```
In [42]: s.rolling(2, min_periods=1).apply(lambda x: x.iloc[-1], raw=False)
Out[42]:
1    0.0
2    1.0
3    2.0
4    3.0
5    4.0
Length: 5, dtype: float64
```

Mimic the original behavior of passing a ndarray:

```
In [43]: s.rolling(2, min_periods=1).apply(lambda x: x[-1], raw=True)
Out[43]:
1    0.0
2    1.0
3    2.0
4    3.0
5    4.0
Length: 5, dtype: float64
```

`DataFrame.interpolate` has gained the `limit_area` kwarg

`DataFrame.interpolate()` has gained a `limit_area` parameter to allow further control of which `NaN` s are replaced. Use `limit_area='inside'` to fill only NaNs surrounded by valid values or use `limit_area='outside'` to fill only `NaN` s outside the existing valid values while preserving those inside. (GH16284) See the *full documentation here*.

```
In [44]: ser = pd.Series([np.nan, np.nan, 5, np.nan, np.nan,
   ....:                  np.nan, 13, np.nan, np.nan])
   ....:

In [45]: ser
Out[45]:
0     NaN
1     NaN
2     5.0
3     NaN
4     NaN
5     NaN
6    13.0
7     NaN
8     NaN
Length: 9, dtype: float64
```

Fill one consecutive inside value in both directions

```
In [46]: ser.interpolate(limit_direction='both', limit_area='inside', limit=1)
Out[46]:
0     NaN
1     NaN
2     5.0
3     7.0
4     NaN
5    11.0
6    13.0
```

(continues on next page)

(continued from previous page)

```
7     NaN
8     NaN
Length: 9, dtype: float64
```

Fill all consecutive outside values backward

```
In [47]: ser.interpolate(limit_direction='backward', limit_area='outside')
Out[47]:
0     5.0
1     5.0
2     5.0
3     NaN
4     NaN
5     NaN
6    13.0
7     NaN
8     NaN
Length: 9, dtype: float64
```

Fill all consecutive outside values in both directions

```
In [48]: ser.interpolate(limit_direction='both', limit_area='outside')
Out[48]:
0     5.0
1     5.0
2     5.0
3     NaN
4     NaN
5     NaN
6    13.0
7    13.0
8    13.0
Length: 9, dtype: float64
```

`get_dummies` now supports `dtype` argument

The *`get_dummies()`* now accepts a `dtype` argument, which specifies a dtype for the new columns. The default remains uint8. (GH18330)

```
In [49]: df = pd.DataFrame({'a': [1, 2], 'b': [3, 4], 'c': [5, 6]})

In [50]: pd.get_dummies(df, columns=['c']).dtypes
Out[50]:
a      int64
b      int64
c_5    uint8
c_6    uint8
Length: 4, dtype: object

In [51]: pd.get_dummies(df, columns=['c'], dtype=bool).dtypes
Out[51]:
a      int64
b      int64
c_5     bool
```

(continues on next page)

(continued from previous page)

```
c_6     bool
Length: 4, dtype: object
```

Timedelta mod method

`mod` (%) and `divmod` operations are now defined on `Timedelta` objects when operating with either timedelta-like or with numeric arguments. See the *documentation here*. (GH19365)

```
In [52]: td = pd.Timedelta(hours=37)

In [53]: td % pd.Timedelta(minutes=45)
Out[53]: Timedelta('0 days 00:15:00')
```

`.rank()` handles `inf` values when `NaN` are present

In previous versions, `.rank()` would assign `inf` elements `NaN` as their ranks. Now ranks are calculated properly. (GH6945)

```
In [54]: s = pd.Series([-np.inf, 0, 1, np.nan, np.inf])

In [55]: s
Out[55]:
0   -inf
1    0.0
2    1.0
3    NaN
4    inf
Length: 5, dtype: float64
```

Previous behavior:

```
In [11]: s.rank()
Out[11]:
0    1.0
1    2.0
2    3.0
3    NaN
4    NaN
dtype: float64
```

Current behavior:

```
In [56]: s.rank()
Out[56]:
0    1.0
1    2.0
2    3.0
3    NaN
4    4.0
Length: 5, dtype: float64
```

Furthermore, previously if you rank `inf` or `-inf` values together with `NaN` values, the calculation won't distinguish `NaN` from infinity when using 'top' or 'bottom' argument.

```
In [57]: s = pd.Series([np.nan, np.nan, -np.inf, -np.inf])

In [58]: s
Out[58]:
0    NaN
1    NaN
2   -inf
3   -inf
Length: 4, dtype: float64
```

Previous behavior:

```
In [15]: s.rank(na_option='top')
Out[15]:
0    2.5
1    2.5
2    2.5
3    2.5
dtype: float64
```

Current behavior:

```
In [59]: s.rank(na_option='top')
Out[59]:
0    1.5
1    1.5
2    3.5
3    3.5
Length: 4, dtype: float64
```

These bugs were squashed:

- Bug in *DataFrame.rank()* and *Series.rank()* when `method='dense'` and `pct=True` in which percentile ranks were not being used with the number of distinct observations (GH15630)
- Bug in *Series.rank()* and *DataFrame.rank()* when `ascending='False'` failed to return correct ranks for infinity if `NaN` were present (GH19538)
- Bug in `DataFrameGroupBy.rank()` where ranks were incorrect when both infinity and `NaN` were present (GH20561)

`Series.str.cat` has gained the `join` kwarg

Previously, *Series.str.cat()* did not – in contrast to most of `pandas` – align *Series* on their index before concatenation (see GH18657). The method has now gained a keyword `join` to control the manner of alignment, see examples below and *here*.

In v.0.23 *join* will default to None (meaning no alignment), but this default will change to `'left'` in a future version of pandas.

```
In [60]: s = pd.Series(['a', 'b', 'c', 'd'])

In [61]: t = pd.Series(['b', 'd', 'e', 'c'], index=[1, 3, 4, 2])

In [62]: s.str.cat(t)
Out[62]:
0    NaN
```

(continues on next page)

(continued from previous page)

```
1     bb
2     cc
3     dd
Length: 4, dtype: object

In [63]: s.str.cat(t, join='left', na_rep='-')
Out[63]:
0    a-
1    bb
2    cc
3    dd
Length: 4, dtype: object
```

Furthermore, *Series.str.cat()* now works for `CategoricalIndex` as well (previously raised a `ValueError`; see GH20842).

`DataFrame.astype` performs column-wise conversion to `Categorical`

DataFrame.astype() can now perform column-wise conversion to `Categorical` by supplying the string `'category'` or a `CategoricalDtype`. Previously, attempting this would raise a `NotImplementedError`. See the *Object creation* section of the documentation for more details and examples. (GH12860, GH18099)

Supplying the string `'category'` performs column-wise conversion, with only labels appearing in a given column set as categories:

```
In [64]: df = pd.DataFrame({'A': list('abca'), 'B': list('bccd')})

In [65]: df = df.astype('category')

In [66]: df['A'].dtype
Out[66]: CategoricalDtype(categories=['a', 'b', 'c'], ordered=False)

In [67]: df['B'].dtype
Out[67]: CategoricalDtype(categories=['b', 'c', 'd'], ordered=False)
```

Supplying a `CategoricalDtype` will make the categories in each column consistent with the supplied dtype:

```
In [68]: from pandas.api.types import CategoricalDtype

In [69]: df = pd.DataFrame({'A': list('abca'), 'B': list('bccd')})

In [70]: cdt = CategoricalDtype(categories=list('abcd'), ordered=True)

In [71]: df = df.astype(cdt)

In [72]: df['A'].dtype
Out[72]: CategoricalDtype(categories=['a', 'b', 'c', 'd'], ordered=True)

In [73]: df['B'].dtype
Out[73]: CategoricalDtype(categories=['a', 'b', 'c', 'd'], ordered=True)
```

Other enhancements

- Unary + now permitted for `Series` and `DataFrame` as numeric operator (GH16073)
- Better support for *`to_excel()`* output with the `xlsxwriter` engine. (GH16149)
- *`pandas.tseries.frequencies.to_offset()`* now accepts leading '+' signs e.g. '+1h'. (GH18171)
- `MultiIndex.unique()` now supports the `level=` argument, to get unique values from a specific index level (GH17896)
- *`pandas.io.formats.style.Styler`* now has method `hide_index()` to determine whether the index will be rendered in output (GH14194)
- *`pandas.io.formats.style.Styler`* now has method `hide_columns()` to determine whether columns will be hidden in output (GH14194)
- Improved wording of `ValueError` raised in *`to_datetime()`* when `unit=` is passed with a non-convertible value (GH14350)
- *`Series.fillna()`* now accepts a Series or a dict as a `value` for a categorical dtype (GH17033)
- *`pandas.read_clipboard()`* updated to use qtpy, falling back to PyQt5 and then PyQt4, adding compatibility with Python3 and multiple python-qt bindings (GH17722)
- Improved wording of `ValueError` raised in *`read_csv()`* when the `usecols` argument cannot match all columns. (GH17301)
- *`DataFrame.corrwith()`* now silently drops non-numeric columns when passed a Series. Before, an exception was raised (GH18570).
- *`IntervalIndex`* now supports time zone aware `Interval` objects (GH18537, GH18538)
- *`Series()`* / *`DataFrame()`* tab completion also returns identifiers in the first level of a *`MultiIndex()`*. (GH16326)
- *`read_excel()`* has gained the `nrows` parameter (GH16645)
- *`DataFrame.append()`* can now in more cases preserve the type of the calling dataframe's columns (e.g. if both are `CategoricalIndex`) (GH18359)
- *`DataFrame.to_json()`* and *`Series.to_json()`* now accept an `index` argument which allows the user to exclude the index from the JSON output (GH17394)
- `IntervalIndex.to_tuples()` has gained the `na_tuple` parameter to control whether NA is returned as a tuple of NA, or NA itself (GH18756)
- `Categorical.rename_categories`, `CategoricalIndex.rename_categories` and *`Series.cat.rename_categories`* can now take a callable as their argument (GH18862)
- *`Interval`* and *`IntervalIndex`* have gained a `length` attribute (GH18789)
- `Resampler` objects now have a functioning *`pipe`* method. Previously, calls to `pipe` were diverted to the `mean` method (GH17905).
- *`is_scalar()`* now returns `True` for `DateOffset` objects (GH18943).
- *`DataFrame.pivot()`* now accepts a list for the `values=` kwarg (GH17160).
- Added *`pandas.api.extensions.register_dataframe_accessor()`*, *`pandas.api.extensions.register_series_accessor()`*, and *`pandas.api.extensions.register_index_accessor()`*, accessor for libraries downstream of pandas to register custom accessors like `.cat` on pandas objects. See *Registering Custom Accessors* for more (GH14781).

- `IntervalIndex.astype` now supports conversions between subtypes when passed an `IntervalDtype` (GH19197)
- *`IntervalIndex`* and its associated constructor methods (`from_arrays`, `from_breaks`, `from_tuples`) have gained a `dtype` parameter (GH19262)
- Added *`pandas.core.groupby.SeriesGroupBy.is_monotonic_increasing()`* and *`pandas.core.groupby.SeriesGroupBy.is_monotonic_decreasing()`* (GH17015)
- For subclassed `DataFrames`, *`DataFrame.apply()`* will now preserve the `Series` subclass (if defined) when passing the data to the applied function (GH19822)
- *`DataFrame.from_dict()`* now accepts a `columns` argument that can be used to specify the column names when `orient='index'` is used (GH18529)
- Added option `display.html.use_mathjax` so MathJax can be disabled when rendering tables in `Jupyter` notebooks (GH19856, GH19824)
- *`DataFrame.replace()`* now supports the `method` parameter, which can be used to specify the replacement method when `to_replace` is a scalar, list or tuple and `value` is `None` (GH19632)
- *`Timestamp.month_name()`*, *`DatetimeIndex.month_name()`*, and *`Series.dt.month_name()`* are now available (GH12805)
- *`Timestamp.day_name()`* and *`DatetimeIndex.day_name()`* are now available to return day names with a specified locale (GH12806)
- *`DataFrame.to_sql()`* now performs a multi-value insert if the underlying connection supports itk rather than inserting row by row. `SQLAlchemy` dialects supporting multi-value inserts include: `mysql`, `postgresql`, `sqlite` and any dialect with `supports_multivalues_insert`. (GH14315, GH8953)
- *`read_html()`* now accepts a `displayed_only` keyword argument to controls whether or not hidden elements are parsed (`True` by default) (GH20027)
- *`read_html()`* now reads all `<tbody>` elements in a `<table>`, not just the first. (GH20690)
- `quantile()` and `quantile()` now accept the `interpolation` keyword, `linear` by default (GH20497)
- zip compression is supported via `compression=zip` in *`DataFrame.to_pickle()`*, *`Series.to_pickle()`*, *`DataFrame.to_csv()`*, *`Series.to_csv()`*, *`DataFrame.to_json()`*, *`Series.to_json()`*. (GH17778)
- *`WeekOfMonth`* constructor now supports `n=0` (GH20517).
- *`DataFrame`* and *`Series`* now support matrix multiplication (`@`) operator (GH10259) for Python>=3.5
- Updated *`DataFrame.to_gbq()`* and *`pandas.read_gbq()`* signature and documentation to reflect changes from the Pandas-GBQ library version 0.4.0. Adds intersphinx mapping to Pandas-GBQ library. (GH20564)
- Added new writer for exporting Stata dta files in version 117, `StataWriter117`. This format supports exporting strings with lengths up to 2,000,000 characters (GH16450)
- `to_hdf()` and *`read_hdf()`* now accept an `errors` keyword argument to control encoding error handling (GH20835)
- *`cut()`* has gained the `duplicates='raise'|'drop'` option to control whether to raise on duplicated edges (GH20947)
- *`date_range()`*, *`timedelta_range()`*, and *`interval_range()`* now return a linearly spaced index if `start`, `stop`, and `periods` are specified, but `freq` is not. (GH20808, GH20983, GH20976)

Backwards incompatible API changes

Dependencies have increased minimum versions

We have updated our minimum supported versions of dependencies (GH15184). If installed, we now require:

Package	Minimum Version	Required	Issue
python-dateutil	2.5.0	X	GH15184
openpyxl	2.4.0		GH15184
beautifulsoup4	4.2.1		GH20082
setuptools	24.2.0		GH20698

Instantiation from dicts preserves dict insertion order for python 3.6+

Until Python 3.6, dicts in Python had no formally defined ordering. For Python version 3.6 and later, dicts are ordered by insertion order, see PEP 468. Pandas will use the dict's insertion order, when creating a `Series` or `DataFrame` from a dict and you're using Python version 3.6 or higher. (GH19884)

Previous behavior (and current behavior if on Python < 3.6):

```
In [16]: pd.Series({'Income': 2000,
   ....:            'Expenses': -1500,
   ....:            'Taxes': -200,
   ....:            'Net result': 300})
Out[16]:
Expenses     -1500
Income        2000
Net result     300
Taxes         -200
dtype: int64
```

Note the Series above is ordered alphabetically by the index values.

New behavior (for Python >= 3.6):

```
In [74]: pd.Series({'Income': 2000,
   ....:            'Expenses': -1500,
   ....:            'Taxes': -200,
   ....:            'Net result': 300})
   ....:
Out[74]:
Income        2000
Expenses     -1500
Taxes         -200
Net result     300
Length: 4, dtype: int64
```

Notice that the Series is now ordered by insertion order. This new behavior is used for all relevant pandas types (`Series`, `DataFrame`, `SparseSeries` and `SparseDataFrame`).

If you wish to retain the old behavior while using Python >= 3.6, you can use `.sort_index()`:

```
In [75]: pd.Series({'Income': 2000,
   ....:            'Expenses': -1500,
   ....:            'Taxes': -200,
```

(continues on next page)

(continued from previous page)

```
   ....:             'Net result': 300}).sort_index()
   ....:
Out[75]:
Expenses     -1500
Income        2000
Net result     300
Taxes         -200
Length: 4, dtype: int64
```

Deprecate Panel

`Panel` was deprecated in the 0.20.x release, showing as a `DeprecationWarning`. Using `Panel` will now show a `FutureWarning`. The recommended way to represent 3-D data are with a `MultiIndex` on a `DataFrame` via the `to_frame()` or with the xarray package. Pandas provides a `to_xarray()` method to automate this conversion (GH13563, GH18324).

```
In [75]: import pandas._testing as tm

In [76]: p = tm.makePanel()

In [77]: p
Out[77]:
<class 'pandas.core.panel.Panel'>
Dimensions: 3 (items) x 3 (major_axis) x 4 (minor_axis)
Items axis: ItemA to ItemC
Major_axis axis: 2000-01-03 00:00:00 to 2000-01-05 00:00:00
Minor_axis axis: A to D
```

Convert to a MultiIndex DataFrame

```
In [78]: p.to_frame()
Out[78]:
                     ItemA     ItemB     ItemC
major      minor
2000-01-03 A      0.469112  0.721555  0.404705
           B     -1.135632  0.271860 -1.039268
           C      0.119209  0.276232 -1.344312
           D     -2.104569  0.113648 -0.109050
2000-01-04 A     -0.282863 -0.706771  0.577046
           B      1.212112 -0.424972 -0.370647
           C     -1.044236 -1.087401  0.844885
           D     -0.494929 -1.478427  1.643563
2000-01-05 A     -1.509059 -1.039575 -1.715002
           B     -0.173215  0.567020 -1.157892
           C     -0.861849 -0.673690  1.075770
           D      1.071804  0.524988 -1.469388

[12 rows x 3 columns]
```

Convert to an xarray DataArray

```
In [79]: p.to_xarray()
Out[79]:
<xarray.DataArray (items: 3, major_axis: 3, minor_axis: 4)>
```

(continues on next page)

(continued from previous page)

```
array([[[ 0.469112, -1.135632,  0.119209, -2.104569],
        [-0.282863,  1.212112, -1.044236, -0.494929],
        [-1.509059, -0.173215, -0.861849,  1.071804]],

       [[ 0.721555,  0.27186 ,  0.276232,  0.113648],
        [-0.706771, -0.424972, -1.087401, -1.478427],
        [-1.039575,  0.56702 , -0.67369 ,  0.524988]],

       [[ 0.404705, -1.039268, -1.344312, -0.10905 ],
        [ 0.577046, -0.370647,  0.844885,  1.643563],
        [-1.715002, -1.157892,  1.07577 , -1.469388]]])
Coordinates:
  * items       (items) object 'ItemA' 'ItemB' 'ItemC'
  * major_axis  (major_axis) datetime64[ns] 2000-01-03 2000-01-04 2000-01-05
  * minor_axis  (minor_axis) object 'A' 'B' 'C' 'D'
```

pandas.core.common removals

The following error & warning messages are removed from `pandas.core.common` (GH13634, GH19769):

- `PerformanceWarning`
- `UnsupportedFunctionCall`
- `UnsortedIndexError`
- `AbstractMethodError`

These are available from import from `pandas.errors` (since 0.19.0).

Changes to make output of `DataFrame.apply` consistent

`DataFrame.apply()` was inconsistent when applying an arbitrary user-defined-function that returned a list-like with `axis=1`. Several bugs and inconsistencies are resolved. If the applied function returns a Series, then pandas will return a DataFrame; otherwise a Series will be returned, this includes the case where a list-like (e.g. `tuple` or `list` is returned) (GH16353, GH17437, GH17970, GH17348, GH17892, GH18573, GH17602, GH18775, GH18901, GH18919).

```
In [76]: df = pd.DataFrame(np.tile(np.arange(3), 6).reshape(6, -1) + 1,
   ....:                   columns=['A', 'B', 'C'])
   ....:

In [77]: df
Out[77]:
   A  B  C
0  1  2  3
1  1  2  3
2  1  2  3
3  1  2  3
4  1  2  3
5  1  2  3

[6 rows x 3 columns]
```

Previous behavior: if the returned shape happened to match the length of original columns, this would return a `DataFrame`. If the return shape did not match, a `Series` with lists was returned.

```
In [3]: df.apply(lambda x: [1, 2, 3], axis=1)
Out[3]:
   A  B  C
0  1  2  3
1  1  2  3
2  1  2  3
3  1  2  3
4  1  2  3
5  1  2  3

In [4]: df.apply(lambda x: [1, 2], axis=1)
Out[4]:
0    [1, 2]
1    [1, 2]
2    [1, 2]
3    [1, 2]
4    [1, 2]
5    [1, 2]
dtype: object
```

New behavior: When the applied function returns a list-like, this will now *always* return a `Series`.

```
In [78]: df.apply(lambda x: [1, 2, 3], axis=1)
Out[78]:
0    [1, 2, 3]
1    [1, 2, 3]
2    [1, 2, 3]
3    [1, 2, 3]
4    [1, 2, 3]
5    [1, 2, 3]
Length: 6, dtype: object

In [79]: df.apply(lambda x: [1, 2], axis=1)
Out[79]:
0    [1, 2]
1    [1, 2]
2    [1, 2]
3    [1, 2]
4    [1, 2]
5    [1, 2]
Length: 6, dtype: object
```

To have expanded columns, you can use `result_type='expand'`

```
In [80]: df.apply(lambda x: [1, 2, 3], axis=1, result_type='expand')
Out[80]:
   0  1  2
0  1  2  3
1  1  2  3
2  1  2  3
3  1  2  3
4  1  2  3
5  1  2  3

[6 rows x 3 columns]
```

To broadcast the result across the original columns (the old behaviour for list-likes of the correct length), you can use `result_type='broadcast'`. The shape must match the original columns.

```
In [81]: df.apply(lambda x: [1, 2, 3], axis=1, result_type='broadcast')
Out[81]:
   A  B  C
0  1  2  3
1  1  2  3
2  1  2  3
3  1  2  3
4  1  2  3
5  1  2  3

[6 rows x 3 columns]
```

Returning a `Series` allows one to control the exact return structure and column names:

```
In [82]: df.apply(lambda x: pd.Series([1, 2, 3], index=['D', 'E', 'F']), axis=1)
Out[82]:
   D  E  F
0  1  2  3
1  1  2  3
2  1  2  3
3  1  2  3
4  1  2  3
5  1  2  3

[6 rows x 3 columns]
```

Concatenation will no longer sort

In a future version of pandas *`pandas.concat()`* will no longer sort the non-concatenation axis when it is not already aligned. The current behavior is the same as the previous (sorting), but now a warning is issued when `sort` is not specified and the non-concatenation axis is not aligned (GH4588).

```
In [83]: df1 = pd.DataFrame({"a": [1, 2], "b": [1, 2]}, columns=['b', 'a'])

In [84]: df2 = pd.DataFrame({"a": [4, 5]})

In [85]: pd.concat([df1, df2])
Out[85]:
     b  a
0  1.0  1
1  2.0  2
0  NaN  4
1  NaN  5

[4 rows x 2 columns]
```

To keep the previous behavior (sorting) and silence the warning, pass `sort=True`

```
In [86]: pd.concat([df1, df2], sort=True)
Out[86]:
   a    b
0  1  1.0
1  2  2.0
0  4  NaN
1  5  NaN
```

(continues on next page)

(continued from previous page)

```
[4 rows x 2 columns]
```

To accept the future behavior (no sorting), pass `sort=False`

Note that this change also applies to *`DataFrame.append()`*, which has also received a `sort` keyword for controlling this behavior.

Build changes

- Building pandas for development now requires `cython >= 0.24` (GH18613)
- Building from source now explicitly requires `setuptools` in `setup.py` (GH18113)
- Updated conda recipe to be in compliance with conda-build 3.0+ (GH18002)

Index division by zero fills correctly

Division operations on `Index` and subclasses will now fill division of positive numbers by zero with `np.inf`, division of negative numbers by zero with `-np.inf` and *0 / 0* with `np.nan`. This matches existing `Series` behavior. (GH19322, GH19347)

Previous behavior:

```
In [6]: index = pd.Int64Index([-1, 0, 1])

In [7]: index / 0
Out[7]: Int64Index([0, 0, 0], dtype='int64')

# Previous behavior yielded different results depending on the type of zero in the␣
↪divisor
In [8]: index / 0.0
Out[8]: Float64Index([-inf, nan, inf], dtype='float64')

In [9]: index = pd.UInt64Index([0, 1])

In [10]: index / np.array([0, 0], dtype=np.uint64)
Out[10]: UInt64Index([0, 0], dtype='uint64')

In [11]: pd.RangeIndex(1, 5) / 0
ZeroDivisionError: integer division or modulo by zero
```

Current behavior:

```
In [87]: index = pd.Int64Index([-1, 0, 1])

# division by zero gives -infinity where negative,
# +infinity where positive, and NaN for 0 / 0
In [88]: index / 0
Out[88]: Float64Index([-inf, nan, inf], dtype='float64')

# The result of division by zero should not depend on
# whether the zero is int or float
In [89]: index / 0.0
Out[89]: Float64Index([-inf, nan, inf], dtype='float64')
```

(continues on next page)

(continued from previous page)

```
In [90]: index = pd.UInt64Index([0, 1])

In [91]: index / np.array([0, 0], dtype=np.uint64)
Out[91]: Float64Index([nan, inf], dtype='float64')

In [92]: pd.RangeIndex(1, 5) / 0
Out[92]: Float64Index([inf, inf, inf, inf], dtype='float64')
```

Extraction of matching patterns from strings

By default, extracting matching patterns from strings with `str.extract()` used to return a `Series` if a single group was being extracted (a `DataFrame` if more than one group was extracted). As of Pandas 0.23.0 `str.extract()` always returns a `DataFrame`, unless `expand` is set to `False`. Finally, `None` was an accepted value for the `expand` parameter (which was equivalent to `False`), but now raises a `ValueError`. (GH11386)

Previous behavior:

```
In [1]: s = pd.Series(['number 10', '12 eggs'])

In [2]: extracted = s.str.extract(r'.*(\d\d).*')

In [3]: extracted
Out [3]:
0    10
1    12
dtype: object

In [4]: type(extracted)
Out [4]:
pandas.core.series.Series
```

New behavior:

```
In [93]: s = pd.Series(['number 10', '12 eggs'])

In [94]: extracted = s.str.extract(r'.*(\d\d).*')

In [95]: extracted
Out[95]:
    0
0  10
1  12

[2 rows x 1 columns]

In [96]: type(extracted)
Out[96]: pandas.core.frame.DataFrame
```

To restore previous behavior, simply set `expand` to `False`:

```
In [97]: s = pd.Series(['number 10', '12 eggs'])

In [98]: extracted = s.str.extract(r'.*(\d\d).*', expand=False)
```

(continues on next page)

(continued from previous page)

```
In [99]: extracted
Out[99]:
0    10
1    12
Length: 2, dtype: object

In [100]: type(extracted)
Out[100]: pandas.core.series.Series
```

Default value for the `ordered` parameter of `CategoricalDtype`

The default value of the `ordered` parameter for `CategoricalDtype` has changed from `False` to `None` to allow updating of `categories` without impacting `ordered`. Behavior should remain consistent for downstream objects, such as *`Categorical`* (GH18790)

In previous versions, the default value for the `ordered` parameter was `False`. This could potentially lead to the `ordered` parameter unintentionally being changed from `True` to `False` when users attempt to update `categories` if `ordered` is not explicitly specified, as it would silently default to `False`. The new behavior for `ordered=None` is to retain the existing value of `ordered`.

New behavior:

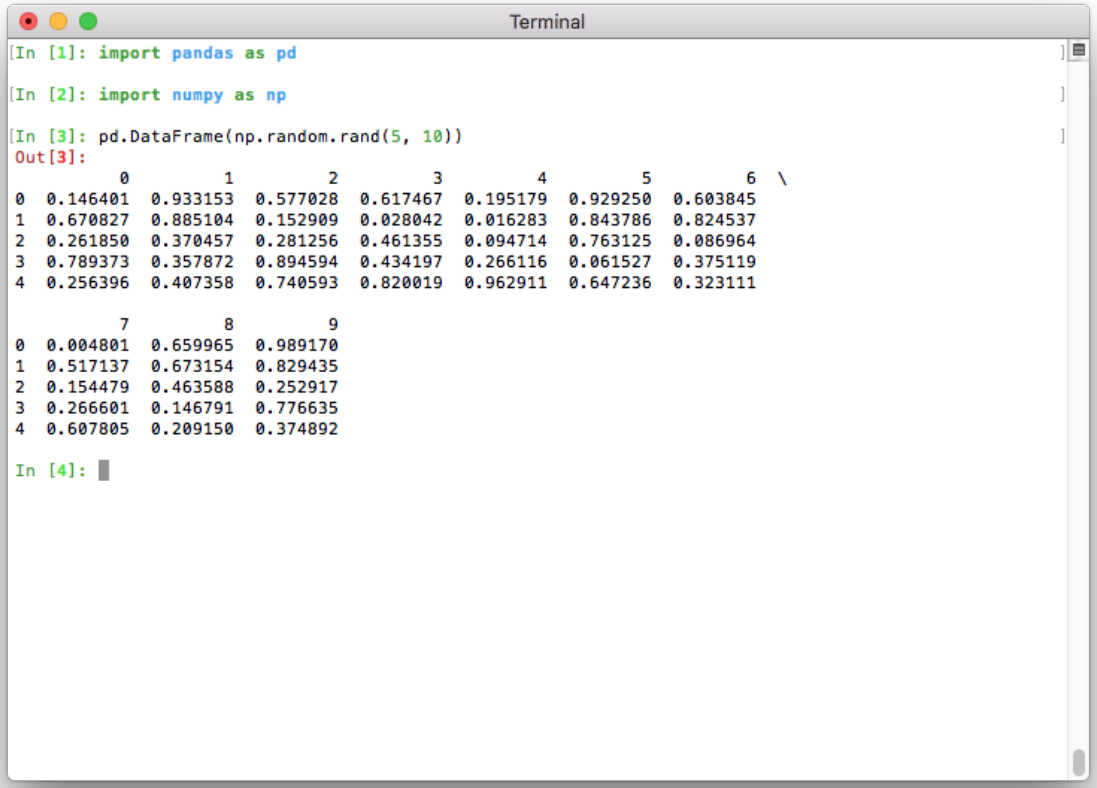
```
In [2]: from pandas.api.types import CategoricalDtype

In [3]: cat = pd.Categorical(list('abcaba'), ordered=True, categories=list('cba'))

In [4]: cat
Out[4]:
[a, b, c, a, b, a]
Categories (3, object): [c < b < a]

In [5]: cdt = CategoricalDtype(categories=list('cbad'))

In [6]: cat.astype(cdt)
Out[6]:
[a, b, c, a, b, a]
Categories (4, object): [c < b < a < d]
```

Notice in the example above that the converted `Categorical` has retained `ordered=True`. Had the default value for `ordered` remained as `False`, the converted `Categorical` would have become unordered, despite `ordered=False` never being explicitly specified. To change the value of `ordered`, explicitly pass it to the new dtype, e.g. `CategoricalDtype(categories=list('cbad'), ordered=False)`.

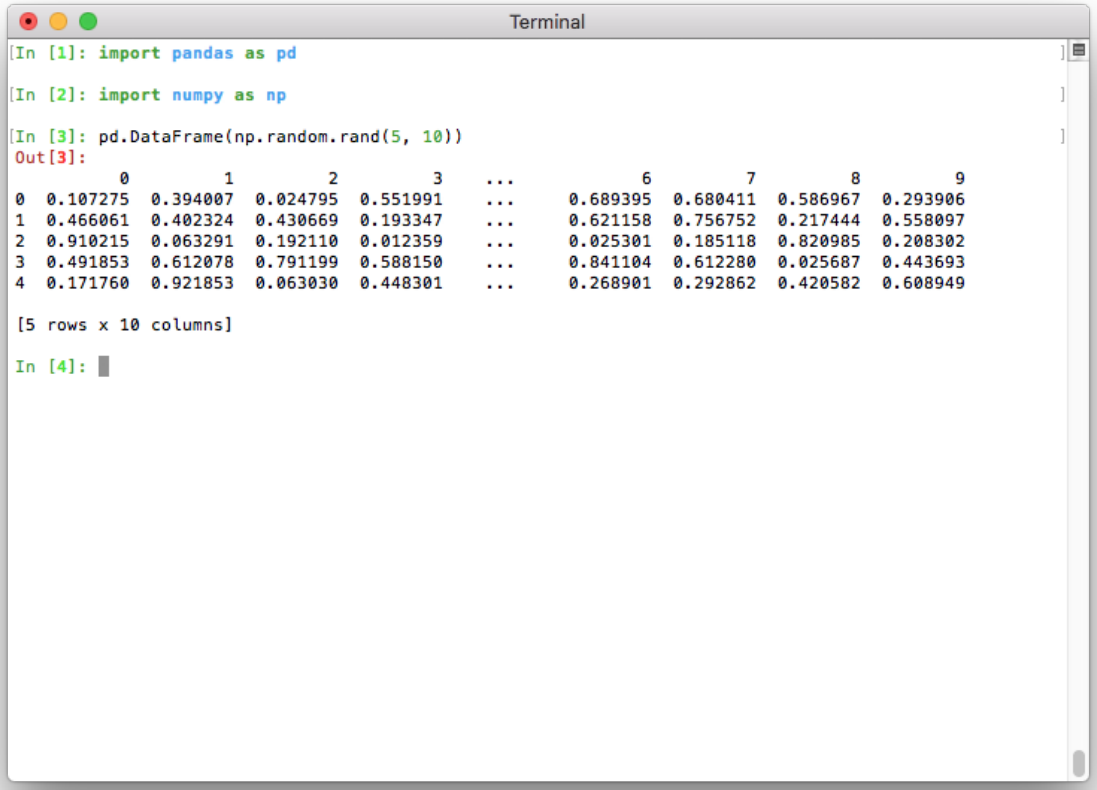
Note that the unintentional conversion of `ordered` discussed above did not arise in previous versions due to separate bugs that prevented `astype` from doing any type of category to category conversion (GH10696, GH18593). These bugs have been fixed in this release, and motivated changing the default value of `ordered`.

Better pretty-printing of DataFrames in a terminal

Previously, the default value for the maximum number of columns was `pd.options.display.max_columns=20`. This meant that relatively wide data frames would not fit within the terminal width, and pandas would introduce line breaks to display these 20 columns. This resulted in an output that was relatively difficult to read:



If Python runs in a terminal, the maximum number of columns is now determined automatically so that the printed data frame fits within the current terminal width (`pd.options.display.max_columns=0`) (GH17023). If Python runs as a Jupyter kernel (such as the Jupyter QtConsole or a Jupyter notebook, as well as in many IDEs), this value cannot be inferred automatically and is thus set to *20* as in previous versions. In a terminal, this results in a much nicer output:



Note that if you don't like the new default, you can always set this option yourself. To revert to the old setting, you can run this line:

```
pd.options.display.max_columns = 20
```

Datetimelike API changes

- The default `Timedelta` constructor now accepts an `ISO 8601 Duration` string as an argument (GH19040)
- Subtracting `NaT` from a *`Series`* with `dtype='datetime64[ns]'` returns a `Series` with `dtype='timedelta64[ns]'` instead of `dtype='datetime64[ns]'` (GH18808)
- Addition or subtraction of `NaT` from *`TimedeltaIndex`* will return `TimedeltaIndex` instead of `DatetimeIndex` (GH19124)
- `DatetimeIndex.shift()` and `TimedeltaIndex.shift()` will now raise `NullFrequencyError` (which subclasses `ValueError`, which was raised in older versions) when the index object frequency is `None` (GH19147)
- Addition and subtraction of `NaN` from a *`Series`* with `dtype='timedelta64[ns]'` will raise a `TypeError` instead of treating the `NaN` as `NaT` (GH19274)
- `NaT` division with `datetime.timedelta` will now return `NaN` instead of raising (GH17876)

- Operations between a *Series* with dtype `dtype='datetime64[ns]'` and a *PeriodIndex* will correctly raises `TypeError` (GH18850)
- Subtraction of *Series* with timezone-aware `dtype='datetime64[ns]'` with mis-matched timezones will raise `TypeError` instead of `ValueError` (GH18817)
- *Timestamp* will no longer silently ignore unused or invalid `tz` or `tzinfo` keyword arguments (GH17690)
- *Timestamp* will no longer silently ignore invalid `freq` arguments (GH5168)
- `CacheableOffset` and `WeekDay` are no longer available in the `pandas.tseries.offsets` module (GH17830)
- `pandas.tseries.frequencies.get_freq_group()` and `pandas.tseries.frequencies.DAYS` are removed from the public API (GH18034)
- *Series.truncate()* and *DataFrame.truncate()* will raise a `ValueError` if the index is not sorted instead of an unhelpful `KeyError` (GH17935)
- *Series.first* and *DataFrame.first* will now raise a `TypeError` rather than `NotImplementedError` when index is not a *DatetimeIndex* (GH20725).
- *Series.last* and *DataFrame.last* will now raise a `TypeError` rather than `NotImplementedError` when index is not a *DatetimeIndex* (GH20725).
- Restricted `DateOffset` keyword arguments. Previously, `DateOffset` subclasses allowed arbitrary keyword arguments which could lead to unexpected behavior. Now, only valid arguments will be accepted. (GH17176, GH18226).
- *pandas.merge()* provides a more informative error message when trying to merge on timezone-aware and timezone-naive columns (GH15800)
- For *DatetimeIndex* and *TimedeltaIndex* with `freq=None`, addition or subtraction of integer-dtyped array or `Index` will raise `NullFrequencyError` instead of `TypeError` (GH19895)
- *Timestamp* constructor now accepts a *nanosecond* keyword or positional argument (GH18898)
- *DatetimeIndex* will now raise an `AttributeError` when the `tz` attribute is set after instantiation (GH3746)
- *DatetimeIndex* with a `pytz` timezone will now return a consistent `pytz` timezone (GH18595)

Other API changes

- *Series.astype()* and *Index.astype()* with an incompatible dtype will now raise a `TypeError` rather than a `ValueError` (GH18231)
- `Series` construction with an `object` dtyped tz-aware datetime and `dtype=object` specified, will now return an `object` dtyped `Series`, previously this would infer the datetime dtype (GH18231)
- A *Series* of `dtype=category` constructed from an empty `dict` will now have categories of `dtype=object` rather than `dtype=float64`, consistently with the case in which an empty list is passed (GH18515)
- All-NaN levels in a `MultiIndex` are now assigned `float` rather than `object` dtype, promoting consistency with `Index` (GH17929).
- Levels names of a `MultiIndex` (when not None) are now required to be unique: trying to create a `MultiIndex` with repeated names will raise a `ValueError` (GH18872)
- Both construction and renaming of `Index`/`MultiIndex` with non-hashable `name`/`names` will now raise `TypeError` (GH20527)

- *Index.map()* can now accept Series and dictionary input objects (GH12756, GH18482, GH18509).
- *DataFrame.unstack()* will now default to filling with np.nan for object columns. (GH12815)
- *IntervalIndex* constructor will raise if the closed parameter conflicts with how the input data is inferred to be closed (GH18421)
- Inserting missing values into indexes will work for all types of indexes and automatically insert the correct type of missing value (NaN, NaT, etc.) regardless of the type passed in (GH18295)
- When created with duplicate labels, MultiIndex now raises a ValueError. (GH17464)
- *Series.fillna()* now raises a TypeError instead of a ValueError when passed a list, tuple or DataFrame as a value (GH18293)
- *pandas.DataFrame.merge()* no longer casts a float column to object when merging on int and float columns (GH16572)
- *pandas.merge()* now raises a ValueError when trying to merge on incompatible data types (GH9780)
- The default NA value for *UInt64Index* has changed from 0 to NaN, which impacts methods that mask with NA, such as UInt64Index.where() (GH18398)
- Refactored setup.py to use find_packages instead of explicitly listing out all subpackages (GH18535)
- Rearranged the order of keyword arguments in *read_excel()* to align with *read_csv()* (GH16672)
- *wide_to_long()* previously kept numeric-like suffixes as object dtype. Now they are cast to numeric if possible (GH17627)
- In *read_excel()*, the comment argument is now exposed as a named parameter (GH18735)
- Rearranged the order of keyword arguments in *read_excel()* to align with *read_csv()* (GH16672)
- The options html.border and mode.use_inf_as_null were deprecated in prior versions, these will now show FutureWarning rather than a DeprecationWarning (GH19003)
- *IntervalIndex* and IntervalDtype no longer support categorical, object, and string subtypes (GH19016)
- IntervalDtype now returns True when compared against 'interval' regardless of subtype, and IntervalDtype.name now returns 'interval' regardless of subtype (GH18980)
- KeyError now raises instead of ValueError in *drop()*, drop(), *drop()*, *drop()* when dropping a non-existent element in an axis with duplicates (GH19186)
- *Series.to_csv()* now accepts a compression argument that works in the same way as the compression argument in *DataFrame.to_csv()* (GH18958)
- Set operations (union, difference...) on *IntervalIndex* with incompatible index types will now raise a TypeError rather than a ValueError (GH19329)
- DateOffset objects render more simply, e.g. <DateOffset: days=1> instead of <DateOffset: kwds={'days': 1}> (GH19403)
- Categorical.fillna now validates its value and method keyword arguments. It now raises when both or none are specified, matching the behavior of *Series.fillna()* (GH19682)
- pd.to_datetime('today') now returns a datetime, consistent with pd.Timestamp('today'); previously pd.to_datetime('today') returned a .normalized() datetime (GH19935)
- *Series.str.replace()* now takes an optional *regex* keyword which, when set to False, uses literal string replacement rather than regex replacement (GH16808)
- *DatetimeIndex.strftime()* and *PeriodIndex.strftime()* now return an Index instead of a numpy array to be consistent with similar accessors (GH20127)

- Constructing a Series from a list of length 1 no longer broadcasts this list when a longer index is specified (GH19714, GH20391).
- *DataFrame.to_dict()* with `orient='index'` no longer casts int columns to float for a DataFrame with only int and float columns (GH18580)
- A user-defined-function that is passed to `Series.rolling().aggregate()`, `DataFrame.rolling().aggregate()`, or its expanding cousins, will now *always* be passed a `Series`, rather than a `np.array`; `.apply()` only has the `raw` keyword, see *here*. This is consistent with the signatures of `.aggregate()` across pandas (GH20584)
- Rolling and Expanding types raise `NotImplementedError` upon iteration (GH11704).

Deprecations

- `Series.from_array` and `SparseSeries.from_array` are deprecated. Use the normal constructor `Series(..)` and `SparseSeries(..)` instead (GH18213).
- `DataFrame.as_matrix` is deprecated. Use `DataFrame.values` instead (GH18458).
- `Series.asobject`, `DatetimeIndex.asobject`, `PeriodIndex.asobject` and `TimeDeltaIndex.asobject` have been deprecated. Use `.astype(object)` instead (GH18572)
- Grouping by a tuple of keys now emits a `FutureWarning` and is deprecated. In the future, a tuple passed to `'by'` will always refer to a single key that is the actual tuple, instead of treating the tuple as multiple keys. To retain the previous behavior, use a list instead of a tuple (GH18314)
- `Series.valid` is deprecated. Use *Series.dropna()* instead (GH18800).
- *read_excel()* has deprecated the `skip_footer` parameter. Use `skipfooter` instead (GH18836)
- *ExcelFile.parse()* has deprecated `sheetname` in favor of `sheet_name` for consistency with *read_excel()* (GH20920).
- The `is_copy` attribute is deprecated and will be removed in a future version (GH18801).
- `IntervalIndex.from_intervals` is deprecated in favor of the *IntervalIndex* constructor (GH19263)
- `DataFrame.from_items` is deprecated. Use *DataFrame.from_dict()* instead, or `DataFrame.from_dict(OrderedDict())` if you wish to preserve the key order (GH17320, GH17312)
- Indexing a *MultiIndex* or a `FloatIndex` with a list containing some missing keys will now show a `FutureWarning`, which is consistent with other types of indexes (GH17758).
- The `broadcast` parameter of `.apply()` is deprecated in favor of `result_type='broadcast'` (GH18577)
- The `reduce` parameter of `.apply()` is deprecated in favor of `result_type='reduce'` (GH18577)
- The `order` parameter of *factorize()* is deprecated and will be removed in a future release (GH19727)
- `Timestamp.weekday_name`, `DatetimeIndex.weekday_name`, and `Series.dt.weekday_name` are deprecated in favor of *Timestamp.day_name()*, *DatetimeIndex.day_name()*, and *Series.dt.day_name()* (GH12806)
- `pandas.tseries.plotting.tsplot` is deprecated. Use *Series.plot()* instead (GH18627)
- `Index.summary()` is deprecated and will be removed in a future version (GH18217)
- `NDFrame.get_ftype_counts()` is deprecated and will be removed in a future version (GH18243)

- The `convert_datetime64` parameter in *`DataFrame.to_records()`* has been deprecated and will be removed in a future version. The NumPy bug motivating this parameter has been resolved. The default value for this parameter has also changed from `True` to `None` (GH18160).
- `Series.rolling().apply()`, `DataFrame.rolling().apply()`, `Series.expanding().apply()`, and `DataFrame.expanding().apply()` have deprecated passing an `np.array` by default. One will need to pass the new `raw` parameter to be explicit about what is passed (GH20584)
- The `data`, `base`, `strides`, `flags` and `itemsize` properties of the `Series` and `Index` classes have been deprecated and will be removed in a future version (GH20419).
- `DatetimeIndex.offset` is deprecated. Use `DatetimeIndex.freq` instead (GH20716)
- Floor division between an integer ndarray and a *`Timedelta`* is deprecated. Divide by *`Timedelta.value`* instead (GH19761)
- Setting `PeriodIndex.freq` (which was not guaranteed to work correctly) is deprecated. Use *`PeriodIndex.asfreq()`* instead (GH20678)
- `Index.get_duplicates()` is deprecated and will be removed in a future version (GH20239)
- The previous default behavior of negative indices in `Categorical.take` is deprecated. In a future version it will change from meaning missing values to meaning positional indices from the right. The future behavior is consistent with *`Series.take()`* (GH20664).
- Passing multiple axes to the `axis` parameter in *`DataFrame.dropna()`* has been deprecated and will be removed in a future version (GH20987)

Removal of prior version deprecations/changes

- Warnings against the obsolete usage `Categorical(codes, categories)`, which were emitted for instance when the first two arguments to `Categorical()` had different dtypes, and recommended the use of `Categorical.from_codes`, have now been removed (GH8074)
- The `levels` and `labels` attributes of a `MultiIndex` can no longer be set directly (GH4039).
- `pd.tseries.util.pivot_annual` has been removed (deprecated since v0.19). Use `pivot_table` instead (GH18370)
- `pd.tseries.util.isleapyear` has been removed (deprecated since v0.19). Use `.is_leap_year` property in Datetime-likes instead (GH18370)
- `pd.ordered_merge` has been removed (deprecated since v0.19). Use `pd.merge_ordered` instead (GH18459)
- The `SparseList` class has been removed (GH14007)
- The `pandas.io.wb` and `pandas.io.data` stub modules have been removed (GH13735)
- `Categorical.from_array` has been removed (GH13854)
- The `freq` and `how` parameters have been removed from the `rolling`/`expanding`/`ewm` methods of DataFrame and Series (deprecated since v0.18). Instead, resample before calling the methods. (GH18601 & GH18668)
- `DatetimeIndex.to_datetime`, `Timestamp.to_datetime`, `PeriodIndex.to_datetime`, and `Index.to_datetime` have been removed (GH8254, GH14096, GH14113)
- *`read_csv()`* has dropped the `skip_footer` parameter (GH13386)
- *`read_csv()`* has dropped the `as_recarray` parameter (GH13373)
- *`read_csv()`* has dropped the `buffer_lines` parameter (GH13360)

- *read_csv()* has dropped the compact_ints and use_unsigned parameters (GH13323)
- The Timestamp class has dropped the offset attribute in favor of freq (GH13593)
- The Series, Categorical, and Index classes have dropped the reshape method (GH13012)
- pandas.tseries.frequencies.get_standard_freq has been removed in favor of pandas.tseries.frequencies.to_offset(freq).rule_code (GH13874)
- The freqstr keyword has been removed from pandas.tseries.frequencies.to_offset in favor of freq (GH13874)
- The Panel4D and PanelND classes have been removed (GH13776)
- The Panel class has dropped the to_long and toLong methods (GH19077)
- The options display.line_with and display.height are removed in favor of display.width and display.max_rows respectively (GH4391, GH19107)
- The labels attribute of the Categorical class has been removed in favor of *Categorical.codes* (GH7768)
- The flavor parameter have been removed from func:*to_sql* method (GH13611)
- The modules pandas.tools.hashing and pandas.util.hashing have been removed (GH16223)
- The top-level functions pd.rolling_*, pd.expanding_* and pd.ewm* have been removed (Deprecated since v0.18). Instead, use the DataFrame/Series methods *rolling*, *expanding* and *ewm* (GH18723)
- Imports from pandas.core.common for functions such as is_datetime64_dtype are now removed. These are located in pandas.api.types. (GH13634, GH19769)
- The infer_dst keyword in *Series.tz_localize()*, *DatetimeIndex.tz_localize()* and *DatetimeIndex* have been removed. infer_dst=True is equivalent to ambiguous='infer', and infer_dst=False to ambiguous='raise' (GH7963).
- When .resample() was changed from an eager to a lazy operation, like .groupby() in v0.18.0, we put in place compatibility (with a FutureWarning), so operations would continue to work. This is now fully removed, so a Resampler will no longer forward compat operations (GH20554)
- Remove long deprecated axis=None parameter from .replace() (GH20271)

Performance improvements

- Indexers on Series or DataFrame no longer create a reference cycle (GH17956)
- Added a keyword argument, cache, to *to_datetime()* that improved the performance of converting duplicate datetime arguments (GH11665)
- DateOffset arithmetic performance is improved (GH18218)
- Converting a Series of Timedelta objects to days, seconds, etc... sped up through vectorization of underlying methods (GH18092)
- Improved performance of .map() with a Series/dict input (GH15081)
- The overridden Timedelta properties of days, seconds and microseconds have been removed, leveraging their built-in Python versions instead (GH18242)
- Series construction will reduce the number of copies made of the input data in certain cases (GH17449)
- Improved performance of *Series.dt.date()* and *DatetimeIndex.date()* (GH18058)
- Improved performance of *Series.dt.time()* and *DatetimeIndex.time()* (GH18461)

- Improved performance of `IntervalIndex.symmetric_difference()` (GH18475)
- Improved performance of `DatetimeIndex` and `Series` arithmetic operations with Business-Month and Business-Quarter frequencies (GH18489)
- *`Series()`* / *`DataFrame()`* tab completion limits to 100 values, for better performance. (GH18587)
- Improved performance of *`DataFrame.median()`* with `axis=1` when bottleneck is not installed (GH16468)
- Improved performance of *`MultiIndex.get_loc()`* for large indexes, at the cost of a reduction in performance for small ones (GH18519)
- Improved performance of *`MultiIndex.remove_unused_levels()`* when there are no unused levels, at the cost of a reduction in performance when there are (GH19289)
- Improved performance of *`Index.get_loc()`* for non-unique indexes (GH19478)
- Improved performance of pairwise `.rolling()` and `.expanding()` with `.cov()` and `.corr()` operations (GH17917)
- Improved performance of *`pandas.core.groupby.GroupBy.rank()`* (GH15779)
- Improved performance of variable `.rolling()` on `.min()` and `.max()` (GH19521)
- Improved performance of *`pandas.core.groupby.GroupBy.ffill()`* and *`pandas.core.groupby.GroupBy.bfill()`* (GH11296)
- Improved performance of *`pandas.core.groupby.GroupBy.any()`* and *`pandas.core.groupby.GroupBy.all()`* (GH15435)
- Improved performance of *`pandas.core.groupby.GroupBy.pct_change()`* (GH19165)
- Improved performance of *`Series.isin()`* in the case of categorical dtypes (GH20003)
- Improved performance of `getattr(Series, attr)` when the Series has certain index types. This manifested in slow printing of large Series with a `DatetimeIndex` (GH19764)
- Fixed a performance regression for `GroupBy.nth()` and `GroupBy.last()` with some object columns (GH19283)
- Improved performance of `pandas.core.arrays.Categorical.from_codes()` (GH18501)

Documentation changes

Thanks to all of the contributors who participated in the Pandas Documentation Sprint, which took place on March 10th. We had about 500 participants from over 30 locations across the world. You should notice that many of the *API docstrings* have greatly improved.

There were too many simultaneous contributions to include a release note for each improvement, but this GitHub search should give you an idea of how many docstrings were improved.

Special thanks to Marc Garcia for organizing the sprint. For more information, read the NumFOCUS blogpost recapping the sprint.

- Changed spelling of "numpy" to "NumPy", and "python" to "Python". (GH19017)
- Consistency when introducing code samples, using either colon or period. Rewrote some sentences for greater clarity, added more dynamic references to functions, methods and classes. (GH18941, GH18948, GH18973, GH19017)
- Added a reference to *`DataFrame.assign()`* in the concatenate section of the merging documentation (GH18665)

Bug fixes

Categorical

Warning: A class of bugs were introduced in pandas 0.21 with `CategoricalDtype` that affects the correctness of operations like `merge`, `concat`, and indexing when comparing multiple unordered `Categorical` arrays that have the same categories, but in a different order. We highly recommend upgrading or manually aligning your categories before doing these operations.

- Bug in `Categorical.equals` returning the wrong result when comparing two unordered `Categorical` arrays with the same categories, but in a different order (GH16603)
- Bug in *pandas.api.types.union_categoricals()* returning the wrong result when for unordered categoricals with the categories in a different order. This affected *pandas.concat()* with Categorical data (GH19096).
- Bug in *pandas.merge()* returning the wrong result when joining on an unordered `Categorical` that had the same categories but in a different order (GH19551)
- Bug in `CategoricalIndex.get_indexer()` returning the wrong result when `target` was an unordered `Categorical` that had the same categories as `self` but in a different order (GH19551)
- Bug in *Index.astype()* with a categorical dtype where the resultant index is not converted to a *CategoricalIndex* for all types of index (GH18630)
- Bug in *Series.astype()* and `Categorical.astype()` where an existing categorical data does not get updated (GH10696, GH18593)
- Bug in *Series.str.split()* with `expand=True` incorrectly raising an IndexError on empty strings (GH20002).
- Bug in *Index* constructor with `dtype=CategoricalDtype(...)` where `categories` and `ordered` are not maintained (GH19032)
- Bug in *Series* constructor with scalar and `dtype=CategoricalDtype(...)` where `categories` and `ordered` are not maintained (GH19565)
- Bug in `Categorical.__iter__` not converting to Python types (GH19909)
- Bug in *pandas.factorize()* returning the unique codes for the `uniques`. This now returns a `Categorical` with the same dtype as the input (GH19721)
- Bug in *pandas.factorize()* including an item for missing values in the `uniques` return value (GH19721)
- Bug in *Series.take()* with categorical data interpreting `-1` in *indices* as missing value markers, rather than the last element of the Series (GH20664)

Datetimelike

- Bug in `Series.__sub__()` subtracting a non-nanosecond `np.datetime64` object from a `Series` gave incorrect results (GH7996)
- Bug in *`DatetimeIndex`*, *`TimedeltaIndex`* addition and subtraction of zero-dimensional integer arrays gave incorrect results (GH19012)
- Bug in *`DatetimeIndex`* and *`TimedeltaIndex`* where adding or subtracting an array-like of `DateOffset` objects either raised (`np.array`, `pd.Index`) or broadcast incorrectly (`pd.Series`) (GH18849)
- Bug in `Series.__add__()` adding Series with dtype `timedelta64[ns]` to a timezone-aware `DatetimeIndex` incorrectly dropped timezone information (GH13905)
- Adding a `Period` object to a `datetime` or `Timestamp` object will now correctly raise a `TypeError` (GH17983)
- Bug in *`Timestamp`* where comparison with an array of `Timestamp` objects would result in a `RecursionError` (GH15183)
- Bug in *`Series`* floor-division where operating on a scalar `timedelta` raises an exception (GH18846)
- Bug in *`DatetimeIndex`* where the repr was not showing high-precision time values at the end of a day (e.g., 23:59:59.999999999) (GH19030)
- Bug in `.astype()` to non-ns timedelta units would hold the incorrect dtype (GH19176, GH19223, GH12425)
- Bug in subtracting *`Series`* from `NaT` incorrectly returning `NaT` (GH19158)
- Bug in *`Series.truncate()`* which raises `TypeError` with a monotonic `PeriodIndex` (GH17717)
- Bug in *`pct_change()`* using `periods` and `freq` returned different length outputs (GH7292)
- Bug in comparison of *`DatetimeIndex`* against `None` or `datetime.date` objects raising `TypeError` for `==` and `!=` comparisons instead of all-`False` and all-`True`, respectively (GH19301)
- Bug in *`Timestamp`* and *`to_datetime()`* where a string representing a barely out-of-bounds timestamp would be incorrectly rounded down instead of raising `OutOfBoundsDatetime` (GH19382)
- Bug in *`Timestamp.floor()`* *`DatetimeIndex.floor()`* where time stamps far in the future and past were not rounded correctly (GH19206)
- Bug in *`to_datetime()`* where passing an out-of-bounds datetime with `errors='coerce'` and `utc=True` would raise `OutOfBoundsDatetime` instead of parsing to `NaT` (GH19612)
- Bug in *`DatetimeIndex`* and *`TimedeltaIndex`* addition and subtraction where name of the returned object was not always set consistently. (GH19744)
- Bug in *`DatetimeIndex`* and *`TimedeltaIndex`* addition and subtraction where operations with numpy arrays raised `TypeError` (GH19847)
- Bug in *`DatetimeIndex`* and *`TimedeltaIndex`* where setting the `freq` attribute was not fully supported (GH20678)