pandas.Series.interpolate

`Series.interpolate`(*self*, *method='linear'*, *axis=0*, *limit=None*, *inplace=False*, *limit_direction='forward'*, *limit_area=None*, *downcast=None*, ***kwargs*) [source]

Interpolate values according to different methods.

Please note that only `method='linear'` is supported for DataFrame/Series with a MultiIndex.

Parameters:

method : *str, default 'linear'*

Interpolation technique to use. One of:

- 'linear': Ignore the index and treat the values as equally spaced. This is the only method supported on MultiIndexes.
- 'time': Works on daily and higher resolution data to interpolate given length of interval.
- 'index', 'values': use the actual numerical values of the index.
- 'pad': Fill in NaNs using existing values.
- 'nearest', 'zero', 'slinear', 'quadratic', 'cubic', 'spline', 'barycentric', 'polynomial': Passed to *scipy.interpolate.interp1d*. These methods use the numerical values of the index. Both 'polynomial' and 'spline' require that you also specify an *order* (int), e.g. `df.interpolate(method='polynomial', order=5)`.
- 'krogh', 'piecewise_polynomial', 'spline', 'pchip', 'akima': Wrappers around the SciPy interpolation methods of similar names. See *Notes*.
- 'from_derivatives': Refers to *scipy.interpolate.BPoly.from_derivatives* which replaces 'piecewise_polynomial' interpolation method in scipy 0.18.

axis : *{0 or 'index', 1 or 'columns', None}, default None*

Axis to interpolate along.

limit : *int, optional*

Maximum number of consecutive NaNs to fill. Must be greater than 0.

inplace : *bool, default False*

Update the data in place if possible.

limit_direction : *{'forward', 'backward', 'both'}, default 'forward'*

If limit is specified, consecutive NaNs will be filled in this direction.

limit_area : *{None, 'inside', 'outside'}, default None*

If limit is specified, consecutive NaNs will be filled with this restriction.

- `None`: No fill restriction.
- 'inside': Only fill NaNs surrounded by valid values (interpolate).
- 'outside': Only fill NaNs outside valid values (extrapolate).

New in version 0.23.0.

downcast : *optional, 'infer' or None, defaults to None*

Downcast dtypes if possible.

****kwargs**

Keyword arguments to pass on to the interpolating function.

Returns: **Series or DataFrame**

Returns the same object type as the caller, interpolated at some or all `NaN` values.

See also

`fillna`

Fill missing values using different methods.

`scipy.interpolate.Akima1DInterpolator`

Piecewise cubic polynomials (Akima interpolator).

`scipy.interpolate.BPoly.from_derivatives`

Piecewise polynomial in the Bernstein basis.

`scipy.interpolate.interp1d`

Interpolate a 1-D function.

`scipy.interpolate.KroghInterpolator`

Interpolate polynomial (Krogh interpolator).

`scipy.interpolate.PchipInterpolator`

PCHIP 1-d monotonic cubic interpolation.

`scipy.interpolate.CubicSpline`

Cubic spline data interpolator.

Notes

The ‘krogh’, ‘piecewise_polynomial’, ‘spline’, ‘pchip’ and ‘akima’ methods are wrappers around the respective SciPy implementations of similar names. These use the actual numerical values of the index. For more information on their behavior, see the SciPy documentation and SciPy tutorial.

Examples

Filling in `NaN` in a `Series` via linear interpolation.

```
>>> s = pd.Series([0, 1, np.nan, 3])
>>> s
0    0.0
1    1.0
2    NaN
3    3.0
dtype: float64
>>> s.interpolate()
0    0.0
1    1.0
2    2.0
3    3.0
dtype: float64
```

Filling in `NaN` in a Series by padding, but filling at most two consecutive `NaN` at a time.

```
>>> s = pd.Series([np.nan, "single_one", np.nan,
...                "fill_two_more", np.nan, np.nan, np.nan,
...                4.71, np.nan])
>>> s
0              NaN
1       single_one
2              NaN
3    fill_two_more
4              NaN
5              NaN
6              NaN
7             4.71
8              NaN
dtype: object
>>> s.interpolate(method='pad', limit=2)
0              NaN
1       single_one
2       single_one
3    fill_two_more
4    fill_two_more
5    fill_two_more
6              NaN
7             4.71
8             4.71
dtype: object
```

Filling in `NaN` in a Series via polynomial interpolation or splines: Both ‘polynomial’ and ‘spline’ methods require that you also specify an `order` (int).

```
>>> s = pd.Series([0, 2, np.nan, 8])
>>> s.interpolate(method='polynomial', order=2)
0    0.000000
1    2.000000
2    4.666667
3    8.000000
dtype: float64
```

Fill the DataFrame forward (that is, going down) along each column using linear interpolation.

Note how the last entry in column 'a' is interpolated differently, because there is no entry after it to use for interpolation. Note how the first entry in column 'b' remains `NaN`, because there is no entry before it to use for interpolation.

```
>>> df = pd.DataFrame([(0.0, np.nan, -1.0, 1.0),
...                    (np.nan, 2.0, np.nan, np.nan),
...                    (2.0, 3.0, np.nan, 9.0),
...                    (np.nan, 4.0, -4.0, 16.0)],
...                   columns=list('abcd'))
>>> df
     a    b    c     d
0  0.0  NaN -1.0   1.0
1  NaN  2.0  NaN   NaN
2  2.0  3.0  NaN   9.0
3  NaN  4.0 -4.0  16.0
>>> df.interpolate(method='linear', limit_direction='forward', axis=0)
     a    b    c     d
0  0.0  NaN -1.0   1.0
1  1.0  2.0 -2.0   5.0
2  2.0  3.0 -3.0   9.0
3  2.0  4.0 -4.0  16.0
```

Using polynomial interpolation.

```
>>> df['d'].interpolate(method='polynomial', order=2)
0     1.0
1     4.0
2     9.0
3    16.0
Name: d, dtype: float64
```

<< pandas.Series.infer_objects

pandas.Series.isin >>