String

- *Series.str.split()* will now propagate NaN values across all expanded columns instead of None (GH18450)

Contributors

A total of 46 people contributed patches to this release. People with a "+" by their names contributed a patch for the first time.

- Aaron Critchley +
- Alex Rychyk
- Alexander Buchkovsky +
- Alexander Michael Schade +
- Chris Mazzullo
- Cornelius Riemenschneider +
- Dave Hirschfeld +
- David Fischer +
- David Stansby +
- Dror Atariah +
- Eric Kisslinger +
- Hans +
- Ingolf Becker +
- Jan Werkmann +
- Jeff Reback
- Joris Van den Bossche
- Jörg Döpfert +
- Kevin Kuhl +
- Krzysztof Chomski +
- Leif Walsh
- Licht Takeuchi
- Manraj Singh +
- Matt Braymer-Hayes +
- Michael Waskom +
- Mie~~~ +
- Peter Hoffmann +
- Robert Meyer +
- Sam Cohan +
- Sietse Brouwer +

- Sven +
- Tim Swast
- Tom Augspurger
- Wes Turner
- William Ayd +
- Yee Mey +
- bolkedebruin +
- cgohlke
- derestle-htwg +
- fjdiod +
- gabrielclow +
- gfyoung
- ghasemnaddaf +
- jbrockmendel
- jschendel
- miker985 +
- topper-123

5.6.2 v0.21.0 (October 27, 2017)

This is a major release from 0.20.3 and includes a number of API changes, deprecations, new features, enhancements, and performance improvements along with a large number of bug fixes. We recommend that all users upgrade to this version.

Highlights include:

- Integration with Apache Parquet, including a new top-level *read_parquet()* function and *DataFrame.to_parquet()* method, see *here*.
- New user-facing `pandas.api.types.CategoricalDtype` for specifying categoricals independent of the data, see *here*.
- The behavior of `sum` and `prod` on all-NaN Series/DataFrames is now consistent and no longer depends on whether bottleneck is installed, and `sum` and `prod` on empty Series now return NaN instead of 0, see *here*.
- Compatibility fixes for pypy, see *here*.
- Additions to the `drop`, `reindex` and `rename` API to make them more consistent, see *here*.
- Addition of the new methods `DataFrame.infer_objects` (see *here*) and `GroupBy.pipe` (see *here*).
- Indexing with a list of labels, where one or more of the labels is missing, is deprecated and will raise a KeyError in a future version, see *here*.

Check the *API Changes* and *deprecations* before updating.

What's new in v0.21.0

New features

Integration with Apache Parquet file format

Integration with Apache Parquet, including a new top-level `read_parquet()` and `DataFrame.to_parquet()` method, see *here* (GH15838, GH17438).

Apache Parquet provides a cross-language, binary file format for reading and writing data frames efficiently. Parquet is designed to faithfully serialize and de-serialize `DataFrame` s, supporting all of the pandas dtypes, including extension dtypes such as datetime with timezones.

This functionality depends on either the pyarrow or fastparquet library. For more details, see see *the IO docs on Parquet*.

`infer_objects` type conversion

The `DataFrame.infer_objects()` and `Series.infer_objects()` methods have been added to perform dtype inference on object columns, replacing some of the functionality of the deprecated `convert_objects` method. See the documentation *here* for more details. (GH11221)

This method only performs soft conversions on object columns, converting Python objects to native types, but not any coercive conversions. For example:

```
In [1]: df = pd.DataFrame({'A': [1, 2, 3],
   ...:                    'B': np.array([1, 2, 3], dtype='object'),
   ...:                    'C': ['1', '2', '3']})
   ...:

In [2]: df.dtypes
Out[2]:
A     int64
B    object
C    object
Length: 3, dtype: object

In [3]: df.infer_objects().dtypes
Out[3]:
A     int64
B     int64
```

(continues on next page)

(continued from previous page)

```
C    object
Length: 3, dtype: object
```

Note that column `'C'` was not converted - only scalar numeric types will be converted to a new type. Other types of conversion should be accomplished using the *to_numeric()* function (or *to_datetime()*, *to_timedelta()*).

```
In [4]: df = df.infer_objects()

In [5]: df['C'] = pd.to_numeric(df['C'], errors='coerce')

In [6]: df.dtypes
Out[6]:
A    int64
B    int64
C    int64
Length: 3, dtype: object
```

Improved warnings when attempting to create columns

New users are often puzzled by the relationship between column operations and attribute access on `DataFrame` instances (GH7175). One specific instance of this confusion is attempting to create a new column by setting an attribute on the `DataFrame`:

```
In [1]: df = pd.DataFrame({'one': [1., 2., 3.]})
In [2]: df.two = [4, 5, 6]
```

This does not raise any obvious exceptions, but also does not create a new column:

```
In [3]: df
Out[3]:
    one
0  1.0
1  2.0
2  3.0
```

Setting a list-like data structure into a new attribute now raises a `UserWarning` about the potential for unexpected behavior. See *Attribute Access*.

`drop` now also accepts index/columns keywords

The *drop()* method has gained `index`/`columns` keywords as an alternative to specifying the `axis`. This is similar to the behavior of `reindex` (GH12392).

For example:

```
In [7]: df = pd.DataFrame(np.arange(8).reshape(2, 4),
   ...:                   columns=['A', 'B', 'C', 'D'])
   ...:

In [8]: df
Out[8]:
   A  B  C  D
```

(continues on next page)

(continued from previous page)

```
0  0  1  2  3
1  4  5  6  7

[2 rows x 4 columns]

In [9]: df.drop(['B', 'C'], axis=1)
Out[9]:
   A  D
0  0  3
1  4  7

[2 rows x 2 columns]

# the following is now equivalent
In [10]: df.drop(columns=['B', 'C'])
Out[10]:
   A  D
0  0  3
1  4  7

[2 rows x 2 columns]
```

`rename`, `reindex` now also accept axis keyword

The *`DataFrame.rename()`* and *`DataFrame.reindex()`* methods have gained the `axis` keyword to specify the axis to target with the operation (GH12392).

Here's `rename`:

```
In [11]: df = pd.DataFrame({"A": [1, 2, 3], "B": [4, 5, 6]})

In [12]: df.rename(str.lower, axis='columns')
Out[12]:
   a  b
0  1  4
1  2  5
2  3  6

[3 rows x 2 columns]

In [13]: df.rename(id, axis='index')
Out[13]:
                 A  B
94622462542432   1  4
94622462542464   2  5
94622462542496   3  6

[3 rows x 2 columns]
```

And `reindex`:

```
In [14]: df.reindex(['A', 'B', 'C'], axis='columns')
Out[14]:
   A  B   C
0  1  4 NaN
```

(continues on next page)

(continued from previous page)

```
1  2  5 NaN
2  3  6 NaN

[3 rows x 3 columns]

In [15]: df.reindex([0, 1, 3], axis='index')
Out[15]:
     A    B
0  1.0  4.0
1  2.0  5.0
3  NaN  NaN

[3 rows x 2 columns]
```

The "index, columns" style continues to work as before.

```
In [16]: df.rename(index=id, columns=str.lower)
Out[16]:
                a  b
94622462542432  1  4
94622462542464  2  5
94622462542496  3  6

[3 rows x 2 columns]

In [17]: df.reindex(index=[0, 1, 3], columns=['A', 'B', 'C'])
Out[17]:
     A    B   C
0  1.0  4.0 NaN
1  2.0  5.0 NaN
3  NaN  NaN NaN

[3 rows x 3 columns]
```

We *highly* encourage using named arguments to avoid confusion when using either style.

`CategoricalDtype` for specifying categoricals

`pandas.api.types.CategoricalDtype` has been added to the public API and expanded to include the `categories` and `ordered` attributes. A `CategoricalDtype` can be used to specify the set of categories and orderedness of an array, independent of the data. This can be useful for example, when converting string data to a `Categorical` (GH14711, GH15078, GH16015, GH17643):

```
In [18]: from pandas.api.types import CategoricalDtype

In [19]: s = pd.Series(['a', 'b', 'c', 'a'])  # strings

In [20]: dtype = CategoricalDtype(categories=['a', 'b', 'c', 'd'], ordered=True)

In [21]: s.astype(dtype)
Out[21]:
0    a
1    b
2    c
3    a
```

(continues on next page)

(continued from previous page)

```
Length: 4, dtype: category
Categories (4, object): [a < b < c < d]
```

One place that deserves special mention is in *read_csv()*. Previously, with `dtype={'col': 'category'}`, the returned values and categories would always be strings.

```
In [22]: data = 'A,B\na,1\nb,2\nc,3'

In [23]: pd.read_csv(StringIO(data), dtype={'B': 'category'}).B.cat.categories
Out[23]: Index(['1', '2', '3'], dtype='object')
```

Notice the "object" dtype.

With a `CategoricalDtype` of all numerics, datetimes, or timedeltas, we can automatically convert to the correct type

```
In [24]: dtype = {'B': CategoricalDtype([1, 2, 3])}

In [25]: pd.read_csv(StringIO(data), dtype=dtype).B.cat.categories
Out[25]: Int64Index([1, 2, 3], dtype='int64')
```

The values have been correctly interpreted as integers.

The `.dtype` property of a `Categorical`, `CategoricalIndex` or a `Series` with categorical type will now return an instance of `CategoricalDtype`. While the repr has changed, `str(CategoricalDtype())` is still the string `'category'`. We'll take this moment to remind users that the *preferred* way to detect categorical data is to use *pandas.api.types.is_categorical_dtype()*, and not `str(dtype) == 'category'`.

See the *CategoricalDtype docs* for more.

`GroupBy` objects now have a `pipe` method

`GroupBy` objects now have a `pipe` method, similar to the one on `DataFrame` and `Series`, that allow for functions that take a `GroupBy` to be composed in a clean, readable syntax. (GH17871)

For a concrete example on combining `.groupby` and `.pipe` , imagine having a DataFrame with columns for stores, products, revenue and sold quantity. We'd like to do a groupwise calculation of *prices* (i.e. revenue/quantity) per store and per product. We could do this in a multi-step operation, but expressing it in terms of piping can make the code more readable.

First we set the data:

```
In [26]: import numpy as np

In [27]: n = 1000

In [28]: df = pd.DataFrame({'Store': np.random.choice(['Store_1', 'Store_2'], n),
   ....:                    'Product': np.random.choice(['Product_1',
   ....:                                                 'Product_2',
   ....:                                                 'Product_3'
   ....:                                                 ], n),
   ....:                    'Revenue': (np.random.random(n) * 50 + 10).round(2),
   ....:                    'Quantity': np.random.randint(1, 10, size=n)})
   ....:

In [29]: df.head(2)
```

(continues on next page)

(continued from previous page)

```
Out[29]:
     Store    Product  Revenue  Quantity
0  Store_2  Product_2    32.09         7
1  Store_1  Product_3    14.20         1

[2 rows x 4 columns]
```

Now, to find prices per store/product, we can simply do:

```
In [30]: (df.groupby(['Store', 'Product'])
   ....:     .pipe(lambda grp: grp.Revenue.sum() / grp.Quantity.sum())
   ....:     .unstack().round(2))
   ....:
Out[30]:
Product  Product_1  Product_2  Product_3
Store
Store_1       6.73       6.72       7.14
Store_2       7.59       6.98       7.23

[2 rows x 3 columns]
```

See the *documentation* for more.

`Categorical.rename_categories` accepts a dict-like

`rename_categories()` now accepts a dict-like argument for `new_categories`. The previous categories are looked up in the dictionary's keys and replaced if found. The behavior of missing and extra keys is the same as in *`DataFrame.rename()`*.

```
In [31]: c = pd.Categorical(['a', 'a', 'b'])

In [32]: c.rename_categories({"a": "eh", "b": "bee"})
Out[32]:
[eh, eh, bee]
Categories (2, object): [eh, bee]
```

Warning: To assist with upgrading pandas, `rename_categories` treats `Series` as list-like. Typically, Series are considered to be dict-like (e.g. in `.rename`, `.map`). In a future version of pandas `rename_categories` will change to treat them as dict-like. Follow the warning message's recommendations for writing future-proof code.

```
In [33]: c.rename_categories(pd.Series([0, 1], index=['a', 'c']))
FutureWarning: Treating Series 'new_categories' as a list-like and using the values.
In a future version, 'rename_categories' will treat Series like a dictionary.
For dict-like, use 'new_categories.to_dict()'
For list-like, use 'new_categories.values'.
Out[33]:
[0, 0, 1]
Categories (2, int64): [0, 1]
```

Other enhancements

New functions or methods

- *nearest()* is added to support nearest-neighbor upsampling (GH17496).
- *Index* has added support for a to_frame method (GH15230).

New keywords

- Added a skipna parameter to *infer_dtype()* to support type inference in the presence of missing values (GH17059).
- *Series.to_dict()* and *DataFrame.to_dict()* now support an into keyword which allows you to specify the collections.Mapping subclass that you would like returned. The default is dict, which is backwards compatible. (GH16122)
- *Series.set_axis()* and *DataFrame.set_axis()* now support the inplace parameter. (GH14636)
- *Series.to_pickle()* and *DataFrame.to_pickle()* have gained a protocol parameter (GH16252). By default, this parameter is set to HIGHEST_PROTOCOL
- *read_feather()* has gained the nthreads parameter for multi-threaded operations (GH16359)
- *DataFrame.clip()* and *Series.clip()* have gained an inplace argument. (GH15388)
- *crosstab()* has gained a margins_name parameter to define the name of the row / column that will contain the totals when margins=True. (GH15972)
- *read_json()* now accepts a chunksize parameter that can be used when lines=True. If chunksize is passed, read_json now returns an iterator which reads in chunksize lines with each iteration. (GH17048)
- *read_json()* and *to_json()* now accept a compression argument which allows them to transparently handle compressed files. (GH17798)

Various enhancements

- Improved the import time of pandas by about 2.25x. (GH16764)
- Support for PEP 519 – Adding a file system path protocol on most readers (e.g. *read_csv()*) and writers (e.g. *DataFrame.to_csv()*) (GH13823).
- Added a __fspath__ method to pd.HDFStore, pd.ExcelFile, and pd.ExcelWriter to work properly with the file system path protocol (GH13823).
- The validate argument for *merge()* now checks whether a merge is one-to-one, one-to-many, many-to-one, or many-to-many. If a merge is found to not be an example of specified merge type, an exception of type MergeError will be raised. For more, see *here* (GH16270)
- Added support for PEP 518 (pyproject.toml) to the build system (GH16745)
- RangeIndex.append() now returns a RangeIndex object when possible (GH16212)
- *Series.rename_axis()* and *DataFrame.rename_axis()* with inplace=True now return None while renaming the axis inplace. (GH15704)
- *api.types.infer_dtype()* now infers decimals. (GH15690)
- *DataFrame.select_dtypes()* now accepts scalar values for include/exclude as well as list-like. (GH16855)

- *date_range()* now accepts 'YS' in addition to 'AS' as an alias for start of year. (GH9313)
- *date_range()* now accepts 'Y' in addition to 'A' as an alias for end of year. (GH9313)
- *DataFrame.add_prefix()* and *DataFrame.add_suffix()* now accept strings containing the '%' character. (GH17151)
- Read/write methods that infer compression (*read_csv()*, *read_table()*, *read_pickle()*, and *to_pickle()*) can now infer from path-like objects, such as `pathlib.Path`. (GH17206)
- *read_sas()* now recognizes much more of the most frequently used date (datetime) formats in SAS7BDAT files. (GH15871)
- *DataFrame.items()* and *Series.items()* are now present in both Python 2 and 3 and is lazy in all cases. (GH13918, GH17213)
- *pandas.io.formats.style.Styler.where()* has been implemented as a convenience for *pandas.io.formats.style.Styler.applymap()*. (GH17474)
- `MultiIndex.is_monotonic_decreasing()` has been implemented. Previously returned `False` in all cases. (GH16554)
- *read_excel()* raises `ImportError` with a better message if `xlrd` is not installed. (GH17613)
- *DataFrame.assign()* will preserve the original order of `**kwargs` for Python 3.6+ users instead of sorting the column names. (GH14207)
- *Series.reindex()*, *DataFrame.reindex()*, *Index.get_indexer()* now support list-like argument for `tolerance`. (GH17367)

Backwards incompatible API changes

Dependencies have increased minimum versions

We have updated our minimum supported versions of dependencies (GH15206, GH15543, GH15214). If installed, we now require:

Package	Minimum Version	Required
Numpy	1.9.0	X
Matplotlib	1.4.3	
Scipy	0.14.0	
Bottleneck	1.0.0	

Additionally, support has been dropped for Python 3.4 (GH15251).

Sum/Prod of all-NaN or empty Series/DataFrames is now consistently NaN

Note: The changes described here have been partially reverted. See the *v0.22.0 Whatsnew* for more.

The behavior of `sum` and `prod` on all-NaN Series/DataFrames no longer depends on whether bottleneck is installed, and return value of `sum` and `prod` on an empty Series has changed (GH9422, GH15507).

Calling `sum` or `prod` on an empty or all-`NaN` `Series`, or columns of a `DataFrame`, will result in `NaN`. See the *docs*.

```
In [33]: s = pd.Series([np.nan])
```

Previously WITHOUT `bottleneck` installed:

```
In [2]: s.sum()
Out[2]: np.nan
```

Previously WITH `bottleneck`:

```
In [2]: s.sum()
Out[2]: 0.0
```

New behavior, without regard to the bottleneck installation:

```
In [34]: s.sum()
Out[34]: 0.0
```

Note that this also changes the sum of an empty `Series`. Previously this always returned 0 regardless of a `bottleneck` installation:

```
In [1]: pd.Series([]).sum()
Out[1]: 0
```

but for consistency with the all-NaN case, this was changed to return NaN as well:

```
In [35]: pd.Series([]).sum()
Out[35]: 0.0
```

Indexing with a list with missing labels is deprecated

Previously, selecting with a list of labels, where one or more labels were missing would always succeed, returning `NaN` for missing labels. This will now show a `FutureWarning`. In the future this will raise a `KeyError` (GH15747). This warning will trigger on a `DataFrame` or a `Series` for using `.loc[]` or `[[]]` when passing a list-of-labels with at least 1 missing label. See the *deprecation docs*.

```
In [36]: s = pd.Series([1, 2, 3])

In [37]: s
Out[37]:
0    1
1    2
2    3
Length: 3, dtype: int64
```

Previous behavior

```
In [4]: s.loc[[1, 2, 3]]
Out[4]:
1    2.0
2    3.0
3    NaN
dtype: float64
```

Current behavior

```
In [4]: s.loc[[1, 2, 3]]
Passing list-likes to .loc or [] with any missing label will raise
KeyError in the future, you can use .reindex() as an alternative.

See the documentation here:
https://pandas.pydata.org/pandas-docs/stable/indexing.html#deprecate-loc-reindex-
↪listlike

Out[4]:
1    2.0
2    3.0
3    NaN
dtype: float64
```

The idiomatic way to achieve selecting potentially not-found elements is via `.reindex()`

```
In [38]: s.reindex([1, 2, 3])
Out[38]:
1    2.0
2    3.0
3    NaN
Length: 3, dtype: float64
```

Selection with all keys found is unchanged.

```
In [39]: s.loc[[1, 2]]
Out[39]:
1    2
2    3
Length: 2, dtype: int64
```

NA naming changes

In order to promote more consistency among the pandas API, we have added additional top-level functions *isna()* and *notna()* that are aliases for *isnull()* and *notnull()*. The naming scheme is now more consistent with methods like `.dropna()` and `.fillna()`. Furthermore in all cases where `.isnull()` and `.notnull()` methods are defined, these have additional methods named `.isna()` and `.notna()`, these are included for classes `Categorical`, `Index`, `Series`, and `DataFrame`. (GH15001).

The configuration option `pd.options.mode.use_inf_as_null` is deprecated, and `pd.options.mode.use_inf_as_na` is added as a replacement.

Iteration of Series/Index will now return Python scalars

Previously, when using certain iteration methods for a `Series` with dtype `int` or `float`, you would receive a `numpy` scalar, e.g. a `np.int64`, rather than a Python `int`. Issue (GH10904) corrected this for `Series.tolist()` and `list(Series)`. This change makes all iteration methods consistent, in particular, for `__iter__()` and `.map()`; note that this only affects int/float dtypes. (GH13236, GH13258, GH14216).

```
In [40]: s = pd.Series([1, 2, 3])

In [41]: s
Out[41]:
0    1
```

(continues on next page)

(continued from previous page)

```
1    2
2    3
Length: 3, dtype: int64
```

Previously:

```
In [2]: type(list(s)[0])
Out[2]: numpy.int64
```

New behavior:

```
In [42]: type(list(s)[0])
Out[42]: int
```

Furthermore this will now correctly box the results of iteration for *DataFrame.to_dict()* as well.

```
In [43]: d = {'a': [1], 'b': ['b']}

In [44]: df = pd.DataFrame(d)
```

Previously:

```
In [8]: type(df.to_dict()['a'][0])
Out[8]: numpy.int64
```

New behavior:

```
In [45]: type(df.to_dict()['a'][0])
Out[45]: int
```

Indexing with a Boolean Index

Previously when passing a boolean `Index` to `.loc`, if the index of the `Series/DataFrame` had `boolean` labels, you would get a label based selection, potentially duplicating result labels, rather than a boolean indexing selection (where `True` selects elements), this was inconsistent how a boolean numpy array indexed. The new behavior is to act like a boolean numpy array indexer. (GH17738)

Previous behavior:

```
In [46]: s = pd.Series([1, 2, 3], index=[False, True, False])

In [47]: s
Out[47]:
False    1
True     2
False    3
Length: 3, dtype: int64
```

```
In [59]: s.loc[pd.Index([True, False, True])]
Out[59]:
True     2
False    1
False    3
True     2
dtype: int64
```

Current behavior

```
In [48]: s.loc[pd.Index([True, False, True])]
Out[48]:
False    1
False    3
Length: 2, dtype: int64
```

Furthermore, previously if you had an index that was non-numeric (e.g. strings), then a boolean Index would raise a `KeyError`. This will now be treated as a boolean indexer.

Previously behavior:

```
In [49]: s = pd.Series([1, 2, 3], index=['a', 'b', 'c'])

In [50]: s
Out[50]:
a    1
b    2
c    3
Length: 3, dtype: int64
```

```
In [39]: s.loc[pd.Index([True, False, True])]
KeyError: "None of [Index([True, False, True], dtype='object')] are in the [index]"
```

Current behavior

```
In [51]: s.loc[pd.Index([True, False, True])]
Out[51]:
a    1
c    3
Length: 2, dtype: int64
```

`PeriodIndex` resampling

In previous versions of pandas, resampling a `Series`/`DataFrame` indexed by a `PeriodIndex` returned a `DatetimeIndex` in some cases (GH12884). Resampling to a multiplied frequency now returns a `PeriodIndex` (GH15944). As a minor enhancement, resampling a `PeriodIndex` can now handle `NaT` values (GH13224)

Previous behavior:

```
In [1]: pi = pd.period_range('2017-01', periods=12, freq='M')

In [2]: s = pd.Series(np.arange(12), index=pi)

In [3]: resampled = s.resample('2Q').mean()

In [4]: resampled
Out[4]:
2017-03-31     1.0
2017-09-30     5.5
2018-03-31    10.0
Freq: 2Q-DEC, dtype: float64

In [5]: resampled.index
Out[5]: DatetimeIndex(['2017-03-31', '2017-09-30', '2018-03-31'], dtype=
↪'datetime64[ns]', freq='2Q-DEC')
```

New behavior:

```
In [52]: pi = pd.period_range('2017-01', periods=12, freq='M')

In [53]: s = pd.Series(np.arange(12), index=pi)

In [54]: resampled = s.resample('2Q').mean()

In [55]: resampled
Out[55]:
2017Q1    2.5
2017Q3    8.5
Freq: 2Q-DEC, Length: 2, dtype: float64

In [56]: resampled.index
Out[56]: PeriodIndex(['2017Q1', '2017Q3'], dtype='period[2Q-DEC]', freq='2Q-DEC')
```

Upsampling and calling `.ohlc()` previously returned a `Series`, basically identical to calling `.asfreq()`. OHLC upsampling now returns a DataFrame with columns `open`, `high`, `low` and `close` (GH13083). This is consistent with downsampling and `DatetimeIndex` behavior.

Previous behavior:

```
In [1]: pi = pd.period_range(start='2000-01-01', freq='D', periods=10)

In [2]: s = pd.Series(np.arange(10), index=pi)

In [3]: s.resample('H').ohlc()
Out[3]:
2000-01-01 00:00    0.0
                   ...
2000-01-10 23:00    NaN
Freq: H, Length: 240, dtype: float64

In [4]: s.resample('M').ohlc()
Out[4]:
         open  high  low  close
2000-01     0     9    0      9
```

New behavior:

```
In [57]: pi = pd.period_range(start='2000-01-01', freq='D', periods=10)

In [58]: s = pd.Series(np.arange(10), index=pi)

In [59]: s.resample('H').ohlc()
Out[59]:
                     open  high  low  close
2000-01-01 00:00      0.0   0.0  0.0    0.0
2000-01-01 01:00      NaN   NaN  NaN    NaN
2000-01-01 02:00      NaN   NaN  NaN    NaN
2000-01-01 03:00      NaN   NaN  NaN    NaN
2000-01-01 04:00      NaN   NaN  NaN    NaN
...                   ...   ...  ...    ...
2000-01-10 19:00      NaN   NaN  NaN    NaN
2000-01-10 20:00      NaN   NaN  NaN    NaN
2000-01-10 21:00      NaN   NaN  NaN    NaN
2000-01-10 22:00      NaN   NaN  NaN    NaN
```

(continues on next page)

(continued from previous page)

```
2000-01-10 23:00   NaN   NaN  NaN    NaN

[240 rows x 4 columns]

In [60]: s.resample('M').ohlc()
Out[60]:
         open  high  low  close
2000-01     0     9    0      9

[1 rows x 4 columns]
```

Improved error handling during item assignment in pd.eval

eval() will now raise a `ValueError` when item assignment malfunctions, or inplace operations are specified, but there is no item assignment in the expression (GH16732)

```
In [61]: arr = np.array([1, 2, 3])
```

Previously, if you attempted the following expression, you would get a not very helpful error message:

```
In [3]: pd.eval("a = 1 + 2", target=arr, inplace=True)
...
IndexError: only integers, slices (`:`), ellipsis (`...`), numpy.newaxis (`None`)
and integer or boolean arrays are valid indices
```

This is a very long way of saying numpy arrays don't support string-item indexing. With this change, the error message is now this:

```
In [3]: pd.eval("a = 1 + 2", target=arr, inplace=True)
...
ValueError: Cannot assign expression output to target
```

It also used to be possible to evaluate expressions inplace, even if there was no item assignment:

```
In [4]: pd.eval("1 + 2", target=arr, inplace=True)
Out[4]: 3
```

However, this input does not make much sense because the output is not being assigned to the target. Now, a `ValueError` will be raised when such an input is passed in:

```
In [4]: pd.eval("1 + 2", target=arr, inplace=True)
...
ValueError: Cannot operate inplace if there is no assignment
```

Dtype conversions

Previously assignments, `.where()` and `.fillna()` with a `bool` assignment, would coerce to same the type (e.g. int / float), or raise for datetimelikes. These will now preserve the bools with `object` dtypes. (GH16821).

```
In [62]: s = pd.Series([1, 2, 3])
```

```
In [5]: s[1] = True

In [6]: s
Out[6]:
0    1
1    1
2    3
dtype: int64
```

New behavior

```
In [63]: s[1] = True

In [64]: s
Out[64]:
0       1
1    True
2       3
Length: 3, dtype: object
```

Previously, as assignment to a datetimelike with a non-datetimelike would coerce the non-datetime-like item being assigned (GH14145).

```
In [65]: s = pd.Series([pd.Timestamp('2011-01-01'), pd.Timestamp('2012-01-01')])
```

```
In [1]: s[1] = 1

In [2]: s
Out[2]:
0   2011-01-01 00:00:00.000000000
1   1970-01-01 00:00:00.000000001
dtype: datetime64[ns]
```

These now coerce to `object` dtype.

```
In [66]: s[1] = 1

In [67]: s
Out[67]:
0    2011-01-01 00:00:00
1                      1
Length: 2, dtype: object
```

- Inconsistent behavior in `.where()` with datetimelikes which would raise rather than coerce to `object` (GH16402)
- Bug in assignment against `int64` data with `np.ndarray` with `float64` dtype may keep `int64` dtype (GH14001)

MultiIndex constructor with a single level

The `MultiIndex` constructors no longer squeezes a MultiIndex with all length-one levels down to a regular `Index`. This affects all the `MultiIndex` constructors. (GH17178)

Previous behavior:

```
In [2]: pd.MultiIndex.from_tuples([('a',), ('b',)])
Out[2]: Index(['a', 'b'], dtype='object')
```

Length 1 levels are no longer special-cased. They behave exactly as if you had length 2+ levels, so a *MultiIndex* is always returned from all of the `MultiIndex` constructors:

```
In [68]: pd.MultiIndex.from_tuples([('a',), ('b',)])
Out[68]:
MultiIndex([('a',),
            ('b',)],
           )
```

UTC Localization with Series

Previously, *to_datetime()* did not localize datetime `Series` data when `utc=True` was passed. Now, *to_datetime()* will correctly localize `Series` with a `datetime64[ns, UTC]` dtype to be consistent with how list-like and `Index` data are handled. (GH6415).

Previous behavior

```
In [69]: s = pd.Series(['20130101 00:00:00'] * 3)
```

```
In [12]: pd.to_datetime(s, utc=True)
Out[12]:
0   2013-01-01
1   2013-01-01
2   2013-01-01
dtype: datetime64[ns]
```

New behavior

```
In [70]: pd.to_datetime(s, utc=True)
Out[70]:
0   2013-01-01 00:00:00+00:00
1   2013-01-01 00:00:00+00:00
2   2013-01-01 00:00:00+00:00
Length: 3, dtype: datetime64[ns, UTC]
```

Additionally, DataFrames with datetime columns that were parsed by *read_sql_table()* and *read_sql_query()* will also be localized to UTC only if the original SQL columns were timezone aware datetime columns.

Consistency of range functions

In previous versions, there were some inconsistencies between the various range functions: *date_range()*, *bdate_range()*, *period_range()*, *timedelta_range()*, and *interval_range()*. (GH17471).

One of the inconsistent behaviors occurred when the `start`, `end` and `period` parameters were all specified, potentially leading to ambiguous ranges. When all three parameters were passed, `interval_range` ignored the `period` parameter, `period_range` ignored the `end` parameter, and the other range functions raised. To promote consistency among the range functions, and avoid potentially ambiguous ranges, `interval_range` and `period_range` will now raise when all three parameters are passed.

Previous behavior:

```
In [2]: pd.interval_range(start=0, end=4, periods=6)
Out[2]:
IntervalIndex([(0, 1], (1, 2], (2, 3]]
              closed='right',
              dtype='interval[int64]')

In [3]: pd.period_range(start='2017Q1', end='2017Q4', periods=6, freq='Q')
Out[3]: PeriodIndex(['2017Q1', '2017Q2', '2017Q3', '2017Q4', '2018Q1', '2018Q2'],␣
↪dtype='period[Q-DEC]', freq='Q-DEC')
```

New behavior:

```
In [2]: pd.interval_range(start=0, end=4, periods=6)
---------------------------------------------------------------------------
ValueError: Of the three parameters: start, end, and periods, exactly two must be␣
↪specified

In [3]: pd.period_range(start='2017Q1', end='2017Q4', periods=6, freq='Q')
---------------------------------------------------------------------------
ValueError: Of the three parameters: start, end, and periods, exactly two must be␣
↪specified
```

Additionally, the endpoint parameter `end` was not included in the intervals produced by `interval_range`. However, all other range functions include `end` in their output. To promote consistency among the range functions, `interval_range` will now include `end` as the right endpoint of the final interval, except if `freq` is specified in a way which skips `end`.

Previous behavior:

```
In [4]: pd.interval_range(start=0, end=4)
Out[4]:
IntervalIndex([(0, 1], (1, 2], (2, 3]]
              closed='right',
              dtype='interval[int64]')
```

New behavior:

```
In [71]: pd.interval_range(start=0, end=4)
Out[71]:
IntervalIndex([(0, 1], (1, 2], (2, 3], (3, 4]],
              closed='right',
              dtype='interval[int64]')
```

No automatic Matplotlib converters

Pandas no longer registers our `date`, `time`, `datetime`, `datetime64`, and `Period` converters with matplotlib when pandas is imported. Matplotlib plot methods (`plt.plot`, `ax.plot`, ...), will not nicely format the x-axis for `DatetimeIndex` or `PeriodIndex` values. You must explicitly register these methods:

Pandas built-in `Series.plot` and `DataFrame.plot` *will* register these converters on first-use (GH17710).

Note: This change has been temporarily reverted in pandas 0.21.1, for more details see *here*.

Other API changes

- The Categorical constructor no longer accepts a scalar for the `categories` keyword. (GH16022)
- Accessing a non-existent attribute on a closed `HDFStore` will now raise an `AttributeError` rather than a `ClosedFileError` (GH16301)
- *read_csv()* now issues a `UserWarning` if the `names` parameter contains duplicates (GH17095)
- *read_csv()* now treats `'null'` and `'n/a'` strings as missing values by default (GH16471, GH16078)
- `pandas.HDFStore`'s string representation is now faster and less detailed. For the previous behavior, use `pandas.HDFStore.info()`. (GH16503).
- Compression defaults in HDF stores now follow pytables standards. Default is no compression and if `complib` is missing and `complevel` > 0 `zlib` is used (GH15943)
- `Index.get_indexer_non_unique()` now returns a ndarray indexer rather than an `Index`; this is consistent with `Index.get_indexer()` (GH16819)
- Removed the `@slow` decorator from `pandas._testing`, which caused issues for some downstream packages' test suites. Use `@pytest.mark.slow` instead, which achieves the same thing (GH16850)
- Moved definition of `MergeError` to the `pandas.errors` module.
- The signature of *Series.set_axis()* and *DataFrame.set_axis()* has been changed from `set_axis(axis, labels)` to `set_axis(labels, axis=0)`, for consistency with the rest of the API. The old signature is deprecated and will show a `FutureWarning` (GH14636)
- *Series.argmin()* and *Series.argmax()* will now raise a `TypeError` when used with `object` dtypes, instead of a `ValueError` (GH13595)
- *Period* is now immutable, and will now raise an `AttributeError` when a user tries to assign a new value to the `ordinal` or `freq` attributes (GH17116).
- *to_datetime()* when passed a tz-aware `origin=` kwarg will now raise a more informative `ValueError` rather than a `TypeError` (GH16842)
- *to_datetime()* now raises a `ValueError` when format includes `%W` or `%U` without also including day of the week and calendar year (GH16774)
- Renamed non-functional `index` to `index_col` in *read_stata()* to improve API consistency (GH16342)
- Bug in *DataFrame.drop()* caused boolean labels `False` and `True` to be treated as labels 0 and 1 respectively when dropping indices from a numeric index. This will now raise a ValueError (GH16877)
- 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).

Deprecations

- `DataFrame.from_csv()` and `Series.from_csv()` have been deprecated in favor of *`read_csv()`* (GH4191)
- *`read_excel()`* has deprecated `sheetname` in favor of `sheet_name` for consistency with `.to_excel()` (GH10559).
- *`read_excel()`* has deprecated `parse_cols` in favor of `usecols` for consistency with *`read_csv()`* (GH4988)
- *`read_csv()`* has deprecated the `tupleize_cols` argument. Column tuples will always be converted to a `MultiIndex` (GH17060)
- *`DataFrame.to_csv()`* has deprecated the `tupleize_cols` argument. MultiIndex columns will be always written as rows in the CSV file (GH17060)
- The `convert` parameter has been deprecated in the `.take()` method, as it was not being respected (GH16948)
- `pd.options.html.border` has been deprecated in favor of `pd.options.display.html.border` (GH15793).
- `SeriesGroupBy.nth()` has deprecated `True` in favor of `'all'` for its kwarg `dropna` (GH11038).
- `DataFrame.as_blocks()` is deprecated, as this is exposing the internal implementation (GH17302)
- `pd.TimeGrouper` is deprecated in favor of *`pandas.Grouper`* (GH16747)
- `cdate_range` has been deprecated in favor of *`bdate_range()`*, which has gained `weekmask` and `holidays` parameters for building custom frequency date ranges. See the *documentation* for more details (GH17596)
- passing `categories` or `ordered` kwargs to *`Series.astype()`* is deprecated, in favor of passing a *CategoricalDtype* (GH17636)
- `.get_value` and `.set_value` on `Series`, `DataFrame`, `Panel`, `SparseSeries`, and `SparseDataFrame` are deprecated in favor of using `.iat[]` or `.at[]` accessors (GH15269)
- Passing a non-existent column in `.to_excel(..., columns=)` is deprecated and will raise a `KeyError` in the future (GH17295)
- `raise_on_error` parameter to *`Series.where()`*, *`Series.mask()`*, *`DataFrame.where()`*, *`DataFrame.mask()`* is deprecated, in favor of `errors=` (GH14968)
- Using *`DataFrame.rename_axis()`* and *`Series.rename_axis()`* to alter index or column *labels* is now deprecated in favor of using `.rename`. `rename_axis` may still be used to alter the name of the index or columns (GH17833).
- `reindex_axis()` has been deprecated in favor of *`reindex()`*. See *here* for more (GH17833).

Series.select and DataFrame.select

The `Series.select()` and `DataFrame.select()` methods are deprecated in favor of using `df.loc[labels.map(crit)]` (GH12401)

```
In [72]: df = pd.DataFrame({'A': [1, 2, 3]}, index=['foo', 'bar', 'baz'])
```

```
In [3]: df.select(lambda x: x in ['bar', 'baz'])
FutureWarning: select is deprecated and will be removed in a future release. You can␣
↪use .loc[crit] as a replacement
Out[3]:
     A
bar  2
baz  3
```

```
In [73]: df.loc[df.index.map(lambda x: x in ['bar', 'baz'])]
Out[73]:
     A
bar  2
baz  3

[2 rows x 1 columns]
```

Series.argmax and Series.argmin

The behavior of *Series.argmax()* and *Series.argmin()* have been deprecated in favor of *Series.idxmax()* and *Series.idxmin()*, respectively (GH16830).

For compatibility with NumPy arrays, `pd.Series` implements `argmax` and `argmin`. Since pandas 0.13.0, `argmax` has been an alias for *pandas.Series.idxmax()*, and `argmin` has been an alias for *pandas.Series.idxmin()*. They return the *label* of the maximum or minimum, rather than the *position*.

We've deprecated the current behavior of `Series.argmax` and `Series.argmin`. Using either of these will emit a `FutureWarning`. Use *Series.idxmax()* if you want the label of the maximum. Use `Series.values.argmax()` if you want the position of the maximum. Likewise for the minimum. In a future release `Series.argmax` and `Series.argmin` will return the position of the maximum or minimum.

Removal of prior version deprecations/changes

- *read_excel()* has dropped the `has_index_names` parameter (GH10967)
- The `pd.options.display.height` configuration has been dropped (GH3663)
- The `pd.options.display.line_width` configuration has been dropped (GH2881)
- The `pd.options.display.mpl_style` configuration has been dropped (GH12190)
- `Index` has dropped the `.sym_diff()` method in favor of `.symmetric_difference()` (GH12591)
- `Categorical` has dropped the `.order()` and `.sort()` methods in favor of `.sort_values()` (GH12882)
- *eval()* and *DataFrame.eval()* have changed the default of `inplace` from `None` to `False` (GH11149)
- The function `get_offset_name` has been dropped in favor of the `.freqstr` attribute for an offset (GH11834)
- pandas no longer tests for compatibility with hdf5-files created with pandas < 0.11 (GH17404).

Performance improvements

- Improved performance of instantiating `SparseDataFrame` (GH16773)
- *`Series.dt`* no longer performs frequency inference, yielding a large speedup when accessing the attribute (GH17210)
- Improved performance of *`set_categories()`* by not materializing the values (GH17508)
- *`Timestamp.microsecond`* no longer re-computes on attribute access (GH17331)
- Improved performance of the *`CategoricalIndex`* for data that is already categorical dtype (GH17513)
- Improved performance of `RangeIndex.min()` and `RangeIndex.max()` by using `RangeIndex` properties to perform the computations (GH17607)

Documentation changes

- Several `NaT` method docstrings (e.g. `NaT.ctime()`) were incorrect (GH17327)
- The documentation has had references to versions < v0.17 removed and cleaned up (GH17442, GH17442, GH17404 & GH17504)

Bug fixes

Conversion

- Bug in assignment against datetime-like data with `int` may incorrectly convert to datetime-like (GH14145)
- Bug in assignment against `int64` data with `np.ndarray` with `float64` dtype may keep `int64` dtype (GH14001)
- Fixed the return type of `IntervalIndex.is_non_overlapping_monotonic` to be a Python `bool` for consistency with similar attributes/methods. Previously returned a `numpy.bool_`. (GH17237)
- Bug in `IntervalIndex.is_non_overlapping_monotonic` when intervals are closed on both sides and overlap at a point (GH16560)
- Bug in *`Series.fillna()`* returns frame when `inplace=True` and `value` is dict (GH16156)
- Bug in `Timestamp.weekday_name` returning a UTC-based weekday name when localized to a timezone (GH17354)
- Bug in `Timestamp.replace` when replacing `tzinfo` around DST changes (GH15683)
- Bug in `Timedelta` construction and arithmetic that would not propagate the `Overflow` exception (GH17367)
- Bug in *`astype()`* converting to object dtype when passed extension type classes (`DatetimeTZDtype`, `CategoricalDtype`) rather than instances. Now a `TypeError` is raised when a class is passed (GH17780).
- Bug in *`to_numeric()`* in which elements were not always being coerced to numeric when `errors='coerce'` (GH17007, GH17125)
- Bug in `DataFrame` and `Series` constructors where `range` objects are converted to `int32` dtype on Windows instead of `int64` (GH16804)

Indexing

- When called with a null slice (e.g. `df.iloc[:]`), the `.iloc` and `.loc` indexers return a shallow copy of the original object. Previously they returned the original object. (GH13873).
- When called on an unsorted `MultiIndex`, the `loc` indexer now will raise `UnsortedIndexError` only if proper slicing is used on non-sorted levels (GH16734).
- Fixes regression in 0.20.3 when indexing with a string on a `TimedeltaIndex` (GH16896).
- Fixed `TimedeltaIndex.get_loc()` handling of `np.timedelta64` inputs (GH16909).
- Fix `MultiIndex.sort_index()` ordering when `ascending` argument is a list, but not all levels are specified, or are in a different order (GH16934).
- Fixes bug where indexing with `np.inf` caused an `OverflowError` to be raised (GH16957)
- Bug in reindexing on an empty `CategoricalIndex` (GH16770)
- Fixes `DataFrame.loc` for setting with alignment and tz-aware `DatetimeIndex` (GH16889)
- Avoids `IndexError` when passing an Index or Series to `.iloc` with older numpy (GH17193)
- Allow unicode empty strings as placeholders in multilevel columns in Python 2 (GH17099)
- Bug in `.iloc` when used with inplace addition or assignment and an int indexer on a `MultiIndex` causing the wrong indexes to be read from and written to (GH17148)
- Bug in `.isin()` in which checking membership in empty `Series` objects raised an error (GH16991)
- Bug in `CategoricalIndex` reindexing in which specified indices containing duplicates were not being respected (GH17323)
- Bug in intersection of `RangeIndex` with negative step (GH17296)
- Bug in `IntervalIndex` where performing a scalar lookup fails for included right endpoints of non-overlapping monotonic decreasing indexes (GH16417, GH17271)
- Bug in *`DataFrame.first_valid_index()`* and *`DataFrame.last_valid_index()`* when no valid entry (GH17400)
- Bug in *`Series.rename()`* when called with a callable, incorrectly alters the name of the `Series`, rather than the name of the `Index`. (GH17407)
- Bug in `String.str_get()` raises `IndexError` instead of inserting NaNs when using a negative index. (GH17704)

I/O

- Bug in *`read_hdf()`* when reading a timezone aware index from `fixed` format HDFStore (GH17618)
- Bug in *`read_csv()`* in which columns were not being thoroughly de-duplicated (GH17060)
- Bug in *`read_csv()`* in which specified column names were not being thoroughly de-duplicated (GH17095)
- Bug in *`read_csv()`* in which non integer values for the header argument generated an unhelpful / unrelated error message (GH16338)
- Bug in *`read_csv()`* in which memory management issues in exception handling, under certain conditions, would cause the interpreter to segfault (GH14696, GH16798).
- Bug in *`read_csv()`* when called with `low_memory=False` in which a CSV with at least one column > 2GB in size would incorrectly raise a `MemoryError` (GH16798).