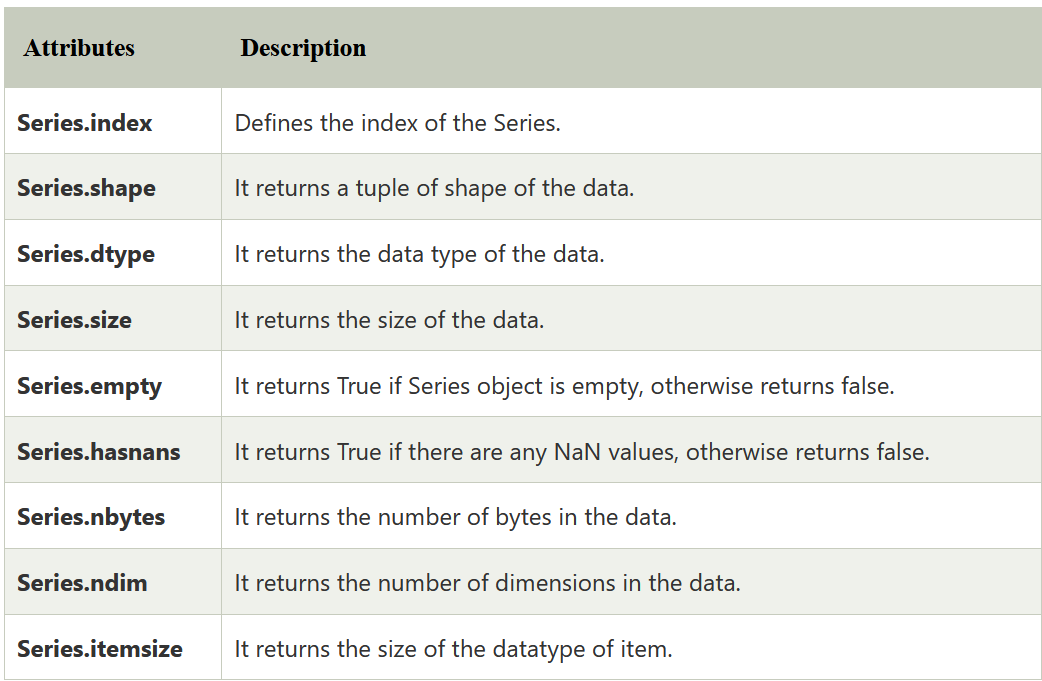
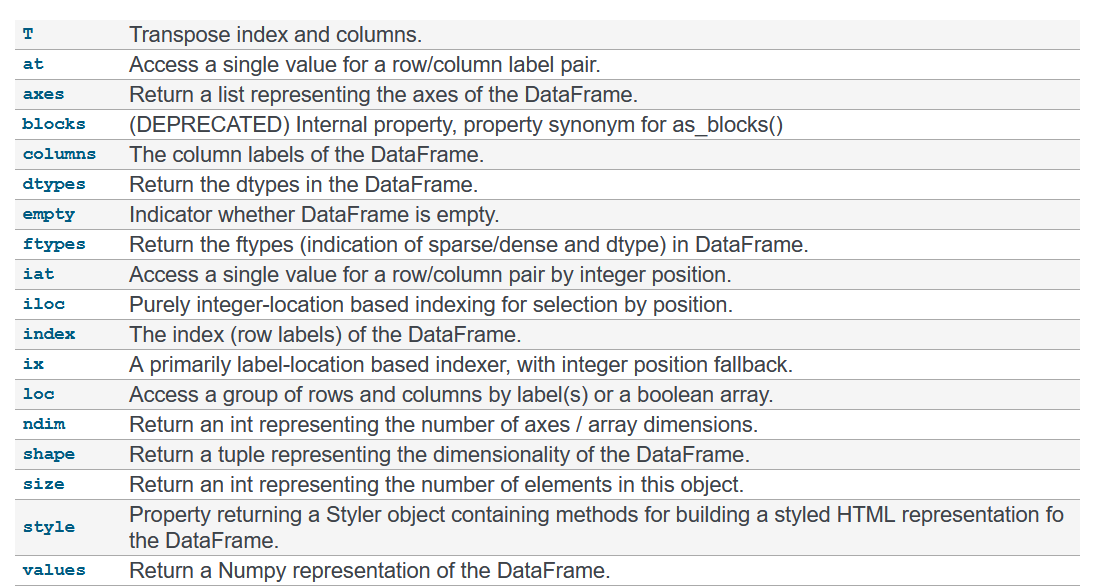
Pandas Series attributes:

Pandas series have below attributes-



**Pandas Dataframe attributes**

Pandas dataframe have below attributes-



**Pandas Index**

Pandas Index is an immutable ndarray implementing an ordered, sliceable set. It is the basic object which stores the axis labels for all pandas objects.

pandas.Index(data=None, dtype=None, copy=False, name=None, tupleize\_cols=True, \*\*kwargs)

**data**: array-like (1-dimensional)

**dtype**: NumPy dtype (default: object), If dtype is None, we find the dtype that best fits the data. If an actual dtype is provided, we coerce to that dtype if it’s safe. Otherwise, an error will be raised.

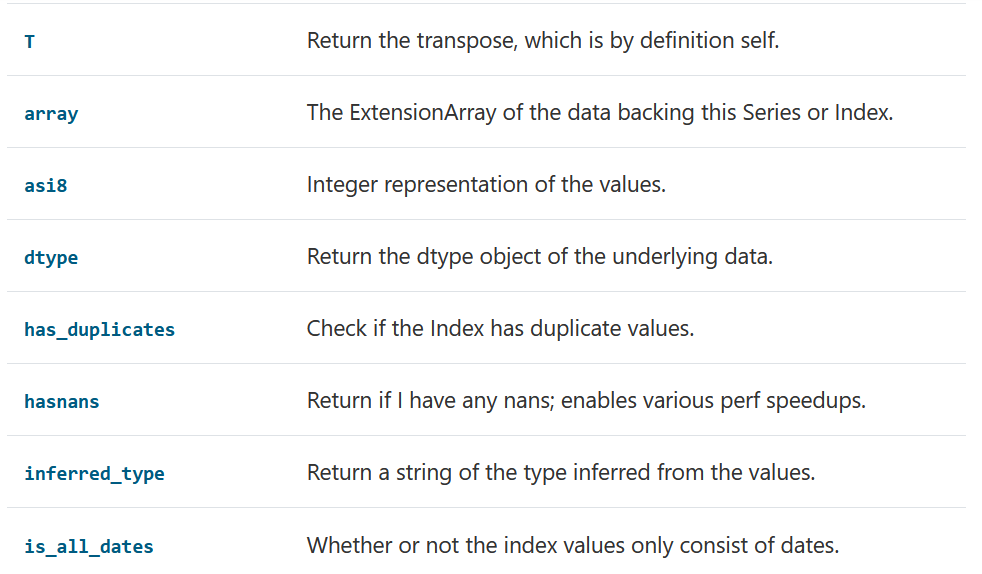
**copy**: bool, Make a copy of input ndarray.

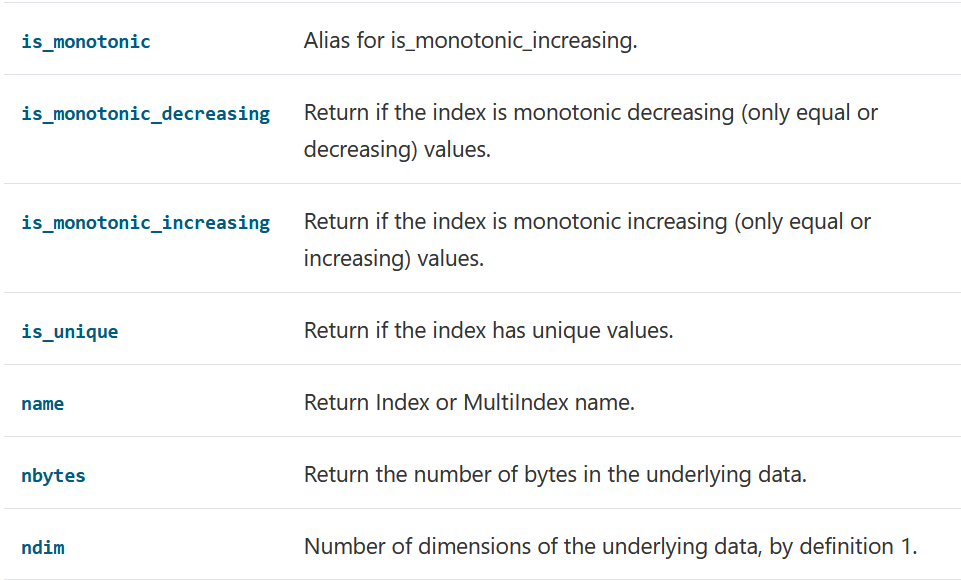
idx = pd.Index(['Jan', 'Feb', 'Mar', 'Apr', 'May'])

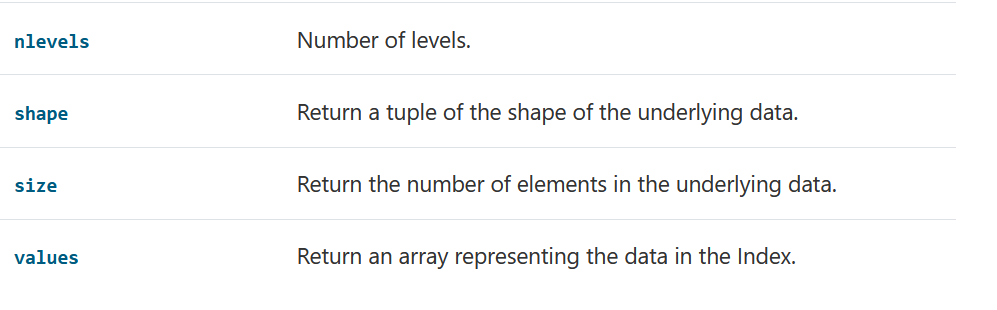
# Print the index

print(idx)

**Attributes of pandas Index**







**Selection methods for pandas Indexes**

Index.get\_loc(key, method=None, tolerance=None)

Returns loc: int if unique index, slice if monotonic index, else mask

**Key**: labels or datas

**default**: exact matches only.

**pad / ffill**: find the PREVIOUS index value if no exact match.

**backfill / bfill**: use NEXT index value if no exact match

**nearest**: use the NEAREST index value if no exact match. Tied distances are broken by preferring the larger index value.

**tolerance**: int or float, optional

**Example**

pd.Index(list('abc'))

unique\_index.get\_loc('b') #1

**Example**

monotonic\_index = pd.Index(list('abbc'))

monotonic\_index.get\_loc('b') #slice(1, 3, None)

**Example**

non\_monotonic\_index = pd.Index(list('abcb'))

non\_monotonic\_index.get\_loc('b') # array([False, True, False, True]) --- this is mask

Index.get\_value(series, key) ------ Returns series or scalar

Fast lookup of value from 1-dimensional ndarray. Only use this if you know what you’re doing.

Index.get\_slice\_bound(label, side, kind=None)

Calculate slice bound that corresponds to given label.

Returns leftmost (one-past-the-rightmost if side=='right') position of given label.

Returns: int, index od label

Index.isin(values, level=None)

Return a boolean array where the index values are in values.

Index.slice\_locs(start=None, end=None, step=None, kind=None)

Compute slice locations for input labels. Returns start,end as integer

start: label, default None

end: label, default None

step: int, defaults None

In output first values will be index value of start argument and second value will be position value of end argument.

**Note:**

This method only works if the index is monotonic or unique.

Example:

idx = pd.Index(list('abcd'))

idx.slice\_locs(start='b', end='c') #(1, 3) 1--> index of b, 3 --> position of c

In output first values will be index value of start argument and second value will be position value of end argument.

**Question ------ Good**

Write a Pandas program to get the positions of items of a given series(series2) in another given series(series1).

<https://www.w3resource.com/python-exercises/pandas/python-pandas-data-series-exercise-23.php>

series1 = pd.Series([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])

series2 = pd.Series([1, 3, 5, 7, 10])

print("Original Series:")

print(series1)

print(series2)

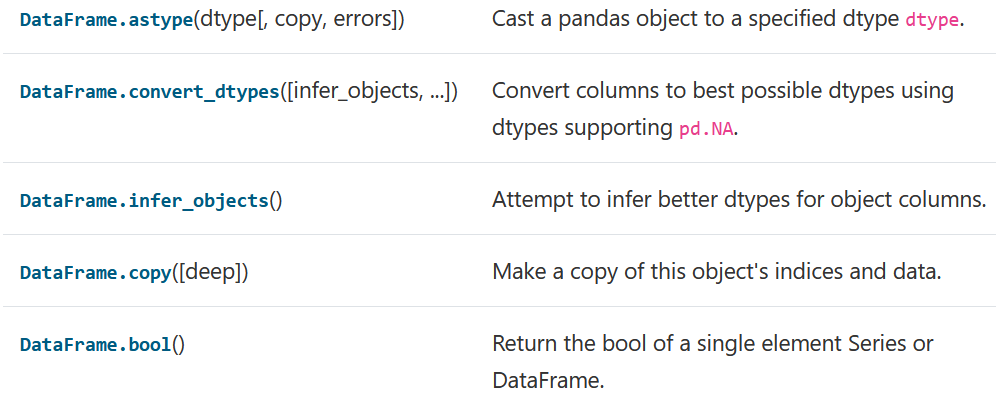
**Solution**

result = [pd.Index(series1).get\_loc(i) for i in series2]

print("Positions of items of series2 in series1:")

print(result)

**Dataframe conversion function**



DataFrame.astype(dtype, copy=True, errors='raise')

Cast a pandas object to a specified dtype dtype.

**dtype**: data type which to be casted,it's umpy.dtype or Python type.

**copy**: bool, default True, Returns a copy when copy=True

**errors**: {‘raise’, ‘ignore’}, default. Control raising of exceptions on invalid data for provided dtype.

raise : allow exceptions to be raised

ignore : suppress exceptions. On error return original object.

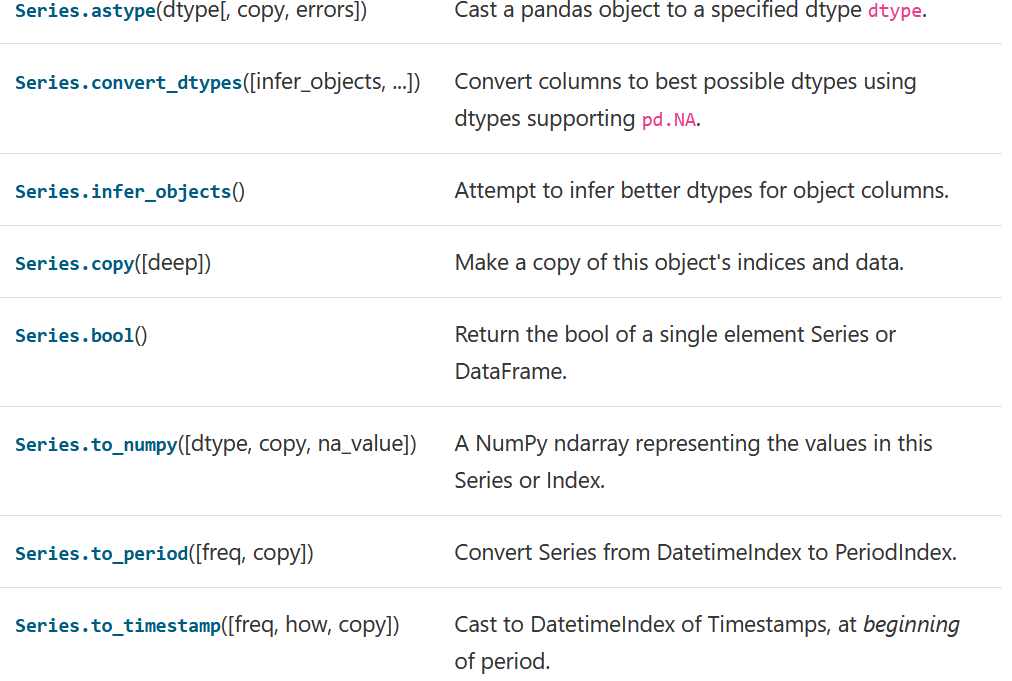
DataFrame.copy(deep=True)

Make copy (deep or shallow copy) of dataframe and return it.

**deep**: bool, default True

**Series Conversion functions**

We have below methods for conversion will see few of them.





Series.astype() --- same as df

Series.copy() ---- same as df

Series.to\_numpy(dtype=None, copy=False, na\_value=NoDefault.no\_default, \*\*kwargs)

Gives a NumPy ndarray representing the values in this Series or Index.

*dtype*: str or numpy.dtype, optional

*copy*: bool, default False

*na\_value* : Any, optional

Series.to\_timestamp(freq=None, how='start', copy=True)

Cast to DatetimeIndex of Timestamps, at beginning of period

Series.to\_list()

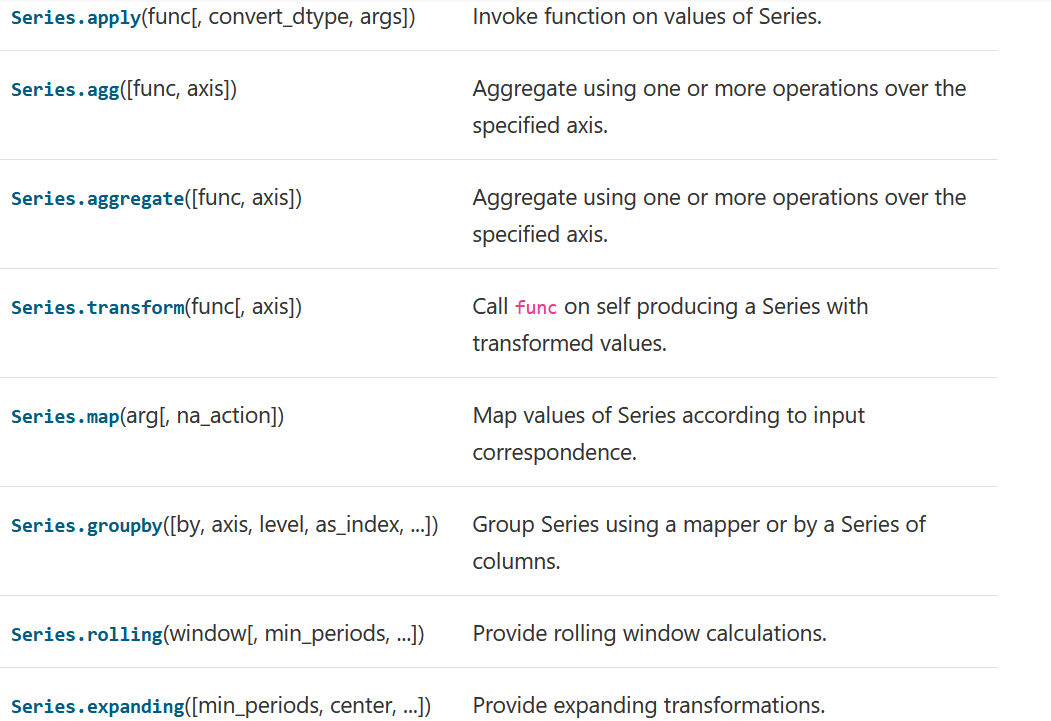
Convert numpy series to a list types and returns a list of values.

##########################################

# Series Function application, GroupBy & window #

##########################################

We have below function but we will see few commonly used.



Series.apply(func, convert\_dtype=True, args=(), \*\*kwargs)

It invokes ‘func’ on on called series and **returns dataframe or series**. Function can be ufunc(universal function) or Python function that only works on single values.

*func*: function --Python function or NumPy ufunc to apply

*args*: tuple --Positional arguments passed to func after the series value.

If func returns a Series object the result will be a DataFrame.

Question:

Below temperature contains temperature for given city, square the temperature value for each city.

s = pd.Series([20, 21, 12],index=['London', 'New York', 'Helsinki'])

Answer:

print(s.apply(lambda x:x\*x))

**Question**:

For above temperature series, add 5 in temperature.

Answer:

We create a custom method for adding and will pass 5 as argument. ---- **Try using transform()**

def my\_add(x,arg):

    return x+5

print(s.apply(my\_add,args=(5,)))

Series.agg(func=None, axis=0, \*args, \*\*kwargs)

Aggregate using one or more operations over the specified axis.

It returns scalar, Series or DataFrame

*scalar* : when Series.agg is called with single function

*Series* : when DataFrame.agg is called with a single function or series with many function

*DataFrame* : when DataFrame.agg is called with several functions

*func*: function, str, list or dict----Function to use for aggregating the data.

*axis*: {0 or ‘index’} ---Parameter needed for compatibility with DataFrame.

*\*args*: ---Positional arguments to pass to func.

Example:

s = pd.Series([1, 2, 3, 4])

s.agg('min') # 1------------ it will be scalar

s.agg(['min', 'max']) #this will be series

min 1

max 4

dtype: int64

Series.aggregate(func=None, axis=0, \*args, \*\*kwargs)

Aggregate using one or more function. Exactly same as agg, agg is alias of aggregate.

Series.transform(func, axis=0, \*args, \*\*kwargs)

Call func on self producing a Series with transformed values.Produced Series will have same axis length as self. It A Series that must have the same length as self.

*func*: function, str, list-like or dict-like.

*axis*: {0 or ‘index’}

*\*args*: Positional arguments to pass to func.

**Example**:

For given dataframe add 1 for each value.

df = pd.DataFrame({'A': range(3), 'B': range(1, 4)})

Asnwer:

df.transform(lambda x: x + 1) #

Series.map(arg, na\_action=None)

Map values of Series according to input correspondence and retuns series

Used for substituting each value in a Series with another value, that may be derived from a function.

*map accepts a dict or a Series.*

*arg*: function, collections.abc.Mapping subclass or Series

Mapping correspondence.

*na\_action*: {None, ‘ignore’}, default None

If ‘ignore’, propagate NaN values, without passing them to the mapping correspondence.

Example:

For given series map cat to kitten and dog to puppy.

s = pd.Series(['cat', 'dog', np.nan, 'rabbit'])

Answer:

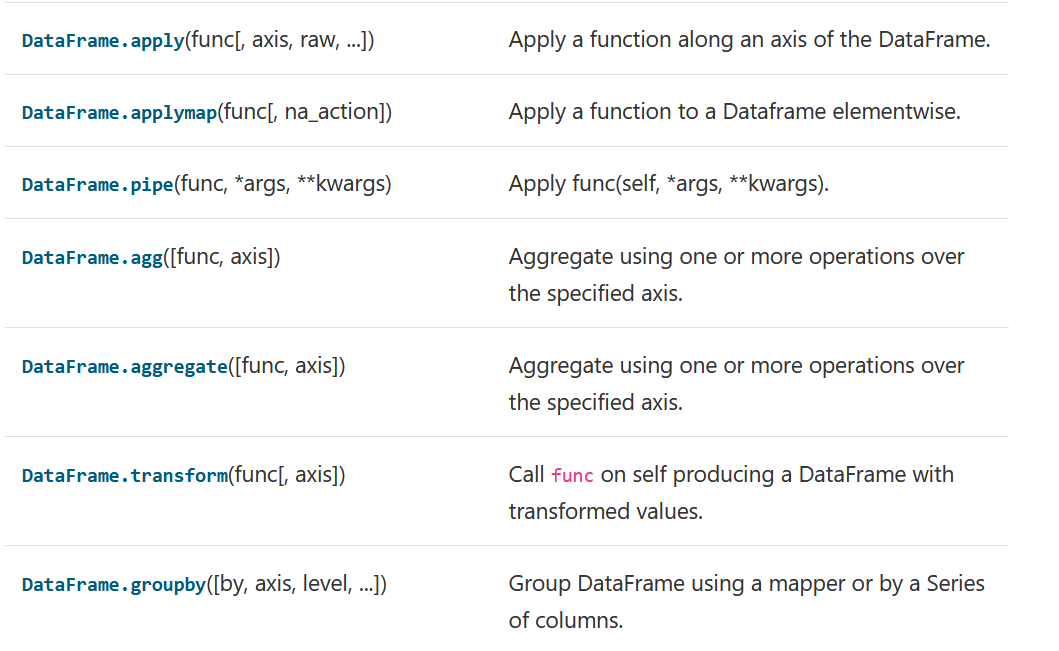
|  |  |
| --- | --- |
| print(s.map({'cat': 'kitten', 'dog': 'puppy'})) |  |

##############################################

# Dataframe Function application, GroupBy & window #

##############################################

We have below methods for dataframe grouping and application. They are almost same as that of series with minor defference. We will check few



DataFrame.apply(func, axis=0, raw=False, result\_type=None, args=(), \*\*kwargs)

Apply a function (func) along an axis of the DataFrame.

*func*: function --- function to be applied

*axis*: {0 or ‘index’, 1 or ‘columns’}, **default 0**

*0 or ‘index’: apply function to each column*

1 or ‘columns’: apply function to each row

Example:

For below dataframe calculate sum for each row and each column.

df = pd.DataFrame([[4, 9]] \* 3, columns=['A', 'B'])

For each column

|  |  |
| --- | --- |
| print(df.apply(np.sum)) #defaults for col |  |

For each row

|  |  |
| --- | --- |
| print(df.apply(np.sum,axis=1))  axis=1 means for each row |  |

DataFrame.applymap(func, na\_action=None, \*\*kwargs)

Apply a function to a Dataframe **elementwise or on each element.**

This method applies a function that accepts and returns a scalar to every element of a DataFrame.

----------This is same as of pandas series map function.

Example:

Squre the value of each element of given dataframe.

pd.DataFrame([[1, 2.12], [3.356, 4.567]])

Answer:

df.applymap(lambda x: x\*\*2)

DataFrame.agg(func=None, axis=0, \*args, \*\*kwargs)

Aggregate using one or more operations over the specified axis.

It’s same as series agg/aggregate function, additionally we have to specify the axis.

It returns scalar, Series or DataFrame.

*scalar* : when Series.agg is called with single function

*Series* : when DataFrame.agg is called with a single function

*DataFrame* : when DataFrame.agg is called with several functions

*axis*: {0 or ‘index’, 1 or ‘columns’}, **default 0**

*0 or ‘index’: apply function to each column.*

*1 or ‘columns’: apply function to each row.*

DataFrame.transform(func, axis=0, \*args, \*\*kwargs)

Call func on self producing a DataFrame with transformed values.

Produced DataFrame will have same axis length as self.

It accepts same parameter as that of series with additional axis argument.

*axis*: {0 or ‘index’, 1 or ‘columns’}, **default 0**

*0 or ‘index’: apply function to each column.*

*1 or ‘columns’: apply function to each row.*

It returns dataframe that have same length as that of self.