pandas.DataFrame.apply

DataFrame.apply(*func*, *axis=0*, *raw=False*, *result_type=None*, *args=()*, *by_row='compat'*, *engine='python'*, *engine_kwargs=None*, ***kwargs*) [source]

Apply a function along an axis of the DataFrame.

Objects passed to the function are Series objects whose index is either the DataFrame's index (`axis=0`) or the DataFrame's columns (`axis=1`). By default (`result_type=None`), the final return type is inferred from the return type of the applied function. Otherwise, it depends on the *result_type* argument.

Parameters:

func : *function*

Function to apply to each column or row.

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

Axis along which the function is applied:

- 0 or 'index': apply function to each column.
- 1 or 'columns': apply function to each row.

raw : *bool, default False*

Determines if row or column is passed as a Series or ndarray object:

- `False` : passes each row or column as a Series to the function.
- `True` : the passed function will receive ndarray objects instead. If you are just applying a NumPy reduction function this will achieve much better performance.

result_type : *{'expand', 'reduce', 'broadcast', None}, default None*

These only act when `axis=1` (columns):

- 'expand' : list-like results will be turned into columns.
- 'reduce' : returns a Series if possible rather than expanding list-like results. This is the opposite of 'expand'.

Skip to main content

- ‘broadcast’ : results will be broadcast to the original shape of the DataFrame, the original index and columns will be retained.

The default behaviour (None) depends on the return value of the applied function: list-like results will be returned as a Series of those. However if the apply function returns a Series these are expanded to columns.

args : *tuple*

Positional arguments to pass to *func* in addition to the array/series.

by_row : *False or “compat”, default “compat”*

Only has an effect when `func` is a listlike or dictlike of funcs and the func isn’t a string. If “compat”, will if possible first translate the func into pandas methods (e.g. `Series().apply(np.sum)` will be translated to `Series().sum()`). If that doesn’t work, will try call to apply again with `by_row=True` and if that fails, will call apply again with `by_row=False` (backward compatible). If False, the funcs will be passed the whole Series at once.

> **Added in version 2.1.0.**

engine : *{‘python’, ‘numba’}, default ‘python’*

Choose between the python (default) engine or the numba engine in apply.

The numba engine will attempt to JIT compile the passed function, which may result in speedups for large DataFrames. It also supports the following engine_kwargs :

- nopython (compile the function in nopython mode)
- nogil (release the GIL inside the JIT compiled function)
- parallel (try to apply the function in parallel over the DataFrame)

  Note: Due to limitations within numba/how pandas interfaces with numba, you should only use this if raw=True

Note: The numba compiler only supports a subset of valid Python/numpy operations. Please read more about the supported python features and supported numpy features in numba to learn what you can or cannot use in the passed function.

> **Added in version 2.2.0.**

engine_kwargs : *dict*

Pass keyword arguments to the engine. This is currently only used by the numba engine, see the documentation for the engine argument for more information.

****kwargs**

Additional keyword arguments to pass as keywords arguments to *func*.

Returns:

Series or DataFrame

Result of applying `func` along the given axis of the DataFrame.

See also

`DataFrame.map`

For elementwise operations.

`DataFrame.aggregate`

Only perform aggregating type operations.

`DataFrame.transform`

Only perform transforming type operations.

Notes

Functions that mutate the passed object can produce unexpected behavior or errors and are not supported. See Mutating with User Defined Function (UDF) methods for more details.

Examples

```
>>> df = pd.DataFrame([[4, 9]] * 3, columns=['A', 'B'])
>>> df
   A  B
0  4  9
1  4  9
2  4  9
```

Using a numpy universal function (in this case the same as `np.sqrt(df)`):

```
>>> df.apply(np.sqrt)
     A    B
0  2.0  3.0
```

```
1  2.0  3.0
2  2.0  3.0
```

Using a reducing function on either axis

```
>>> df.apply(np.sum, axis=0)
A    12
B    27
dtype: int64
```

```
>>> df.apply(np.sum, axis=1)
0    13
1    13
2    13
dtype: int64
```

Returning a list-like will result in a Series

```
>>> df.apply(lambda x: [1, 2], axis=1)
0    [1, 2]
1    [1, 2]
2    [1, 2]
dtype: object
```

Passing `result_type='expand'` will expand list-like results to columns of a Dataframe

```
>>> df.apply(lambda x: [1, 2], axis=1, result_type='expand')
   0  1
0  1  2
1  1  2
2  1  2
```

Returning a Series inside the function is similar to passing `result_type='expand'`. The resulting column names will be the Series index.

```
>>> df.apply(lambda x: pd.Series([1, 2], index=['foo', 'bar']), axis=1)
   foo  bar
0    1    2
1    1    2
2    1    2
```

Passing `result_type='broadcast'` will ensure the same shape result, whether list-like or scalar is returned by the function, and broadcast it along the axis. The resulting column names will be the originals.

```
>>> df.apply(lambda x: [1, 2], axis=1, result_type='broadcast')
   A  B
0  1  2
1  1  2
```