(continued from previous page)

```
[5 rows x 5 columns]

In [3]: iris.assign(sepal_ratio=iris['SepalWidth'] / iris['SepalLength']).head()
Out[3]:
   SepalLength  SepalWidth  PetalLength  PetalWidth         Name  sepal_ratio
0          5.1         3.5          1.4         0.2  Iris-setosa     0.686275
1          4.9         3.0          1.4         0.2  Iris-setosa     0.612245
2          4.7         3.2          1.3         0.2  Iris-setosa     0.680851
3          4.6         3.1          1.5         0.2  Iris-setosa     0.673913
4          5.0         3.6          1.4         0.2  Iris-setosa     0.720000

[5 rows x 6 columns]
```

Above was an example of inserting a precomputed value. We can also pass in a function to be evaluated.

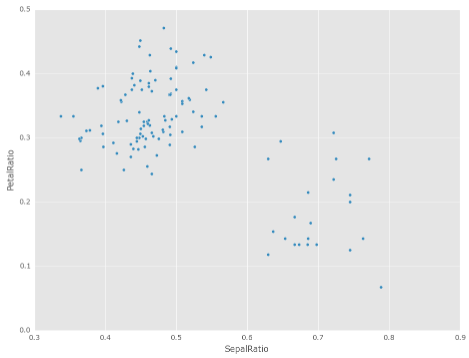
```
In [4]: iris.assign(sepal_ratio=lambda x: (x['SepalWidth']
   ...:                                    / x['SepalLength'])).head()
   ...:
Out[4]:
   SepalLength  SepalWidth  PetalLength  PetalWidth         Name  sepal_ratio
0          5.1         3.5          1.4         0.2  Iris-setosa     0.686275
1          4.9         3.0          1.4         0.2  Iris-setosa     0.612245
2          4.7         3.2          1.3         0.2  Iris-setosa     0.680851
3          4.6         3.1          1.5         0.2  Iris-setosa     0.673913
4          5.0         3.6          1.4         0.2  Iris-setosa     0.720000

[5 rows x 6 columns]
```

The power of `assign` comes when used in chains of operations. For example, we can limit the DataFrame to just those with a Sepal Length greater than 5, calculate the ratio, and plot

```
In [5]: iris = pd.read_csv('data/iris.data')

In [6]: (iris.query('SepalLength > 5')
   ...:       .assign(SepalRatio=lambda x: x.SepalWidth / x.SepalLength,
   ...:               PetalRatio=lambda x: x.PetalWidth / x.PetalLength)
   ...:       .plot(kind='scatter', x='SepalRatio', y='PetalRatio'))
   ...:
Out[6]: <matplotlib.axes._subplots.AxesSubplot at 0x7f52bd600b90>
```



See the *documentation* for more. (GH9229)

Interaction with scipy.sparse

Added `SparseSeries.to_coo()` and `SparseSeries.from_coo()` methods (GH8048) for converting to and from `scipy.sparse.coo_matrix` instances (see *here*). For example, given a SparseSeries with MultiIndex we can convert to a *scipy.sparse.coo_matrix* by specifying the row and column labels as index levels:

```
s = pd.Series([3.0, np.nan, 1.0, 3.0, np.nan, np.nan])
s.index = pd.MultiIndex.from_tuples([(1, 2, 'a', 0),
                                     (1, 2, 'a', 1),
                                     (1, 1, 'b', 0),
                                     (1, 1, 'b', 1),
                                     (2, 1, 'b', 0),
                                     (2, 1, 'b', 1)],
                                    names=['A', 'B', 'C', 'D'])

s

# SparseSeries
ss = s.to_sparse()
ss

A, rows, columns = ss.to_coo(row_levels=['A', 'B'],
                             column_levels=['C', 'D'],
                             sort_labels=False)

A
A.todense()
rows
columns
```

The from_coo method is a convenience method for creating a `SparseSeries` from a `scipy.sparse.coo_matrix`:

```
from scipy import sparse
A = sparse.coo_matrix(([3.0, 1.0, 2.0], ([1, 0, 0], [0, 2, 3])),
                      shape=(3, 4))
A
A.todense()

ss = pd.SparseSeries.from_coo(A)
ss
```

String methods enhancements

- Following new methods are accessible via `.str` accessor to apply the function to each values. This is intended to make it more consistent with standard methods on strings. (GH9282, GH9352, GH9386, GH9387, GH9439)

		Methods		
`isalnum()`	`isalpha()`	`isdigit()`	`isdigit()`	`isspace()`
`islower()`	`isupper()`	`istitle()`	`isnumeric()`	`isdecimal()`
`find()`	`rfind()`	`ljust()`	`rjust()`	`zfill()`

```
In [7]: s = pd.Series(['abcd', '3456', 'EFGH'])
```

(continues on next page)

(continued from previous page)

```
In [8]: s.str.isalpha()
Out[8]:
0     True
1    False
2     True
Length: 3, dtype: bool

In [9]: s.str.find('ab')
Out[9]:
0    0
1   -1
2   -1
Length: 3, dtype: int64
```

- *Series.str.pad()* and *Series.str.center()* now accept `fillchar` option to specify filling character (GH9352)

```
In [10]: s = pd.Series(['12', '300', '25'])

In [11]: s.str.pad(5, fillchar='_')
Out[11]:
0    ___12
1    __300
2    ___25
Length: 3, dtype: object
```

- Added *Series.str.slice_replace()*, which previously raised `NotImplementedError` (GH8888)

```
In [12]: s = pd.Series(['ABCD', 'EFGH', 'IJK'])

In [13]: s.str.slice_replace(1, 3, 'X')
Out[13]:
0    AXD
1    EXH
2     IX
Length: 3, dtype: object

# replaced with empty char
In [14]: s.str.slice_replace(0, 1)
Out[14]:
0    BCD
1    FGH
2     JK
Length: 3, dtype: object
```

Other enhancements

- Reindex now supports `method='nearest'` for frames or series with a monotonic increasing or decreasing index (GH9258):

```
In [15]: df = pd.DataFrame({'x': range(5)})

In [16]: df.reindex([0.2, 1.8, 3.5], method='nearest')
Out[16]:
     x
```

(continues on next page)

(continued from previous page)

```
0.2  0
1.8  2
3.5  4

[3 rows x 1 columns]
```

This method is also exposed by the lower level `Index.get_indexer` and `Index.get_loc` methods.

- The `read_excel()` function's *sheetname* argument now accepts a list and `None`, to get multiple or all sheets respectively. If more than one sheet is specified, a dictionary is returned. (GH9450)

```
# Returns the 1st and 4th sheet, as a dictionary of DataFrames.
pd.read_excel('path_to_file.xls', sheetname=['Sheet1', 3])
```

- Allow Stata files to be read incrementally with an iterator; support for long strings in Stata files. See the docs *here* (GH9493:).
- Paths beginning with ~ will now be expanded to begin with the user's home directory (GH9066)
- Added time interval selection in `get_data_yahoo` (GH9071)
- Added `Timestamp.to_datetime64()` to complement `Timedelta.to_timedelta64()` (GH9255)
- `tseries.frequencies.to_offset()` now accepts `Timedelta` as input (GH9064)
- Lag parameter was added to the autocorrelation method of `Series`, defaults to lag-1 autocorrelation (GH9192)
- `Timedelta` will now accept `nanoseconds` keyword in constructor (GH9273)
- SQL code now safely escapes table and column names (GH8986)
- Added auto-complete for `Series.str.<tab>`, `Series.dt.<tab>` and `Series.cat.<tab>` (GH9322)
- `Index.get_indexer` now supports `method='pad'` and `method='backfill'` even for any target array, not just monotonic targets. These methods also work for monotonic decreasing as well as monotonic increasing indexes (GH9258).
- `Index.asof` now works on all index types (GH9258).
- A `verbose` argument has been augmented in `io.read_excel()`, defaults to False. Set to True to print sheet names as they are parsed. (GH9450)
- Added `days_in_month` (compatibility alias `daysinmonth`) property to `Timestamp`, `DatetimeIndex`, `Period`, `PeriodIndex`, and `Series.dt` (GH9572)
- Added `decimal` option in `to_csv` to provide formatting for non-'.' decimal separators (GH781)
- Added `normalize` option for `Timestamp` to normalized to midnight (GH8794)
- Added example for `DataFrame` import to R using HDF5 file and `rhdf5` library. See the *documentation* for more (GH9636).

Backwards incompatible API changes

Changes in Timedelta

In v0.15.0 a new scalar type `Timedelta` was introduced, that is a sub-class of `datetime.timedelta`. Mentioned *here* was a notice of an API change w.r.t. the `.seconds` accessor. The intent was to provide a user-friendly set of accessors that give the 'natural' value for that unit, e.g. if you had a `Timedelta('1 day, 10:11:12')`, then `.seconds` would return 12. However, this is at odds with the definition of `datetime.timedelta`, which defines `.seconds` as `10 * 3600 + 11 * 60 + 12 == 36672`.

So in v0.16.0, we are restoring the API to match that of `datetime.timedelta`. Further, the component values are still available through the `.components` accessor. This affects the `.seconds` and `.microseconds` accessors, and removes the `.hours`, `.minutes`, `.milliseconds` accessors. These changes affect `TimedeltaIndex` and the Series `.dt` accessor as well. (GH9185, GH9139)

Previous behavior

```
In [2]: t = pd.Timedelta('1 day, 10:11:12.100123')

In [3]: t.days
Out[3]: 1

In [4]: t.seconds
Out[4]: 12

In [5]: t.microseconds
Out[5]: 123
```

New behavior

```
In [17]: t = pd.Timedelta('1 day, 10:11:12.100123')

In [18]: t.days
Out[18]: 1

In [19]: t.seconds
Out[19]: 36672

In [20]: t.microseconds
Out[20]: 100123
```

Using `.components` allows the full component access

```
In [21]: t.components
Out[21]: Components(days=1, hours=10, minutes=11, seconds=12, milliseconds=100,␣
↪microseconds=123, nanoseconds=0)

In [22]: t.components.seconds
Out[22]: 12
```

Indexing changes

The behavior of a small sub-set of edge cases for using `.loc` have changed (GH8613). Furthermore we have improved the content of the error messages that are raised:

- Slicing with `.loc` where the start and/or stop bound is not found in the index is now allowed; this previously would raise a `KeyError`. This makes the behavior the same as `.ix` in this case. This change is only for slicing, not when indexing with a single label.

```
In [23]: df = pd.DataFrame(np.random.randn(5, 4),
   ....:                   columns=list('ABCD'),
   ....:                   index=pd.date_range('20130101', periods=5))
   ....:

In [24]: df
Out[24]:
                   A         B         C         D
2013-01-01  0.469112 -0.282863 -1.509059 -1.135632
2013-01-02  1.212112 -0.173215  0.119209 -1.044236
2013-01-03 -0.861849 -2.104569 -0.494929  1.071804
2013-01-04  0.721555 -0.706771 -1.039575  0.271860
2013-01-05 -0.424972  0.567020  0.276232 -1.087401

[5 rows x 4 columns]

In [25]: s = pd.Series(range(5), [-2, -1, 1, 2, 3])

In [26]: s
Out[26]:
-2    0
-1    1
 1    2
 2    3
 3    4
Length: 5, dtype: int64
```

Previous behavior

```
In [4]: df.loc['2013-01-02':'2013-01-10']
KeyError: 'stop bound [2013-01-10] is not in the [index]'

In [6]: s.loc[-10:3]
KeyError: 'start bound [-10] is not the [index]'
```

New behavior

```
In [27]: df.loc['2013-01-02':'2013-01-10']
Out[27]:
                   A         B         C         D
2013-01-02  1.212112 -0.173215  0.119209 -1.044236
2013-01-03 -0.861849 -2.104569 -0.494929  1.071804
2013-01-04  0.721555 -0.706771 -1.039575  0.271860
2013-01-05 -0.424972  0.567020  0.276232 -1.087401

[4 rows x 4 columns]

In [28]: s.loc[-10:3]
Out[28]:
```

(continues on next page)

(continued from previous page)

```
-2    0
-1    1
 1    2
 2    3
 3    4
Length: 5, dtype: int64
```

- Allow slicing with float-like values on an integer index for `.ix`. Previously this was only enabled for `.loc`:

  Previous behavior

```
In [8]: s.ix[-1.0:2]
TypeError: the slice start value [-1.0] is not a proper indexer for this index␣
↪type (Int64Index)
```

  New behavior

```
In [2]: s.ix[-1.0:2]
Out[2]:
-1    1
 1    2
 2    3
dtype: int64
```

- Provide a useful exception for indexing with an invalid type for that index when using `.loc`. For example trying to use `.loc` on an index of type `DatetimeIndex` or `PeriodIndex` or `TimedeltaIndex`, with an integer (or a float).

  Previous behavior

```
In [4]: df.loc[2:3]
KeyError: 'start bound [2] is not the [index]'
```

  New behavior

```
In [4]: df.loc[2:3]
TypeError: Cannot do slice indexing on <class 'pandas.tseries.index.DatetimeIndex
↪'> with <type 'int'> keys
```

Categorical changes

In prior versions, `Categoricals` that had an unspecified ordering (meaning no `ordered` keyword was passed) were defaulted as `ordered` Categoricals. Going forward, the `ordered` keyword in the `Categorical` constructor will default to `False`. Ordering must now be explicit.

Furthermore, previously you *could* change the `ordered` attribute of a Categorical by just setting the attribute, e.g. `cat.ordered=True`; This is now deprecated and you should use `cat.as_ordered()` or `cat.as_unordered()`. These will by default return a **new** object and not modify the existing object. (GH9347, GH9190)

Previous behavior

```
In [3]: s = pd.Series([0, 1, 2], dtype='category')

In [4]: s
```

(continues on next page)

(continued from previous page)

```
Out[4]:
0    0
1    1
2    2
dtype: category
Categories (3, int64): [0 < 1 < 2]

In [5]: s.cat.ordered
Out[5]: True

In [6]: s.cat.ordered = False

In [7]: s
Out[7]:
0    0
1    1
2    2
dtype: category
Categories (3, int64): [0, 1, 2]
```

New behavior

```
In [29]: s = pd.Series([0, 1, 2], dtype='category')

In [30]: s
Out[30]:
0    0
1    1
2    2
Length: 3, dtype: category
Categories (3, int64): [0, 1, 2]

In [31]: s.cat.ordered
Out[31]: False

In [32]: s = s.cat.as_ordered()

In [33]: s
Out[33]:
0    0
1    1
2    2
Length: 3, dtype: category
Categories (3, int64): [0 < 1 < 2]

In [34]: s.cat.ordered
Out[34]: True

# you can set in the constructor of the Categorical
In [35]: s = pd.Series(pd.Categorical([0, 1, 2], ordered=True))

In [36]: s
Out[36]:
0    0
1    1
2    2
Length: 3, dtype: category
```

(continues on next page)

(continued from previous page)

```
Categories (3, int64): [0 < 1 < 2]

In [37]: s.cat.ordered
Out[37]: True
```

For ease of creation of series of categorical data, we have added the ability to pass keywords when calling `.astype()`. These are passed directly to the constructor.

```
In [54]: s = pd.Series(["a", "b", "c", "a"]).astype('category', ordered=True)

In [55]: s
Out[55]:
0    a
1    b
2    c
3    a
dtype: category
Categories (3, object): [a < b < c]

In [56]: s = (pd.Series(["a", "b", "c", "a"])
   ....:         .astype('category', categories=list('abcdef'), ordered=False))

In [57]: s
Out[57]:
0    a
1    b
2    c
3    a
dtype: category
Categories (6, object): [a, b, c, d, e, f]
```

Other API changes

- `Index.duplicated` now returns `np.array(dtype=bool)` rather than `Index(dtype=object)` containing `bool` values. (GH8875)
- `DataFrame.to_json` now returns accurate type serialisation for each column for frames of mixed dtype (GH9037)

  Previously data was coerced to a common dtype before serialisation, which for example resulted in integers being serialised to floats:

  ```
  In [2]: pd.DataFrame({'i': [1,2], 'f': [3.0, 4.2]}).to_json()
  Out[2]: '{"f":{"0":3.0,"1":4.2},"i":{"0":1.0,"1":2.0}}'
  ```

  Now each column is serialised using its correct dtype:

  ```
  In [2]:  pd.DataFrame({'i': [1,2], 'f': [3.0, 4.2]}).to_json()
  Out[2]: '{"f":{"0":3.0,"1":4.2},"i":{"0":1,"1":2}}'
  ```

- `DatetimeIndex`, `PeriodIndex` and `TimedeltaIndex.summary` now output the same format. (GH9116)
- `TimedeltaIndex.freqstr` now output the same string format as `DatetimeIndex`. (GH9116)

- Bar and horizontal bar plots no longer add a dashed line along the info axis. The prior style can be achieved with matplotlib's `axhline` or `axvline` methods (GH9088).
- `Series` accessors `.dt`, `.cat` and `.str` now raise `AttributeError` instead of `TypeError` if the series does not contain the appropriate type of data (GH9617). This follows Python's built-in exception hierarchy more closely and ensures that tests like `hasattr(s, 'cat')` are consistent on both Python 2 and 3.
- `Series` now supports bitwise operation for integral types (GH9016). Previously even if the input dtypes were integral, the output dtype was coerced to `bool`.

  Previous behavior

```
In [2]: pd.Series([0, 1, 2, 3], list('abcd')) | pd.Series([4, 4, 4, 4], list('abcd
↪'))
Out[2]:
a    True
b    True
c    True
d    True
dtype: bool
```

  New behavior. If the input dtypes are integral, the output dtype is also integral and the output values are the result of the bitwise operation.

```
In [2]: pd.Series([0, 1, 2, 3], list('abcd')) | pd.Series([4, 4, 4, 4], list('abcd
↪'))
Out[2]:
a    4
b    5
c    6
d    7
dtype: int64
```

- During division involving a `Series` or `DataFrame`, `0/0` and `0//0` now give `np.nan` instead of `np.inf`. (GH9144, GH8445)

  Previous behavior

```
In [2]: p = pd.Series([0, 1])

In [3]: p / 0
Out[3]:
0    inf
1    inf
dtype: float64

In [4]: p // 0
Out[4]:
0    inf
1    inf
dtype: float64
```

  New behavior

```
In [38]: p = pd.Series([0, 1])

In [39]: p / 0
Out[39]:
0    NaN
```

(continues on next page)

(continued from previous page)

```
1    inf
Length: 2, dtype: float64

In [40]: p // 0
Out[40]:
0    NaN
1    inf
Length: 2, dtype: float64
```

- `Series.values_counts` and `Series.describe` for categorical data will now put `NaN` entries at the end. (GH9443)
- `Series.describe` for categorical data will now give counts and frequencies of 0, not `NaN`, for unused categories (GH9443)
- Due to a bug fix, looking up a partial string label with `DatetimeIndex.asof` now includes values that match the string, even if they are after the start of the partial string label (GH9258).

  Old behavior:

```
In [4]: pd.to_datetime(['2000-01-31', '2000-02-28']).asof('2000-02')
Out[4]: Timestamp('2000-01-31 00:00:00')
```

  Fixed behavior:

```
In [41]: pd.to_datetime(['2000-01-31', '2000-02-28']).asof('2000-02')
Out[41]: Timestamp('2000-02-28 00:00:00')
```

  To reproduce the old behavior, simply add more precision to the label (e.g., use `2000-02-01` instead of `2000-02`).

Deprecations

- The `rplot` trellis plotting interface is deprecated and will be removed in a future version. We refer to external packages like seaborn for similar but more refined functionality (GH3445). The documentation includes some examples how to convert your existing code from `rplot` to seaborn here.
- The `pandas.sandbox.qtpandas` interface is deprecated and will be removed in a future version. We refer users to the external package pandas-qt. (GH9615)
- The `pandas.rpy` interface is deprecated and will be removed in a future version. Similar functionality can be accessed through the rpy2 project (GH9602)
- Adding `DatetimeIndex/PeriodIndex` to another `DatetimeIndex/PeriodIndex` is being deprecated as a set-operation. This will be changed to a `TypeError` in a future version. `.union()` should be used for the union set operation. (GH9094)
- Subtracting `DatetimeIndex/PeriodIndex` from another `DatetimeIndex/PeriodIndex` is being deprecated as a set-operation. This will be changed to an actual numeric subtraction yielding a `TimeDeltaIndex` in a future version. `.difference()` should be used for the differencing set operation. (GH9094)

Removal of prior version deprecations/changes

- `DataFrame.pivot_table` and `crosstab`'s `rows` and `cols` keyword arguments were removed in favor of `index` and `columns` (GH6581)
- `DataFrame.to_excel` and `DataFrame.to_csv` `cols` keyword argument was removed in favor of `columns` (GH6581)
- Removed `convert_dummies` in favor of `get_dummies` (GH6581)
- Removed `value_range` in favor of `describe` (GH6581)

Performance improvements

- Fixed a performance regression for `.loc` indexing with an array or list-like (GH9126:).
- `DataFrame.to_json` 30x performance improvement for mixed dtype frames. (GH9037)
- Performance improvements in `MultiIndex.duplicated` by working with labels instead of values (GH9125)
- Improved the speed of `nunique` by calling `unique` instead of `value_counts` (GH9129, GH7771)
- Performance improvement of up to 10x in `DataFrame.count` and `DataFrame.dropna` by taking advantage of homogeneous/heterogeneous dtypes appropriately (GH9136)
- Performance improvement of up to 20x in `DataFrame.count` when using a `MultiIndex` and the `level` keyword argument (GH9163)
- Performance and memory usage improvements in `merge` when key space exceeds `int64` bounds (GH9151)
- Performance improvements in multi-key `groupby` (GH9429)
- Performance improvements in `MultiIndex.sortlevel` (GH9445)
- Performance and memory usage improvements in `DataFrame.duplicated` (GH9398)
- Cythonized `Period` (GH9440)
- Decreased memory usage on `to_hdf` (GH9648)

Bug fixes

- Changed `.to_html` to remove leading/trailing spaces in table body (GH4987)
- Fixed issue using `read_csv` on s3 with Python 3 (GH9452)
- Fixed compatibility issue in `DatetimeIndex` affecting architectures where `numpy.int_` defaults to `numpy.int32` (GH8943)
- Bug in Panel indexing with an object-like (GH9140)
- Bug in the returned `Series.dt.components` index was reset to the default index (GH9247)
- Bug in `Categorical.__getitem__/__setitem__` with listlike input getting incorrect results from indexer coercion (GH9469)
- Bug in partial setting with a DatetimeIndex (GH9478)
- Bug in groupby for integer and datetime64 columns when applying an aggregator that caused the value to be changed when the number was sufficiently large (GH9311, GH6620)

- Fixed bug in `to_sql` when mapping a `Timestamp` object column (datetime column with timezone info) to the appropriate sqlalchemy type (GH9085).
- Fixed bug in `to_sql` `dtype` argument not accepting an instantiated SQLAlchemy type (GH9083).
- Bug in `.loc` partial setting with a `np.datetime64` (GH9516)
- Incorrect dtypes inferred on datetimelike looking `Series` & on `.xs` slices (GH9477)
- Items in `Categorical.unique()` (and `s.unique()` if `s` is of dtype `category`) now appear in the order in which they are originally found, not in sorted order (GH9331). This is now consistent with the behavior for other dtypes in pandas.
- Fixed bug on big endian platforms which produced incorrect results in `StataReader` (GH8688).
- Bug in `MultiIndex.has_duplicates` when having many levels causes an indexer overflow (GH9075, GH5873)
- Bug in `pivot` and `unstack` where `nan` values would break index alignment (GH4862, GH7401, GH7403, GH7405, GH7466, GH9497)
- Bug in left `join` on MultiIndex with `sort=True` or null values (GH9210).
- Bug in `MultiIndex` where inserting new keys would fail (GH9250).
- Bug in `groupby` when key space exceeds `int64` bounds (GH9096).
- Bug in `unstack` with `TimedeltaIndex` or `DatetimeIndex` and nulls (GH9491).
- Bug in `rank` where comparing floats with tolerance will cause inconsistent behaviour (GH8365).
- Fixed character encoding bug in `read_stata` and `StataReader` when loading data from a URL (GH9231).
- Bug in adding `offsets.Nano` to other offsets raises `TypeError` (GH9284)
- Bug in `DatetimeIndex` iteration, related to (GH8890), fixed in (GH9100)
- Bugs in `resample` around DST transitions. This required fixing offset classes so they behave correctly on DST transitions. (GH5172, GH8744, GH8653, GH9173, GH9468).
- Bug in binary operator method (eg `.mul()`) alignment with integer levels (GH9463).
- Bug in boxplot, scatter and hexbin plot may show an unnecessary warning (GH8877)
- Bug in subplot with `layout` kw may show unnecessary warning (GH9464)
- Bug in using grouper functions that need passed through arguments (e.g. axis), when using wrapped function (e.g. `fillna`), (GH9221)
- `DataFrame` now properly supports simultaneous `copy` and `dtype` arguments in constructor (GH9099)
- Bug in `read_csv` when using skiprows on a file with CR line endings with the c engine. (GH9079)
- `isnull` now detects `NaT` in `PeriodIndex` (GH9129)
- Bug in groupby `.nth()` with a multiple column groupby (GH8979)
- Bug in `DataFrame.where` and `Series.where` coerce numerics to string incorrectly (GH9280)
- Bug in `DataFrame.where` and `Series.where` raise `ValueError` when string list-like is passed. (GH9280)
- Accessing `Series.str` methods on with non-string values now raises `TypeError` instead of producing incorrect results (GH9184)
- Bug in `DatetimeIndex.__contains__` when index has duplicates and is not monotonic increasing (GH9512)

- Fixed division by zero error for `Series.kurt()` when all values are equal (GH9197)
- Fixed issue in the `xlsxwriter` engine where it added a default 'General' format to cells if no other format was applied. This prevented other row or column formatting being applied. (GH9167)
- Fixes issue with `index_col=False` when `usecols` is also specified in `read_csv`. (GH9082)
- Bug where `wide_to_long` would modify the input stub names list (GH9204)
- Bug in `to_sql` not storing float64 values using double precision. (GH9009)
- `SparseSeries` and `SparsePanel` now accept zero argument constructors (same as their non-sparse counterparts) (GH9272).
- Regression in merging `Categorical` and `object` dtypes (GH9426)
- Bug in `read_csv` with buffer overflows with certain malformed input files (GH9205)
- Bug in groupby MultiIndex with missing pair (GH9049, GH9344)
- Fixed bug in `Series.groupby` where grouping on `MultiIndex` levels would ignore the sort argument (GH9444)
- Fix bug in `DataFrame.Groupby` where `sort=False` is ignored in the case of Categorical columns. (GH8868)
- Fixed bug with reading CSV files from Amazon S3 on python 3 raising a TypeError (GH9452)
- Bug in the Google BigQuery reader where the 'jobComplete' key may be present but False in the query results (GH8728)
- Bug in `Series.values_counts` with excluding `NaN` for categorical type `Series` with `dropna=True` (GH9443)
- Fixed missing numeric_only option for `DataFrame.std/var/sem` (GH9201)
- Support constructing `Panel` or `Panel4D` with scalar data (GH8285)
- `Series` text representation disconnected from *max_rows*/*max_columns* (GH7508).
- `Series` number formatting inconsistent when truncated (GH8532).

  Previous behavior

```
In [2]: pd.options.display.max_rows = 10
In [3]: s = pd.Series([1,1,1,1,1,1,1,1,1,0.9999,1,1]*10)
In [4]: s
Out[4]:
0    1
1    1
2    1
...
127    0.9999
128    1.0000
129    1.0000
Length: 130, dtype: float64
```

  New behavior

```
0      1.0000
1      1.0000
2      1.0000
3      1.0000
4      1.0000
```

(continues on next page)

(continued from previous page)

```
...
125    1.0000
126    1.0000
127    0.9999
128    1.0000
129    1.0000
dtype: float64
```

- A Spurious `SettingWithCopy` Warning was generated when setting a new item in a frame in some cases (GH8730)

  The following would previously report a `SettingWithCopy` Warning.

```
In [42]: df1 = pd.DataFrame({'x': pd.Series(['a', 'b', 'c']),
   ....:                     'y': pd.Series(['d', 'e', 'f'])})
   ....:

In [43]: df2 = df1[['x']]

In [44]: df2['y'] = ['g', 'h', 'i']
```

Contributors

A total of 60 people contributed patches to this release. People with a "+" by their names contributed a patch for the first time.

- Aaron Toth +
- Alan Du +
- Alessandro Amici +
- Artemy Kolchinsky
- Ashwini Chaudhary +
- Ben Schiller
- Bill Letson
- Brandon Bradley +
- Chau Hoang +
- Chris Reynolds
- Chris Whelan +
- Christer van der Meeren +
- David Cottrell +
- David Stephens
- Ehsan Azarnasab +
- Garrett-R +
- Guillaume Gay
- Jake Torcasso +
- Jason Sexauer

- Jeff Reback
- John McNamara
- Joris Van den Bossche
- Joschka zur Jacobsmühlen +
- Juarez Bochi +
- Junya Hayashi +
- K.-Michael Aye
- Kerby Shedden +
- Kevin Sheppard
- Kieran O'Mahony
- Kodi Arfer +
- Matti Airas +
- Min RK +
- Mortada Mehyar
- Robert +
- Scott E Lasley
- Scott Lasley +
- Sergio Pascual +
- Skipper Seabold
- Stephan Hoyer
- Thomas Grainger
- Tom Augspurger
- TomAugspurger
- Vladimir Filimonov +
- Vyomkesh Tripathi +
- Will Holmgren
- Yulong Yang +
- behzad nouri
- bertrandhaut +
- bjonen
- cel4 +
- clham
- hsperr +
- ischwabacher
- jnmclarty
- josham +

- jreback
- omtinez +
- roch +
- sinhrks
- unutbu

5.12 Version 0.15

5.12.1 v0.15.2 (December 12, 2014)

This is a minor release from 0.15.1 and includes a large number of bug fixes along with several new features, enhancements, and performance improvements. A small number of API changes were necessary to fix existing bugs. We recommend that all users upgrade to this version.

- *Enhancements*
- *API Changes*
- *Performance Improvements*
- *Bug Fixes*

API changes

- Indexing in `MultiIndex` beyond lex-sort depth is now supported, though a lexically sorted index will have a better performance. (GH2646)

```
In [1]: df = pd.DataFrame({'jim':[0, 0, 1, 1],
   ...:                    'joe':['x', 'x', 'z', 'y'],
   ...:                    'jolie':np.random.rand(4)}).set_index(['jim', 'joe'])
   ...:

In [2]: df
Out[2]:
            jolie
jim joe
0   x    0.126970
    x    0.966718
1   z    0.260476
    y    0.897237

[4 rows x 1 columns]

In [3]: df.index.lexsort_depth
Out[3]: 1

# in prior versions this would raise a KeyError
# will now show a PerformanceWarning
In [4]: df.loc[(1, 'z')]
Out[4]:
            jolie
jim joe
1   z    0.260476
```

(continues on next page)

(continued from previous page)

```
[1 rows x 1 columns]

# lexically sorting
In [5]: df2 = df.sort_index()

In [6]: df2
Out[6]:
            jolie
jim joe
0   x    0.126970
    x    0.966718
1   y    0.897237
    z    0.260476

[4 rows x 1 columns]

In [7]: df2.index.lexsort_depth
Out[7]: 2

In [8]: df2.loc[(1,'z')]
Out[8]:
            jolie
jim joe
1   z    0.260476

[1 rows x 1 columns]
```

- Bug in unique of Series with `category` dtype, which returned all categories regardless whether they were "used" or not (see GH8559 for the discussion). Previous behaviour was to return all categories:

```
In [3]: cat = pd.Categorical(['a', 'b', 'a'], categories=['a', 'b', 'c'])

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

In [5]: cat.unique()
Out[5]: array(['a', 'b', 'c'], dtype=object)
```

  Now, only the categories that do effectively occur in the array are returned:

```
In [9]: cat = pd.Categorical(['a', 'b', 'a'], categories=['a', 'b', 'c'])

In [10]: cat.unique()
Out[10]:
[a, b]
Categories (2, object): [a, b]
```

- `Series.all` and `Series.any` now support the `level` and `skipna` parameters. `Series.all`, `Series.any`, `Index.all`, and `Index.any` no longer support the `out` and `keepdims` parameters, which existed for compatibility with ndarray. Various index types no longer support the `all` and `any` aggregation functions and will now raise `TypeError`. (GH8302).
- Allow equality comparisons of Series with a categorical dtype and object dtype; previously these would raise `TypeError` (GH8938)

- Bug in `NDFrame`: conflicting attribute/column names now behave consistently between getting and setting. Previously, when both a column and attribute named `y` existed, `data.y` would return the attribute, while `data.y = z` would update the column (GH8994)

```
In [11]: data = pd.DataFrame({'x': [1, 2, 3]})

In [12]: data.y = 2

In [13]: data['y'] = [2, 4, 6]

In [14]: data
Out[14]:
   x  y
0  1  2
1  2  4
2  3  6

[3 rows x 2 columns]

# this assignment was inconsistent
In [15]: data.y = 5
```

Old behavior:

```
In [6]: data.y
Out[6]: 2

In [7]: data['y'].values
Out[7]: array([5, 5, 5])
```

New behavior:

```
In [16]: data.y
Out[16]: 5

In [17]: data['y'].values
Out[17]: array([2, 4, 6])
```

- `Timestamp('now')` is now equivalent to `Timestamp.now()` in that it returns the local time rather than UTC. Also, `Timestamp('today')` is now equivalent to `Timestamp.today()` and both have `tz` as a possible argument. (GH9000)
- Fix negative step support for label-based slices (GH8753)

Old behavior:

```
In [1]: s = pd.Series(np.arange(3), ['a', 'b', 'c'])
Out[1]:
a    0
b    1
c    2
dtype: int64

In [2]: s.loc['c':'a':-1]
Out[2]:
c    2
dtype: int64
```

New behavior:

```
In [18]: s = pd.Series(np.arange(3), ['a', 'b', 'c'])

In [19]: s.loc['c':'a':-1]
Out[19]:
c    2
b    1
a    0
Length: 3, dtype: int64
```

Enhancements

`Categorical` enhancements:

- Added ability to export Categorical data to Stata (GH8633). See *here* for limitations of categorical variables exported to Stata data files.
- Added flag `order_categoricals` to `StataReader` and `read_stata` to select whether to order imported categorical data (GH8836). See *here* for more information on importing categorical variables from Stata data files.
- Added ability to export Categorical data to to/from HDF5 (GH7621). Queries work the same as if it was an object array. However, the `category` dtyped data is stored in a more efficient manner. See *here* for an example and caveats w.r.t. prior versions of pandas.
- Added support for `searchsorted()` on *Categorical* class (GH8420).

Other enhancements:

- Added the ability to specify the SQL type of columns when writing a DataFrame to a database (GH8778). For example, specifying to use the sqlalchemy `String` type instead of the default `Text` type for string columns:

```
from sqlalchemy.types import String
data.to_sql('data_dtype', engine, dtype={'Col_1': String})  # noqa F821
```

- `Series.all` and `Series.any` now support the `level` and `skipna` parameters (GH8302):

```
In [20]: s = pd.Series([False, True, False], index=[0, 0, 1])

In [21]: s.any(level=0)
Out[21]:
0     True
1    False
Length: 2, dtype: bool
```

- `Panel` now supports the `all` and `any` aggregation functions. (GH8302):

```
>>> p = pd.Panel(np.random.rand(2, 5, 4) > 0.1)
>>> p.all()
       0      1      2     3
0   True   True   True  True
1   True  False   True  True
2   True   True   True  True
3  False   True  False  True
4   True   True   True  True
```

- Added support for `utcfromtimestamp()`, `fromtimestamp()`, and `combine()` on *Timestamp* class (GH5351).

- Added Google Analytics (*pandas.io.ga*) basic documentation (GH8835). See here.
- `Timedelta` arithmetic returns `NotImplemented` in unknown cases, allowing extensions by custom classes (GH8813).
- `Timedelta` now supports arithmetic with `numpy.ndarray` objects of the appropriate dtype (numpy 1.8 or newer only) (GH8884).
- Added `Timedelta.to_timedelta64()` method to the public API (GH8884).
- Added `gbq.generate_bq_schema()` function to the gbq module (GH8325).
- `Series` now works with map objects the same way as generators (GH8909).
- Added context manager to `HDFStore` for automatic closing (GH8791).
- `to_datetime` gains an `exact` keyword to allow for a format to not require an exact match for a provided format string (if its `False`). `exact` defaults to `True` (meaning that exact matching is still the default) (GH8904)
- Added `axvlines` boolean option to parallel_coordinates plot function, determines whether vertical lines will be printed, default is True
- Added ability to read table footers to read_html (GH8552)
- `to_sql` now infers data types of non-NA values for columns that contain NA values and have dtype `object` (GH8778).

Performance

- Reduce memory usage when skiprows is an integer in read_csv (GH8681)
- Performance boost for `to_datetime` conversions with a passed `format=`, and the `exact=False` (GH8904)

Bug fixes

- Bug in concat of Series with `category` dtype which were coercing to `object`. (GH8641)
- Bug in Timestamp-Timestamp not returning a Timedelta type and datelike-datelike ops with timezones (GH8865)
- Made consistent a timezone mismatch exception (either tz operated with None or incompatible timezone), will now return `TypeError` rather than `ValueError` (a couple of edge cases only), (GH8865)
- Bug in using a `pd.Grouper(key=...)` with no level/axis or level only (GH8795, GH8866)
- Report a `TypeError` when invalid/no parameters are passed in a groupby (GH8015)
- Bug in packaging pandas with `py2app/cx_Freeze` (GH8602, GH8831)
- Bug in `groupby` signatures that didn't include *args or **kwargs (GH8733).
- `io.data.Options` now raises `RemoteDataError` when no expiry dates are available from Yahoo and when it receives no data from Yahoo (GH8761), (GH8783).
- Unclear error message in csv parsing when passing dtype and names and the parsed data is a different data type (GH8833)
- Bug in slicing a MultiIndex with an empty list and at least one boolean indexer (GH8781)
- `io.data.Options` now raises `RemoteDataError` when no expiry dates are available from Yahoo (GH8761).
- `Timedelta` kwargs may now be numpy ints and floats (GH8757).

- Fixed several outstanding bugs for `Timedelta` arithmetic and comparisons (GH8813, GH5963, GH5436).
- `sql_schema` now generates dialect appropriate `CREATE TABLE` statements (GH8697)
- `slice` string method now takes step into account (GH8754)
- Bug in `BlockManager` where setting values with different type would break block integrity (GH8850)
- Bug in `DatetimeIndex` when using `time` object as key (GH8667)
- Bug in `merge` where `how='left'` and `sort=False` would not preserve left frame order (GH7331)
- Bug in `MultiIndex.reindex` where reindexing at level would not reorder labels (GH4088)
- Bug in certain operations with dateutil timezones, manifesting with dateutil 2.3 (GH8639)
- Regression in DatetimeIndex iteration with a Fixed/Local offset timezone (GH8890)
- Bug in `to_datetime` when parsing a nanoseconds using the `%f` format (GH8989)
- `io.data.Options` now raises `RemoteDataError` when no expiry dates are available from Yahoo and when it receives no data from Yahoo (GH8761), (GH8783).
- Fix: The font size was only set on x axis if vertical or the y axis if horizontal. (GH8765)
- Fixed division by 0 when reading big csv files in python 3 (GH8621)
- Bug in outputting a MultiIndex with `to_html,index=False` which would add an extra column (GH8452)
- Imported categorical variables from Stata files retain the ordinal information in the underlying data (GH8836).
- Defined `.size` attribute across `NDFrame` objects to provide compat with numpy >= 1.9.1; buggy with `np.array_split` (GH8846)
- Skip testing of histogram plots for matplotlib <= 1.2 (GH8648).
- Bug where `get_data_google` returned object dtypes (GH3995)
- Bug in `DataFrame.stack(..., dropna=False)` when the DataFrame's `columns` is a `MultiIndex` whose `labels` do not reference all its `levels`. (GH8844)
- Bug in that Option context applied on `__enter__` (GH8514)
- Bug in resample that causes a ValueError when resampling across multiple days and the last offset is not calculated from the start of the range (GH8683)
- Bug where `DataFrame.plot(kind='scatter')` fails when checking if an np.array is in the DataFrame (GH8852)
- Bug in `pd.infer_freq/DataFrame.inferred_freq` that prevented proper sub-daily frequency inference when the index contained DST days (GH8772).
- Bug where index name was still used when plotting a series with `use_index=False` (GH8558).
- Bugs when trying to stack multiple columns, when some (or all) of the level names are numbers (GH8584).
- Bug in `MultiIndex` where `__contains__` returns wrong result if index is not lexically sorted or unique (GH7724)
- BUG CSV: fix problem with trailing white space in skipped rows, (GH8679), (GH8661), (GH8983)
- Regression in `Timestamp` does not parse 'Z' zone designator for UTC (GH8771)
- Bug in *StataWriter* the produces writes strings with 244 characters irrespective of actual size (GH8969)
- Fixed ValueError raised by cummin/cummax when datetime64 Series contains NaT. (GH8965)
- Bug in DataReader returns object dtype if there are missing values (GH8980)

- Bug in plotting if sharex was enabled and index was a timeseries, would show labels on multiple axes (GH3964).
- Bug where passing a unit to the TimedeltaIndex constructor applied the to nano-second conversion twice. (GH9011).
- Bug in plotting of a period-like array (GH9012)

Contributors

A total of 49 people contributed patches to this release. People with a "+" by their names contributed a patch for the first time.

- Aaron Staple
- Angelos Evripiotis +
- Artemy Kolchinsky
- Benoit Pointet +
- Brian Jacobowski +
- Charalampos Papaloizou +
- Chris Warth +
- David Stephens
- Fabio Zanini +
- Francesc Via +
- Henry Kleynhans +
- Jake VanderPlas +
- Jan Schulz
- Jeff Reback
- Jeff Tratner
- Joris Van den Bossche
- Kevin Sheppard
- Matt Suggit +
- Matthew Brett
- Phillip Cloud
- Rupert Thompson +
- Scott E Lasley +
- Stephan Hoyer
- Stephen Simmons +
- Sylvain Corlay +
- Thomas Grainger +
- Tiago Antao +
- Tom Augspurger
- Trent Hauck

- Victor Chaves +
- Victor Salgado +
- Vikram Bhandoh +
- WANG Aiyong
- Will Holmgren +
- behzad nouri
- broessli +
- charalampos papaloizou +
- immerrr
- jnmclarty
- jreback
- mgilbert +
- onesandzeroes
- peadarcoyle +
- rockg
- seth-p
- sinhrks
- unutbu
- wavedatalab +
- Åsmund Hjulstad +

5.12.2 v0.15.1 (November 9, 2014)

This is a minor bug-fix release from 0.15.0 and includes a small number of API changes, several new features, enhancements, and performance improvements along with a large number of bug fixes. We recommend that all users upgrade to this version.

- *Enhancements*
- *API Changes*
- *Bug Fixes*

API changes

- `s.dt.hour` and other `.dt` accessors will now return `np.nan` for missing values (rather than previously -1), (GH8689)

```
In [1]: s = pd.Series(pd.date_range('20130101', periods=5, freq='D'))

In [2]: s.iloc[2] = np.nan

In [3]: s
Out[3]:
```

(continues on next page)

(continued from previous page)

```
0   2013-01-01
1   2013-01-02
2          NaT
3   2013-01-04
4   2013-01-05
Length: 5, dtype: datetime64[ns]
```

previous behavior:

```
In [6]: s.dt.hour
Out[6]:
0    0
1    0
2   -1
3    0
4    0
dtype: int64
```

current behavior:

```
In [4]: s.dt.hour
Out[4]:
0    0.0
1    0.0
2    NaN
3    0.0
4    0.0
Length: 5, dtype: float64
```

- `groupby` with `as_index=False` will not add erroneous extra columns to result (GH8582):

```
In [5]: np.random.seed(2718281)

In [6]: df = pd.DataFrame(np.random.randint(0, 100, (10, 2)),
   ...:                   columns=['jim', 'joe'])
   ...:

In [7]: df.head()
Out[7]:
   jim  joe
0   61   81
1   96   49
2   55   65
3   72   51
4   77   12

[5 rows x 2 columns]

In [8]: ts = pd.Series(5 * np.random.randint(0, 3, 10))
```

previous behavior:

```
In [4]: df.groupby(ts, as_index=False).max()
Out[4]:
   NaN  jim  joe
0    0   72   83
```

(continues on next page)