pandas.DataFrame.apply

`DataFrame.apply`(*func*, *axis=0*, *broadcast=None*, *raw=False*, *reduce=None*, *result_type=None*, *args=()*, ***kwds*) [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.

broadcast : *bool, optional*
Only relevant for aggregation functions:
- `False` or `None` : returns a Series whose length is the length of the index or the number of columns (based on the *axis* parameter)
- `True` : results will be broadcast to the original shape of the frame, the original index and columns will be retained.

> *Deprecated since version 0.23.0:* This argument will be removed in a future version, replaced by result_type='broadcast'.

raw : *bool, default False*
- `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.

reduce : *bool or None, default None*
Try to apply reduction procedures. If the DataFrame is empty, *apply* will use *reduce* to determine whether the result should be a Series or a DataFrame. If `reduce=None` (the default), *apply*'s return value will be guessed by calling *func* on an empty Series (note: while guessing, exceptions raised by *func* will be ignored). If `reduce=True` a Series will always be returned, and if `reduce=False` a DataFrame will always be returned.

> *Deprecated since version 0.23.0:* This argument will be removed in a future version, replaced by `result_type='reduce'`.

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’.
- ‘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.

New in version 0.23.0.

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

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

Returns: **applied** : *Series or DataFrame*

See also:

DataFrame.applymap
For elementwise operations.

DataFrame.aggregate
Only perform aggregating type operations.

DataFrame.transform
Only perform transforming type operations.

Notes

In the current implementation apply calls *func* twice on the first column/row to decide whether it can take a fast or slow code path. This can lead to unexpected behavior if *func* has side-effects, as they will take effect twice for the first column/row.

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
```

Retuning 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
2  1  2
```