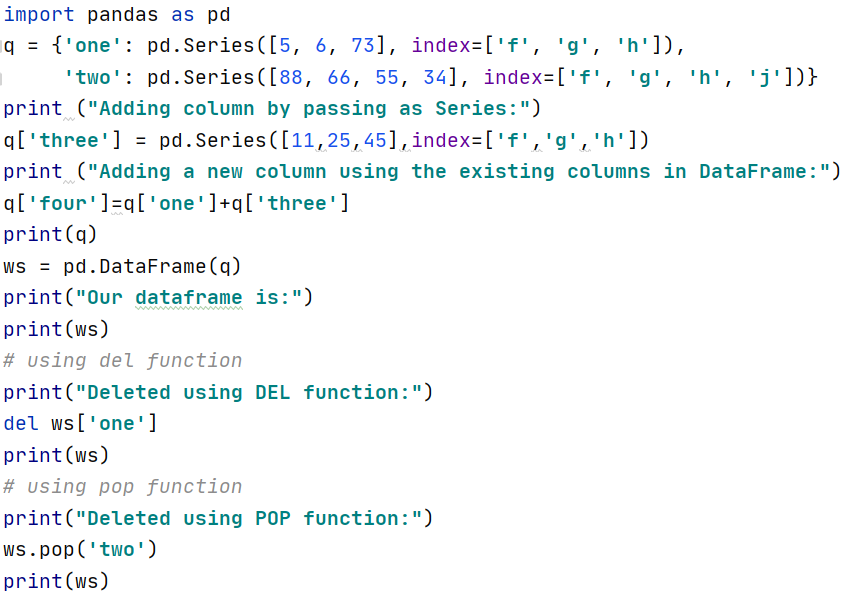
**Column Selection in Pandas Dataframe:**



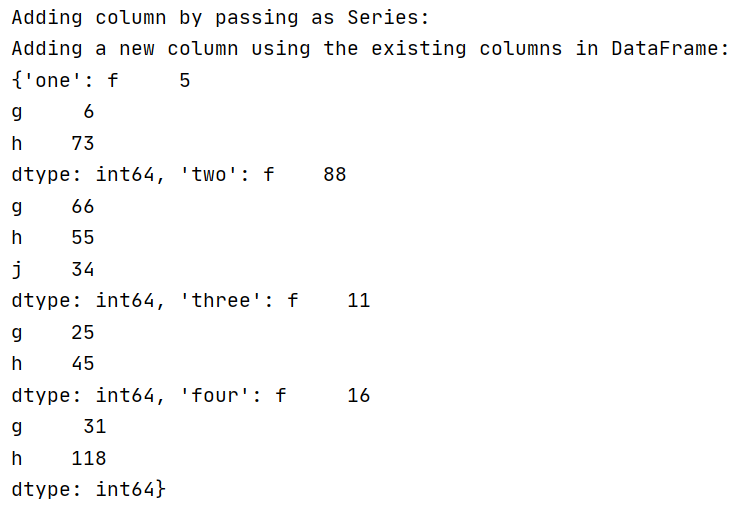
Output:

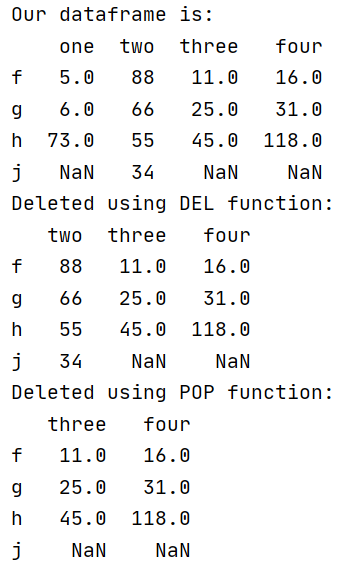


**Column Addition & Deletion in DataFrame:** **del()** & **pop()**



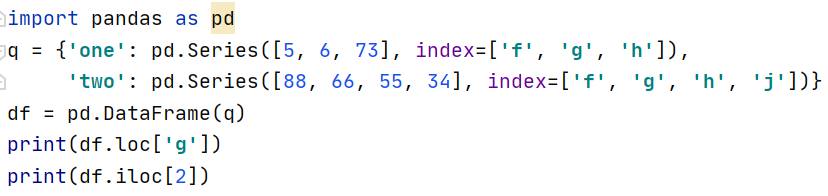
Output:



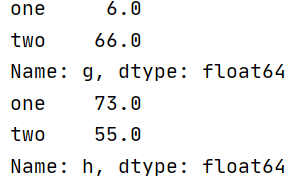


**Row Selection, Addition, & Deletion in Pandas Dataframe:**

Rows can be selected by passing row label to a ‘**loc**’ and ‘**iloc**’ function

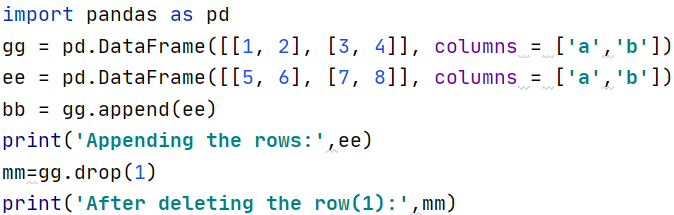


Output:

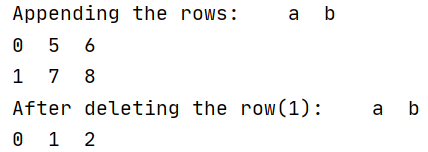


Example 02:

using **append()** and **drop()**

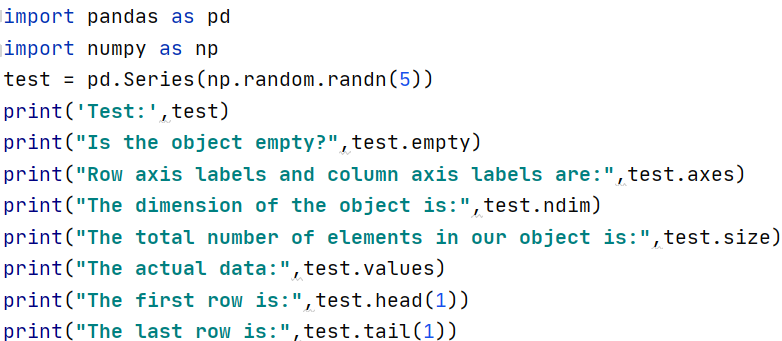


Output:

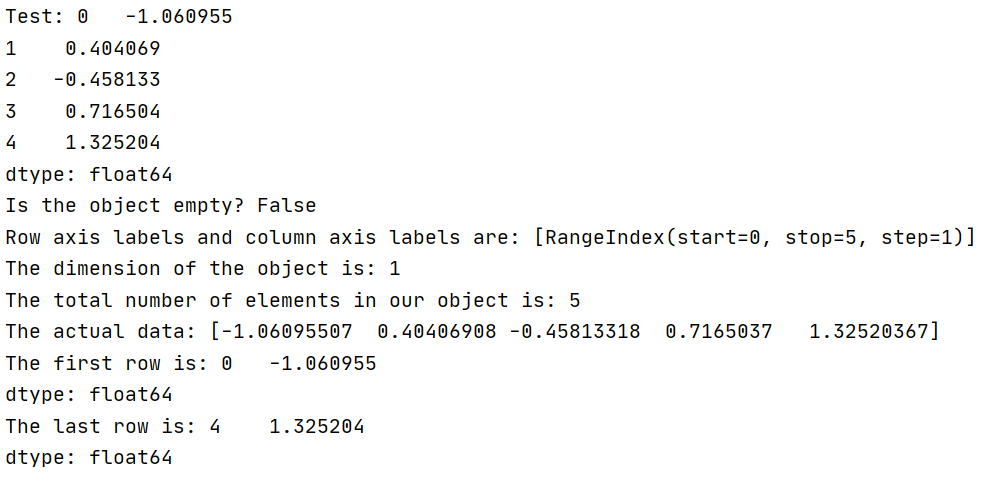


# **Series Basic Functionality:**

axes, dtype empty, ndim, size, values, head(), tail() with this functionality we can check the data by below example

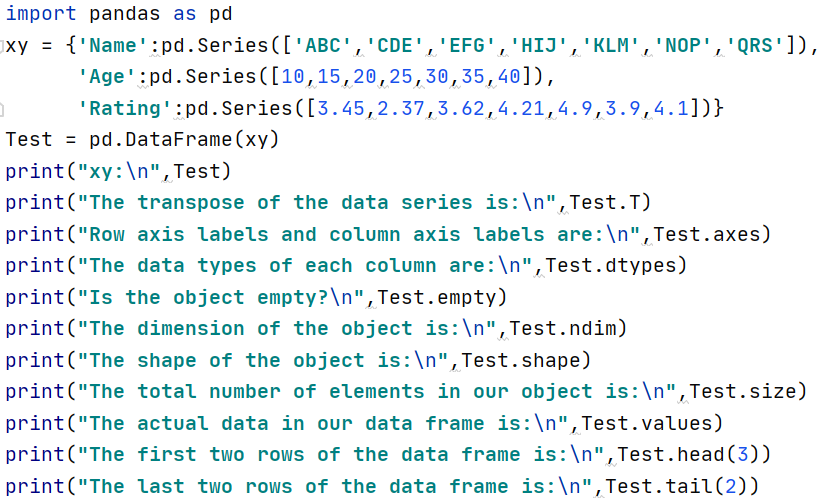


Output:

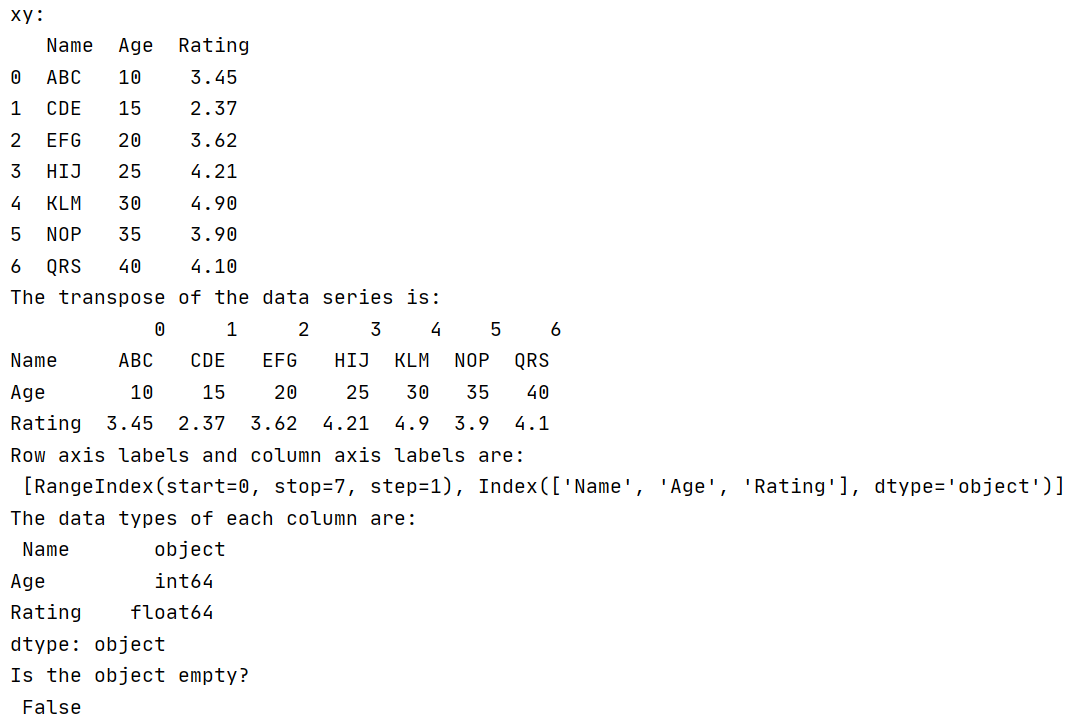


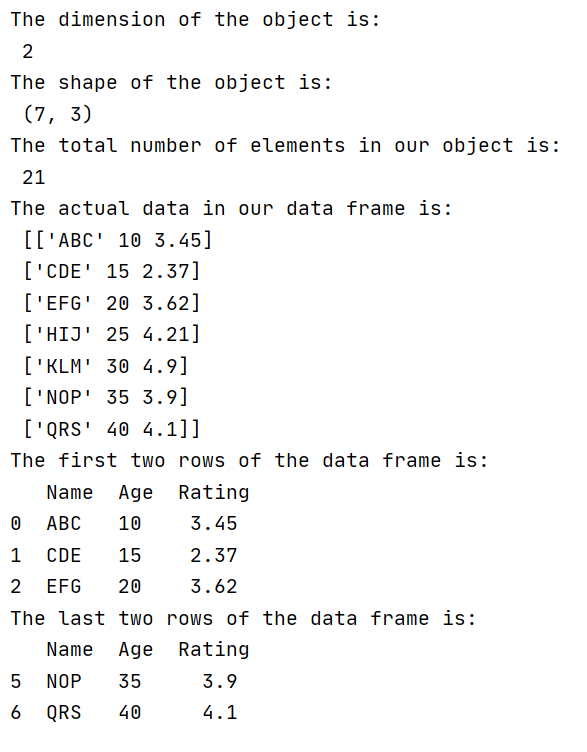
**DataFrame Basic Functionality:**

T, axes, dtypes, empty, ndim, shape, size, values, head(), tail() with this functionality we can check the data by below example



Output:

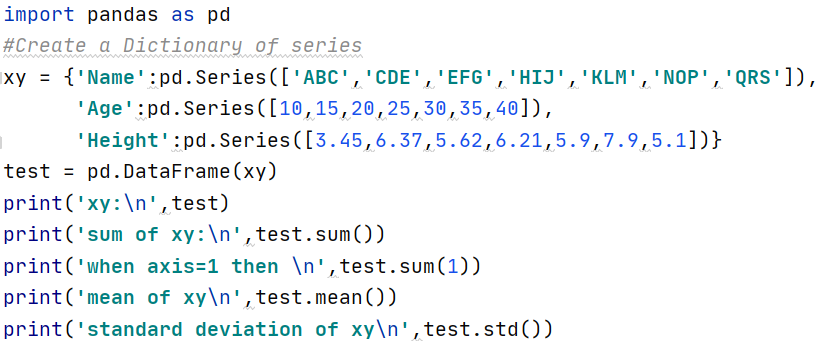




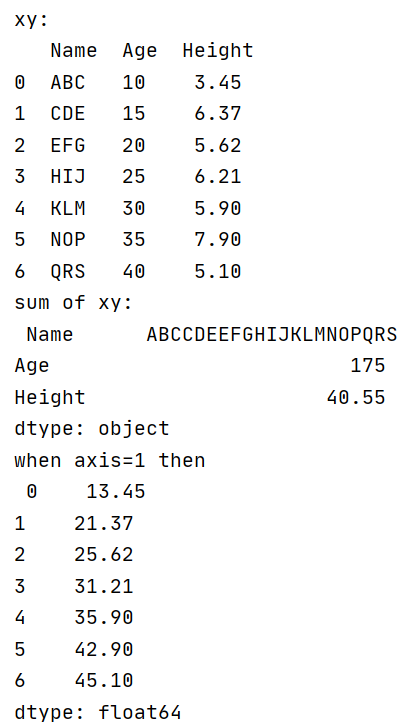
# Descriptive Statistics:

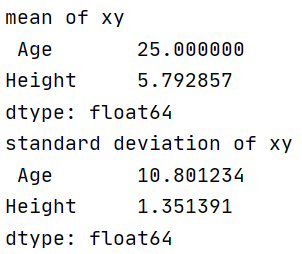
Descriptive statistics is to show and summarize the data in meaningful. Let us compute the data in DataFrame by using the below operations

Example:



Output:





Other Statistical Function:

**>**count() - Number of non-null observations

**>** sum() - Sum of values

**>** mean() - Mean of Values

**>** median() - Median of Values

**>** mode() - Mode of values

**>** std() - Standard Deviation of the Values

**>** min() - Minimum Value

**>** max() - Maximum Value

**>** abs() - Absolute Value

**>** prod() - Product of Values

**>** cumsum() - Cumulative Sum

**>** cumprod() - Cumulative Product

Note − Since DataFrame is a Heterogeneous data structure. Generic operations don’t work with all functions.

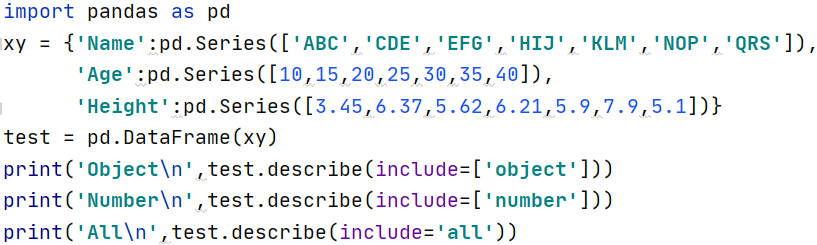
Functions like sum(), cumsum() work with both numeric and character (or) string data elements without any error. Though n practice, character aggregations are never used generally, these functions do not throw any exception.

Functions like abs(), cumprod() throw exception when the DataFrame contains character or string data because such operations cannot be performed.

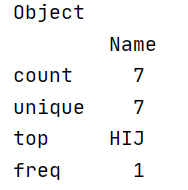
**Summarizing Data:**

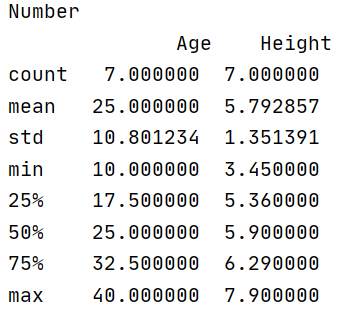
Calculating the summary of statistics using the describe() function.

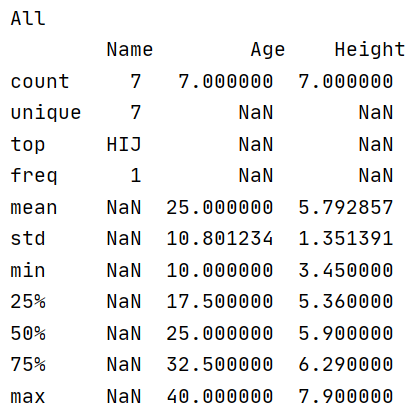
Example:



Output:







# **Function Application:**

Operation performed on dataframe using below types of function application.

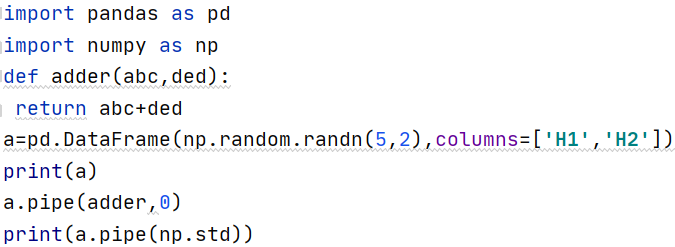
**>**Table wise Function Application: **pipe()**

**>**Row or Column Wise Function Application: **apply()**

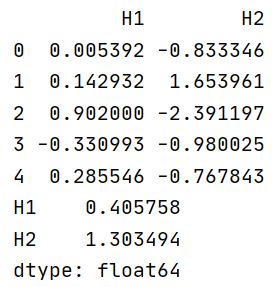
**>**Element wise Function Application: **applymap()**

# **Table wise Function Application:**

Operation is performed using function and appropriate number of parameters as pipe arguments. Then operation done in the dataframe.

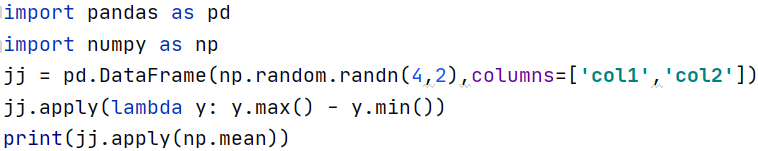


Output:

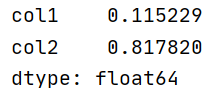


**Row or Column Wise Function Application:**

Arbitrary functions can be applied along the axes of a DataFrame or Panel using the apply() method, which, like the descriptive statistics methods, takes an optional axis argument. By default, the operation performs column wise, taking each column as an array-like.

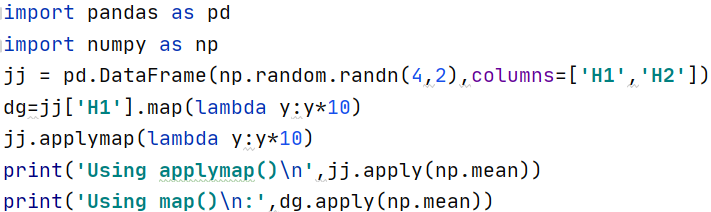


Output:

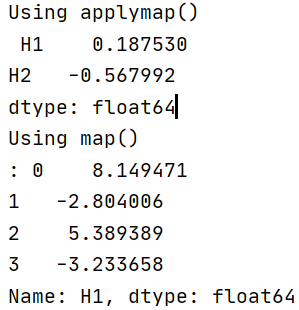


**Element Wise Function Application:**

Not all functions can be vectorized (neither the NumPy arrays which return another array nor any value), the methods applymap() on DataFrame and analogously map() on Series accept any Python function taking a single value and returning a single value.



Output:



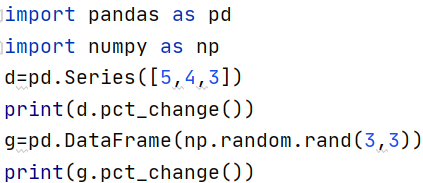
# Statistical Functions:

Statistical methods will help to understand and analyse the behaviour of data. It can be using the statistical function which is applied on objects.

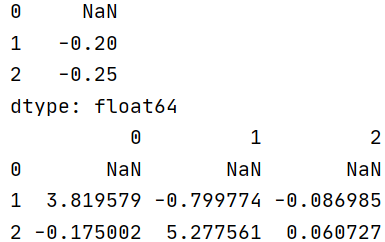
**pct\_change():**

Comparing every element with the prior element.

Example:



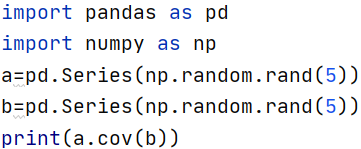
Output:



**Covariance:**

Covariance is a measure of how much two variables vary together.

Example:



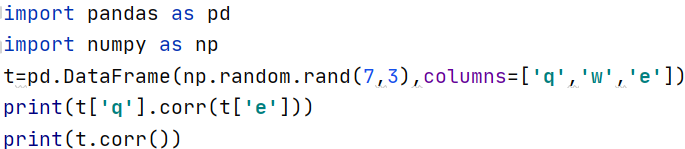
Output:



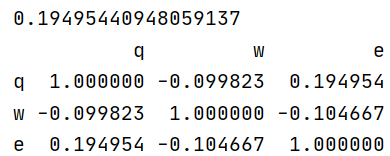
**Correlation:**

Correlation shows linear relationship between two values. It is calculated using Pearson (default), spearman, Kendall.

Example:



Output:

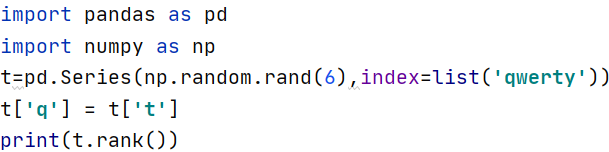


**Data Ranking:**

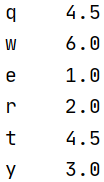
Data Ranking produces ranking for each element in the array of elements. In case of ties, assigns the mean rank.

Rank optionally takes a parameter ascending which by default is true; when false, data is reverse-ranked, with larger values assigned a smaller rank.

Example:



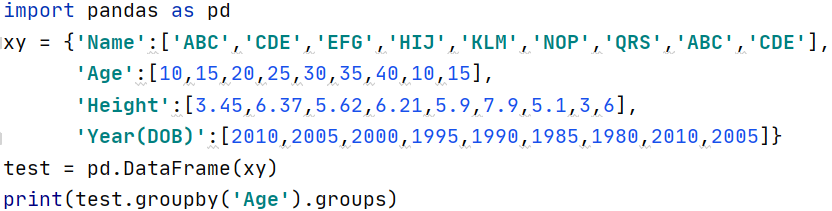
Output:



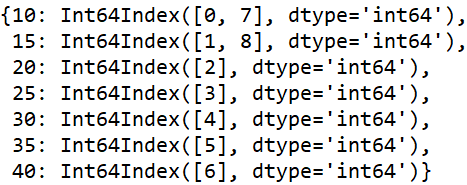
# GroupBy:

Pandas can also perform SQL operation like group by using dataframe objects.

Example:



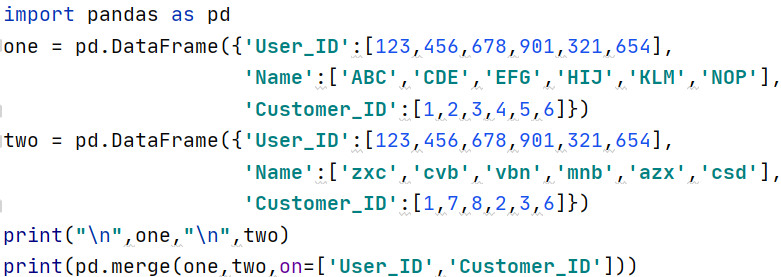
Output:



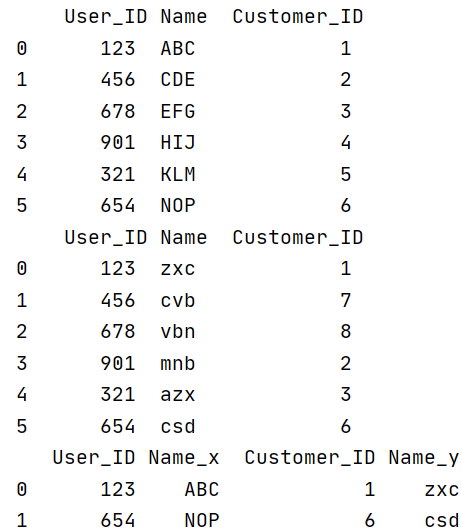
Merging/Joining:

Pandas has feature to do in-memory join operation which is like sql database.

merge - which perform join operation between dataframe objects. Example:



Output:

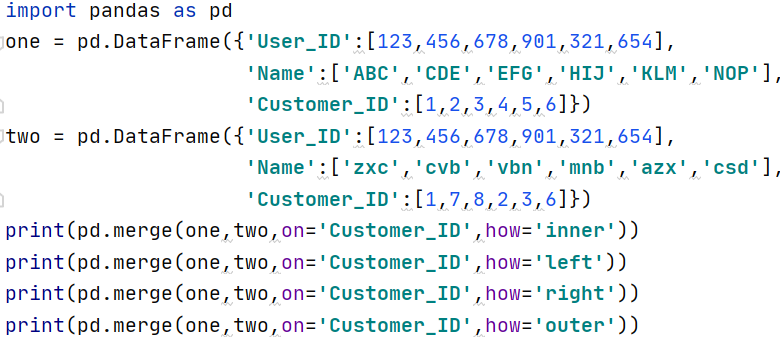


Merge using 'how':

The how argument to merge specifies how to determine which keys are to be included in the resulting table. If a key combination does not appear in either the left or the right tables, the values in the joined table will be NA.

|  |  |  |
| --- | --- | --- |
| Merge Method | SQL | Description |
| Left | LEFT OUTER JOIN | Use keys from left object |
| Right | RIGHT OUTER JOIN | Use keys from right object |
| Outer | FULL OUTER JOIN | Use union of keys |
| Inner | INNER JOIN | Use intersection of keys |

Example:



Output:

