(continued from previous page)

```
0 -0.334077  0.002118
1  0.036142 -2.074978
2 -0.720589  0.887163
3  0.859588 -0.636524

[4 rows x 2 columns]

In [47]: df.groupby('c', sort=False).nth(1)
Out[47]:
          a         b
c
2 -0.720589  0.887163
3  0.859588 -0.636524
0 -0.334077  0.002118
1  0.036142 -2.074978

[4 rows x 2 columns]
```

numpy function compatibility

Compatibility between pandas array-like methods (e.g. `sum` and `take`) and their `numpy` counterparts has been greatly increased by augmenting the signatures of the `pandas` methods so as to accept arguments that can be passed in from `numpy`, even if they are not necessarily used in the `pandas` implementation (GH12644, GH12638, GH12687)

- `.searchsorted()` for `Index` and `TimedeltaIndex` now accept a `sorter` argument to maintain compatibility with numpy's `searchsorted` function (GH12238)
- Bug in numpy compatibility of `np.round()` on a `Series` (GH12600)

An example of this signature augmentation is illustrated below:

```
sp = pd.SparseDataFrame([1, 2, 3])
sp
```

Previous behaviour:

```
In [2]: np.cumsum(sp, axis=0)
...
TypeError: cumsum() takes at most 2 arguments (4 given)
```

New behaviour:

```
np.cumsum(sp, axis=0)
```

Using `.apply` on groupby resampling

Using `apply` on resampling groupby operations (using a `pd.TimeGrouper`) now has the same output types as similar `apply` calls on other groupby operations. (GH11742).

```
In [48]: df = pd.DataFrame({'date': pd.to_datetime(['10/10/2000', '11/10/2000']),
   ....:                    'value': [10, 13]})
   ....:
```

(continues on next page)

(continued from previous page)

```
In [49]: df
Out[49]: 
        date  value
0 2000-10-10     10
1 2000-11-10     13

[2 rows x 2 columns]
```

Previous behavior:

```
In [1]: df.groupby(pd.TimeGrouper(key='date',
   ...:                           freq='M')).apply(lambda x: x.value.sum())
Out[1]:
...
TypeError: cannot concatenate a non-NDFrame object

# Output is a Series
In [2]: df.groupby(pd.TimeGrouper(key='date',
   ...:                           freq='M')).apply(lambda x: x[['value']].sum())
Out[2]:
date
2000-10-31  value    10
2000-11-30  value    13
dtype: int64
```

New behavior:

```
# Output is a Series
In [55]: df.groupby(pd.TimeGrouper(key='date',
    ...:                           freq='M')).apply(lambda x: x.value.sum())
Out[55]: 
date
2000-10-31    10
2000-11-30    13
Freq: M, dtype: int64

# Output is a DataFrame
In [56]: df.groupby(pd.TimeGrouper(key='date',
    ...:                           freq='M')).apply(lambda x: x[['value']].sum())
Out[56]: 
            value
date             
2000-10-31     10
2000-11-30     13
```

Changes in read_csv exceptions

In order to standardize the read_csv API for both the c and python engines, both will now raise an EmptyDataError, a subclass of ValueError, in response to empty columns or header (GH12493, GH12506)

Previous behaviour:

```
In [1]: import io

In [2]: df = pd.read_csv(io.StringIO(''), engine='c')
```

(continues on next page)

(continued from previous page)

```
...
ValueError: No columns to parse from file

In [3]: df = pd.read_csv(io.StringIO(''), engine='python')
...
StopIteration
```

New behaviour:

```
In [1]: df = pd.read_csv(io.StringIO(''), engine='c')
...
pandas.io.common.EmptyDataError: No columns to parse from file

In [2]: df = pd.read_csv(io.StringIO(''), engine='python')
...
pandas.io.common.EmptyDataError: No columns to parse from file
```

In addition to this error change, several others have been made as well:

- CParserError now sub-classes ValueError instead of just a Exception (GH12551)
- A CParserError is now raised instead of a generic Exception in read_csv when the c engine cannot parse a column (GH12506)
- A ValueError is now raised instead of a generic Exception in read_csv when the c engine encounters a NaN value in an integer column (GH12506)
- A ValueError is now raised instead of a generic Exception in read_csv when true_values is specified, and the c engine encounters an element in a column containing unencodable bytes (GH12506)
- pandas.parser.OverflowError exception has been removed and has been replaced with Python's built-in OverflowError exception (GH12506)
- pd.read_csv() no longer allows a combination of strings and integers for the usecols parameter (GH12678)

to_datetime error changes

Bugs in pd.to_datetime() when passing a unit with convertible entries and errors='coerce' or non-convertible with errors='ignore'. Furthermore, an OutOfBoundsDateime exception will be raised when an out-of-range value is encountered for that unit when errors='raise'. (GH11758, GH13052, GH13059)

Previous behaviour:

```
In [27]: pd.to_datetime(1420043460, unit='s', errors='coerce')
Out[27]: NaT

In [28]: pd.to_datetime(11111111, unit='D', errors='ignore')
OverflowError: Python int too large to convert to C long

In [29]: pd.to_datetime(11111111, unit='D', errors='raise')
OverflowError: Python int too large to convert to C long
```

New behaviour:

```
In [2]: pd.to_datetime(1420043460, unit='s', errors='coerce')
Out[2]: Timestamp('2014-12-31 16:31:00')
```

(continues on next page)

(continued from previous page)

```
In [3]: pd.to_datetime(11111111, unit='D', errors='ignore')
Out[3]: 11111111

In [4]: pd.to_datetime(11111111, unit='D', errors='raise')
OutOfBoundsDatetime: cannot convert input with unit 'D'
```

Other API changes

- `.swaplevel()` for `Series`, `DataFrame`, `Panel`, and `MultiIndex` now features defaults for its first two parameters `i` and `j` that swap the two innermost levels of the index. (GH12934)
- `.searchsorted()` for `Index` and `TimedeltaIndex` now accept a `sorter` argument to maintain compatibility with numpy's `searchsorted` function (GH12238)
- `Period` and `PeriodIndex` now raises `IncompatibleFrequency` error which inherits `ValueError` rather than raw `ValueError` (GH12615)
- `Series.apply` for category dtype now applies the passed function to each of the `.categories` (and not the `.codes`), and returns a `category` dtype if possible (GH12473)
- `read_csv` will now raise a `TypeError` if `parse_dates` is neither a boolean, list, or dictionary (matches the doc-string) (GH5636)
- The default for `.query()/.eval()` is now `engine=None`, which will use `numexpr` if it's installed; otherwise it will fallback to the `python` engine. This mimics the pre-0.18.1 behavior if `numexpr` is installed (and which, previously, if numexpr was not installed, `.query()/.eval()` would raise). (GH12749)
- `pd.show_versions()` now includes `pandas_datareader` version (GH12740)
- Provide a proper `__name__` and `__qualname__` attributes for generic functions (GH12021)
- `pd.concat(ignore_index=True)` now uses `RangeIndex` as default (GH12695)
- `pd.merge()` and `DataFrame.join()` will show a `UserWarning` when merging/joining a single- with a multi-leveled dataframe (GH9455, GH12219)
- Compat with `scipy` > 0.17 for deprecated `piecewise_polynomial` interpolation method; support for the replacement `from_derivatives` method (GH12887)

Deprecations

- The method name `Index.sym_diff()` is deprecated and can be replaced by `Index.symmetric_difference()` (GH12591)
- The method name `Categorical.sort()` is deprecated in favor of `Categorical.sort_values()` (GH12882)

Performance improvements

- Improved speed of SAS reader (GH12656, GH12961)
- Performance improvements in `.groupby(..).cumcount()` (GH11039)
- Improved memory usage in `pd.read_csv()` when using `skiprows=an_integer` (GH13005)
- Improved performance of `DataFrame.to_sql` when checking case sensitivity for tables. Now only checks if table has been created correctly when table name is not lower case. (GH12876)
- Improved performance of `Period` construction and time series plotting (GH12903, GH11831).
- Improved performance of `.str.encode()` and `.str.decode()` methods (GH13008)
- Improved performance of `to_numeric` if input is numeric dtype (GH12777)
- Improved performance of sparse arithmetic with `IntIndex` (GH13036)

Bug fixes

- `usecols` parameter in `pd.read_csv` is now respected even when the lines of a CSV file are not even (GH12203)
- Bug in `groupby.transform(..)` when `axis=1` is specified with a non-monotonic ordered index (GH12713)
- Bug in `Period` and `PeriodIndex` creation raises `KeyError` if `freq="Minute"` is specified. Note that "Minute" freq is deprecated in v0.17.0, and recommended to use `freq="T"` instead (GH11854)
- Bug in `.resample(...).count()` with a `PeriodIndex` always raising a `TypeError` (GH12774)
- Bug in `.resample(...)` with a `PeriodIndex` casting to a `DatetimeIndex` when empty (GH12868)
- Bug in `.resample(...)` with a `PeriodIndex` when resampling to an existing frequency (GH12770)
- Bug in printing data which contains `Period` with different `freq` raises `ValueError` (GH12615)
- Bug in `Series` construction with `Categorical` and `dtype='category'` is specified (GH12574)
- Bugs in concatenation with a coercible dtype was too aggressive, resulting in different dtypes in output formatting when an object was longer than `display.max_rows` (GH12411, GH12045, GH11594, GH10571, GH12211)
- Bug in `float_format` option with option not being validated as a callable. (GH12706)
- Bug in `GroupBy.filter` when `dropna=False` and no groups fulfilled the criteria (GH12768)
- Bug in `__name__` of `.cum*` functions (GH12021)
- Bug in `.astype()` of a `Float64Inde/Int64Index` to an `Int64Index` (GH12881)
- Bug in round tripping an integer based index in `.to_json()/.read_json()` when `orient='index'` (the default) (GH12866)
- Bug in plotting `Categorical` dtypes cause error when attempting stacked bar plot (GH13019)
- Compat with >= `numpy` 1.11 for `NaT` comparisons (GH12969)
- Bug in `.drop()` with a non-unique `MultiIndex`. (GH12701)
- Bug in `.concat` of datetime tz-aware and naive DataFrames (GH12467)
- Bug in correctly raising a `ValueError` in `.resample(..).fillna(..)` when passing a non-string (GH12952)

- Bug fixes in various encoding and header processing issues in `pd.read_sas()` (GH12659, GH12654, GH12647, GH12809)
- Bug in `pd.crosstab()` where would silently ignore `aggfunc` if `values=None` (GH12569).
- Potential segfault in `DataFrame.to_json` when serialising `datetime.time` (GH11473).
- Potential segfault in `DataFrame.to_json` when attempting to serialise 0d array (GH11299).
- Segfault in `to_json` when attempting to serialise a `DataFrame` or `Series` with non-ndarray values; now supports serialization of `category`, `sparse`, and `datetime64[ns, tz]` dtypes (GH10778).
- Bug in `DataFrame.to_json` with unsupported dtype not passed to default handler (GH12554).
- Bug in `.align` not returning the sub-class (GH12983)
- Bug in aligning a `Series` with a `DataFrame` (GH13037)
- Bug in `ABCPanel` in which `Panel4D` was not being considered as a valid instance of this generic type (GH12810)
- Bug in consistency of `.name` on `.groupby(..).apply(..)` cases (GH12363)
- Bug in `Timestamp.__repr__` that caused `pprint` to fail in nested structures (GH12622)
- Bug in `Timedelta.min` and `Timedelta.max`, the properties now report the true minimum/maximum `timedeltas` as recognized by pandas. See the *documentation*. (GH12727)
- Bug in `.quantile()` with interpolation may coerce to `float` unexpectedly (GH12772)
- Bug in `.quantile()` with empty `Series` may return scalar rather than empty `Series` (GH12772)
- Bug in `.loc` with out-of-bounds in a large indexer would raise `IndexError` rather than `KeyError` (GH12527)
- Bug in resampling when using a `TimedeltaIndex` and `.asfreq()`, would previously not include the final fencepost (GH12926)
- Bug in equality testing with a `Categorical` in a `DataFrame` (GH12564)
- Bug in `GroupBy.first()`, `.last()` returns incorrect row when `TimeGrouper` is used (GH7453)
- Bug in `pd.read_csv()` with the `c` engine when specifying `skiprows` with newlines in quoted items (GH10911, GH12775)
- Bug in `DataFrame` timezone lost when assigning tz-aware datetime `Series` with alignment (GH12981)
- Bug in `.value_counts()` when `normalize=True` and `dropna=True` where nulls still contributed to the normalized count (GH12558)
- Bug in `Series.value_counts()` loses name if its dtype is `category` (GH12835)
- Bug in `Series.value_counts()` loses timezone info (GH12835)
- Bug in `Series.value_counts(normalize=True)` with `Categorical` raises `UnboundLocalError` (GH12835)
- Bug in `Panel.fillna()` ignoring `inplace=True` (GH12633)
- Bug in `pd.read_csv()` when specifying `names`, `usecols`, and `parse_dates` simultaneously with the `c` engine (GH9755)
- Bug in `pd.read_csv()` when specifying `delim_whitespace=True` and `lineterminator` simultaneously with the `c` engine (GH12912)
- Bug in `Series.rename`, `DataFrame.rename` and `DataFrame.rename_axis` not treating `Series` as mappings to relabel (GH12623).

- Clean in .rolling.min and .rolling.max to enhance dtype handling (GH12373)
- Bug in groupby where complex types are coerced to float (GH12902)
- Bug in Series.map raises TypeError if its dtype is category or tz-aware datetime (GH12473)
- Bugs on 32bit platforms for some test comparisons (GH12972)
- Bug in index coercion when falling back from RangeIndex construction (GH12893)
- Better error message in window functions when invalid argument (e.g. a float window) is passed (GH12669)
- Bug in slicing subclassed DataFrame defined to return subclassed Series may return normal Series (GH11559)
- Bug in .str accessor methods may raise ValueError if input has name and the result is DataFrame or MultiIndex (GH12617)
- Bug in DataFrame.last_valid_index() and DataFrame.first_valid_index() on empty frames (GH12800)
- Bug in CategoricalIndex.get_loc returns different result from regular Index (GH12531)
- Bug in PeriodIndex.resample where name not propagated (GH12769)
- Bug in date_range closed keyword and timezones (GH12684).
- Bug in pd.concat raises AttributeError when input data contains tz-aware datetime and timedelta (GH12620)
- Bug in pd.concat did not handle empty Series properly (GH11082)
- Bug in .plot.bar alignment when width is specified with int (GH12979)
- Bug in fill_value is ignored if the argument to a binary operator is a constant (GH12723)
- Bug in pd.read_html() when using bs4 flavor and parsing table with a header and only one column (GH9178)
- Bug in .pivot_table when margins=True and dropna=True where nulls still contributed to margin count (GH12577)
- Bug in .pivot_table when dropna=False where table index/column names disappear (GH12133)
- Bug in pd.crosstab() when margins=True and dropna=False which raised (GH12642)
- Bug in Series.name when name attribute can be a hashable type (GH12610)
- Bug in .describe() resets categorical columns information (GH11558)
- Bug where loffset argument was not applied when calling resample().count() on a timeseries (GH12725)
- pd.read_excel() now accepts column names associated with keyword argument names (GH12870)
- Bug in pd.to_numeric() with Index returns np.ndarray, rather than Index (GH12777)
- Bug in pd.to_numeric() with datetime-like may raise TypeError (GH12777)
- Bug in pd.to_numeric() with scalar raises ValueError (GH12777)

Contributors

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

- Andrew Fiore-Gartland +
- Bastiaan +
- Benoît Vinot +
- Brandon Rhodes +
- DaCoEx +
- Drew Fustin +
- Ernesto Freitas +
- Filip Ter +
- Gregory Livschitz +
- Gábor Lipták
- Hassan Kibirige +
- Iblis Lin
- Israel Saeta Pérez +
- Jason Wolosonovich +
- Jeff Reback
- Joe Jevnik
- Joris Van den Bossche
- Joshua Storck +
- Ka Wo Chen
- Kerby Shedden
- Kieran O'Mahony
- Leif Walsh +
- Mahmoud Lababidi +
- Maoyuan Liu +
- Mark Roth +
- Matt Wittmann
- MaxU +
- Maximilian Roos
- Michael Droettboom +
- Nick Eubank
- Nicolas Bonnotte
- OXPHOS +
- Pauli Virtanen +

- Peter Waller +
- Pietro Battiston
- Prabhjot Singh +
- Robin Wilson
- Roger Thomas +
- Sebastian Bank
- Stephen Hoover
- Tim Hopper +
- Tom Augspurger
- WANG Aiyong
- Wes Turner
- Winand +
- Xbar +
- Yan Facai +
- adneu +
- ajenkins-cargometrics +
- behzad nouri
- chinskiy +
- gfyoung
- jeps-journal +
- jonaslb +
- kotrfa +
- nileracecrew +
- onesandzeroes
- rs2 +
- sinhrks
- tsdlovell +

5.9.2 v0.18.0 (March 13, 2016)

This is a major release from 0.17.1 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.

> **Warning:** pandas >= 0.18.0 no longer supports compatibility with Python version 2.6 and 3.3 (GH7718, GH11273)

> **Warning:** `numexpr` version 2.4.4 will now show a warning and not be used as a computation back-end for pandas because of some buggy behavior. This does not affect other versions (>= 2.1 and >= 2.4.6). (GH12489)

Highlights include:

- Moving and expanding window functions are now methods on Series and DataFrame, similar to `.groupby`, see *here*.
- Adding support for a `RangeIndex` as a specialized form of the `Int64Index` for memory savings, see *here*.
- API breaking change to the `.resample` method to make it more `.groupby` like, see *here*.
- Removal of support for positional indexing with floats, which was deprecated since 0.14.0. This will now raise a `TypeError`, see *here*.
- The `.to_xarray()` function has been added for compatibility with the xarray package, see *here*.
- The `read_sas` function has been enhanced to read `sas7bdat` files, see *here*.
- Addition of the *.str.extractall() method*, and API changes to the *.str.extract() method* and *.str.cat() method*.
- `pd.test()` top-level nose test runner is available (GH4327).

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

What's new in v0.18.0

New features

Window functions are now methods

Window functions have been refactored to be methods on `Series/DataFrame` objects, rather than top-level functions, which are now deprecated. This allows these window-type functions, to have a similar API to that of `.groupby`. See the full documentation *here* (GH11603, GH12373)

```
In [1]: np.random.seed(1234)

In [2]: df = pd.DataFrame({'A': range(10), 'B': np.random.randn(10)})

In [3]: df
Out[3]:
   A         B
0  0  0.471435
1  1 -1.190976
2  2  1.432707
3  3 -0.312652
4  4 -0.720589
5  5  0.887163
6  6  0.859588
7  7 -0.636524
8  8  0.015696
9  9 -2.242685

[10 rows x 2 columns]
```

Previous behavior:

```
In [8]: pd.rolling_mean(df, window=3)
        FutureWarning: pd.rolling_mean is deprecated for DataFrame and will be␣
↪removed in a future version, replace with
                       DataFrame.rolling(window=3,center=False).mean()
Out[8]:
```

(continues on next page)

(continued from previous page)

```
    A         B
0 NaN       NaN
1 NaN       NaN
2   1  0.237722
3   2 -0.023640
4   3  0.133155
5   4 -0.048693
6   5  0.342054
7   6  0.370076
8   7  0.079587
9   8 -0.954504
```

New behavior:

```
In [4]: r = df.rolling(window=3)
```

These show a descriptive repr

```
In [5]: r
Out[5]: Rolling [window=3,center=False,axis=0]
```

with tab-completion of available methods and properties.

```
In [9]: r.<TAB>  # noqa E225, E999
r.A              r.agg            r.apply          r.count          r.exclusions    r.max            r.
↪median         r.name           r.skew           r.sum
r.B              r.aggregate      r.corr           r.cov            r.kurt          r.mean           r.
↪min            r.quantile       r.std            r.var
```

The methods operate on the `Rolling` object itself

```
In [6]: r.mean()
Out[6]:
     A         B
0  NaN       NaN
1  NaN       NaN
2  1.0  0.237722
3  2.0 -0.023640
4  3.0  0.133155
5  4.0 -0.048693
6  5.0  0.342054
7  6.0  0.370076
8  7.0  0.079587
9  8.0 -0.954504

[10 rows x 2 columns]
```

They provide getitem accessors

```
In [7]: r['A'].mean()
Out[7]:
0    NaN
1    NaN
2    1.0
3    2.0
4    3.0
```

(continues on next page)

(continued from previous page)

```
5    4.0
6    5.0
7    6.0
8    7.0
9    8.0
Name: A, Length: 10, dtype: float64
```

And multiple aggregations

```
In [8]: r.agg({'A': ['mean', 'std'],
   ...:        'B': ['mean', 'std']})
   ...:
Out[8]:
     A              B
  mean  std      mean       std
0  NaN  NaN       NaN       NaN
1  NaN  NaN       NaN       NaN
2  1.0  1.0  0.237722  1.327364
3  2.0  1.0 -0.023640  1.335505
4  3.0  1.0  0.133155  1.143778
5  4.0  1.0 -0.048693  0.835747
6  5.0  1.0  0.342054  0.920379
7  6.0  1.0  0.370076  0.871850
8  7.0  1.0  0.079587  0.750099
9  8.0  1.0 -0.954504  1.162285

[10 rows x 4 columns]
```

Changes to rename

`Series.rename` and `NDFrame.rename_axis` can now take a scalar or list-like argument for altering the Series or axis *name*, in addition to their old behaviors of altering labels. (GH9494, GH11965)

```
In [9]: s = pd.Series(np.random.randn(5))

In [10]: s.rename('newname')
Out[10]:
0    1.150036
1    0.991946
2    0.953324
3   -2.021255
4   -0.334077
Name: newname, Length: 5, dtype: float64
```

```
In [11]: df = pd.DataFrame(np.random.randn(5, 2))

In [12]: (df.rename_axis("indexname")
   ....:    .rename_axis("columns_name", axis="columns"))
   ....:
Out[12]:
columns_name         0         1
indexname
0             0.002118  0.405453
1             0.289092  1.321158
```

(continues on next page)

(continued from previous page)

```
2            -1.546906 -0.202646
3            -0.655969  0.193421
4             0.553439  1.318152

[5 rows x 2 columns]
```

The new functionality works well in method chains. Previously these methods only accepted functions or dicts mapping a *label* to a new label. This continues to work as before for function or dict-like values.

Range index

A `RangeIndex` has been added to the `Int64Index` sub-classes to support a memory saving alternative for common use cases. This has a similar implementation to the python `range` object (`xrange` in python 2), in that it only stores the start, stop, and step values for the index. It will transparently interact with the user API, converting to `Int64Index` if needed.

This will now be the default constructed index for `NDFrame` objects, rather than previous an `Int64Index`. (GH939, GH12070, GH12071, GH12109, GH12888)

Previous behavior:

```
In [3]: s = pd.Series(range(1000))

In [4]: s.index
Out[4]:
Int64Index([  0,   1,   2,   3,   4,   5,   6,   7,   8,   9,
            ...
            990, 991, 992, 993, 994, 995, 996, 997, 998, 999], dtype='int64',␣
↪length=1000)

In [6]: s.index.nbytes
Out[6]: 8000
```

New behavior:

```
In [13]: s = pd.Series(range(1000))

In [14]: s.index
Out[14]: RangeIndex(start=0, stop=1000, step=1)

In [15]: s.index.nbytes
Out[15]: 128
```

Changes to str.extract

The *.str.extract* method takes a regular expression with capture groups, finds the first match in each subject string, and returns the contents of the capture groups (GH11386).

In v0.18.0, the `expand` argument was added to `extract`.

- `expand=False`: it returns a `Series`, `Index`, or `DataFrame`, depending on the subject and regular expression pattern (same behavior as pre-0.18.0).
- `expand=True`: it always returns a `DataFrame`, which is more consistent and less confusing from the perspective of a user.

Currently the default is `expand=None` which gives a `FutureWarning` and uses `expand=False`. To avoid this warning, please explicitly specify `expand`.

```
In [1]: pd.Series(['a1', 'b2', 'c3']).str.extract(r'[ab](\d)', expand=None)
FutureWarning: currently extract(expand=None) means expand=False (return Index/Series/
↪DataFrame)
but in a future version of pandas this will be changed to expand=True (return␣
↪DataFrame)

Out[1]:
0      1
1      2
2    NaN
dtype: object
```

Extracting a regular expression with one group returns a Series if `expand=False`.

```
In [16]: pd.Series(['a1', 'b2', 'c3']).str.extract(r'[ab](\d)', expand=False)
Out[16]:
0      1
1      2
2    NaN
Length: 3, dtype: object
```

It returns a `DataFrame` with one column if `expand=True`.

```
In [17]: pd.Series(['a1', 'b2', 'c3']).str.extract(r'[ab](\d)', expand=True)
Out[17]:
     0
0    1
1    2
2  NaN

[3 rows x 1 columns]
```

Calling on an `Index` with a regex with exactly one capture group returns an `Index` if `expand=False`.

```
In [18]: s = pd.Series(["a1", "b2", "c3"], ["A11", "B22", "C33"])

In [19]: s.index
Out[19]: Index(['A11', 'B22', 'C33'], dtype='object')

In [20]: s.index.str.extract("(?P<letter>[a-zA-Z])", expand=False)
Out[20]: Index(['A', 'B', 'C'], dtype='object', name='letter')
```

It returns a `DataFrame` with one column if `expand=True`.

```
In [21]: s.index.str.extract("(?P<letter>[a-zA-Z])", expand=True)
Out[21]:
  letter
0      A
1      B
2      C

[3 rows x 1 columns]
```

Calling on an `Index` with a regex with more than one capture group raises `ValueError` if `expand=False`.

```
>>> s.index.str.extract("(?P<letter>[a-zA-Z])([0-9]+)", expand=False)
ValueError: only one regex group is supported with Index
```

It returns a `DataFrame` if `expand=True`.

```
In [22]: s.index.str.extract("(?P<letter>[a-zA-Z])([0-9]+)", expand=True)
Out[22]:
  letter   1
0      A  11
1      B  22
2      C  33

[3 rows x 2 columns]
```

In summary, `extract(expand=True)` always returns a `DataFrame` with a row for every subject string, and a column for every capture group.

Addition of str.extractall

The *.str.extractall* method was added (GH11386). Unlike `extract`, which returns only the first match.

```
In [23]: s = pd.Series(["a1a2", "b1", "c1"], ["A", "B", "C"])

In [24]: s
Out[24]:
A    a1a2
B      b1
C      c1
Length: 3, dtype: object

In [25]: s.str.extract(r"(?P<letter>[ab])(?P<digit>\d)", expand=False)
Out[25]:
  letter digit
A      a     1
B      b     1
C    NaN   NaN

[3 rows x 2 columns]
```

The `extractall` method returns all matches.

```
In [26]: s.str.extractall(r"(?P<letter>[ab])(?P<digit>\d)")
Out[26]:
        letter digit
  match
A 0          a     1
  1          a     2
B 0          b     1

[3 rows x 2 columns]
```

Changes to str.cat

The method `.str.cat()` concatenates the members of a `Series`. Before, if `NaN` values were present in the Series, calling `.str.cat()` on it would return `NaN`, unlike the rest of the `Series.str.*` API. This behavior has been amended to ignore `NaN` values by default. (GH11435).

A new, friendlier `ValueError` is added to protect against the mistake of supplying the `sep` as an arg, rather than as a kwarg. (GH11334).

```
In [27]: pd.Series(['a', 'b', np.nan, 'c']).str.cat(sep=' ')
Out[27]: 'a b c'

In [28]: pd.Series(['a', 'b', np.nan, 'c']).str.cat(sep=' ', na_rep='?')
Out[28]: 'a b ? c'
```

```
In [2]: pd.Series(['a', 'b', np.nan, 'c']).str.cat(' ')
ValueError: Did you mean to supply a `sep` keyword?
```

Datetimelike rounding

`DatetimeIndex`, `Timestamp`, `TimedeltaIndex`, `Timedelta` have gained the `.round()`, `.floor()` and `.ceil()` method for datetimelike rounding, flooring and ceiling. (GH4314, GH11963)

Naive datetimes

```
In [29]: dr = pd.date_range('20130101 09:12:56.1234', periods=3)

In [30]: dr
Out[30]:
DatetimeIndex(['2013-01-01 09:12:56.123400', '2013-01-02 09:12:56.123400',
               '2013-01-03 09:12:56.123400'],
              dtype='datetime64[ns]', freq='D')

In [31]: dr.round('s')
Out[31]:
DatetimeIndex(['2013-01-01 09:12:56', '2013-01-02 09:12:56',
               '2013-01-03 09:12:56'],
              dtype='datetime64[ns]', freq=None)

# Timestamp scalar
In [32]: dr[0]
Out[32]: Timestamp('2013-01-01 09:12:56.123400', freq='D')

In [33]: dr[0].round('10s')
Out[33]: Timestamp('2013-01-01 09:13:00')
```

Tz-aware are rounded, floored and ceiled in local times

```
In [34]: dr = dr.tz_localize('US/Eastern')

In [35]: dr
Out[35]:
DatetimeIndex(['2013-01-01 09:12:56.123400-05:00',
               '2013-01-02 09:12:56.123400-05:00',
               '2013-01-03 09:12:56.123400-05:00'],
              dtype='datetime64[ns, US/Eastern]', freq='D')
```

(continues on next page)

(continued from previous page)

```
In [36]: dr.round('s')
Out[36]:
DatetimeIndex(['2013-01-01 09:12:56-05:00', '2013-01-02 09:12:56-05:00',
               '2013-01-03 09:12:56-05:00'],
              dtype='datetime64[ns, US/Eastern]', freq=None)
```

Timedeltas

```
In [37]: t = pd.timedelta_range('1 days 2 hr 13 min 45 us', periods=3, freq='d')

In [38]: t
Out[38]:
TimedeltaIndex(['1 days 02:13:00.000045', '2 days 02:13:00.000045',
                '3 days 02:13:00.000045'],
               dtype='timedelta64[ns]', freq='D')

In [39]: t.round('10min')
Out[39]: TimedeltaIndex(['1 days 02:10:00', '2 days 02:10:00', '3 days 02:10:00'],␣
↪dtype='timedelta64[ns]', freq=None)

# Timedelta scalar
In [40]: t[0]
Out[40]: Timedelta('1 days 02:13:00.000045')

In [41]: t[0].round('2h')
Out[41]: Timedelta('1 days 02:00:00')
```

In addition, `.round()`, `.floor()` and `.ceil()` will be available through the `.dt` accessor of `Series`.

```
In [42]: s = pd.Series(dr)

In [43]: s
Out[43]:
0   2013-01-01 09:12:56.123400-05:00
1   2013-01-02 09:12:56.123400-05:00
2   2013-01-03 09:12:56.123400-05:00
Length: 3, dtype: datetime64[ns, US/Eastern]

In [44]: s.dt.round('D')
Out[44]:
0   2013-01-01 00:00:00-05:00
1   2013-01-02 00:00:00-05:00
2   2013-01-03 00:00:00-05:00
Length: 3, dtype: datetime64[ns, US/Eastern]
```

Formatting of integers in FloatIndex

Integers in `FloatIndex`, e.g. 1., are now formatted with a decimal point and a `0` digit, e.g. `1.0` (GH11713) This change not only affects the display to the console, but also the output of IO methods like `.to_csv` or `.to_html`.

Previous behavior:

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

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

In [4]: s.index
Out[4]: Float64Index([0.0, 1.0, 2.0], dtype='float64')

In [5]: print(s.to_csv(path=None))
0,1
1,2
2,3
```

New behavior:

```
In [45]: s = pd.Series([1, 2, 3], index=np.arange(3.))

In [46]: s
Out[46]:
0.0    1
1.0    2
2.0    3
Length: 3, dtype: int64

In [47]: s.index
Out[47]: Float64Index([0.0, 1.0, 2.0], dtype='float64')

In [48]: print(s.to_csv(path_or_buf=None, header=False))
0.0,1
1.0,2
2.0,3
```

Changes to dtype assignment behaviors

When a DataFrame's slice is updated with a new slice of the same dtype, the dtype of the DataFrame will now remain the same. (GH10503)

Previous behavior:

```
In [5]: df = pd.DataFrame({'a': [0, 1, 1],
                           'b': pd.Series([100, 200, 300], dtype='uint32')})

In [7]: df.dtypes
Out[7]:
a     int64
```

(continues on next page)

(continued from previous page)

```
b    uint32
dtype: object

In [8]: ix = df['a'] == 1

In [9]: df.loc[ix, 'b'] = df.loc[ix, 'b']

In [11]: df.dtypes
Out[11]:
a    int64
b    int64
dtype: object
```

New behavior:

```
In [49]: df = pd.DataFrame({'a': [0, 1, 1],
   ....:                    'b': pd.Series([100, 200, 300], dtype='uint32')})
   ....:

In [50]: df.dtypes
Out[50]:
a      int64
b     uint32
Length: 2, dtype: object

In [51]: ix = df['a'] == 1

In [52]: df.loc[ix, 'b'] = df.loc[ix, 'b']

In [53]: df.dtypes
Out[53]:
a      int64
b     uint32
Length: 2, dtype: object
```

When a DataFrame's integer slice is partially updated with a new slice of floats that could potentially be down-casted to integer without losing precision, the dtype of the slice will be set to float instead of integer.

Previous behavior:

```
In [4]: df = pd.DataFrame(np.array(range(1,10)).reshape(3,3),
                          columns=list('abc'),
                          index=[[4,4,8], [8,10,12]])

In [5]: df
Out[5]:
      a  b  c
4 8   1  2  3
  10  4  5  6
8 12  7  8  9

In [7]: df.ix[4, 'c'] = np.array([0., 1.])

In [8]: df
Out[8]:
      a  b  c
4 8   1  2  0
```

(continues on next page)

(continued from previous page)

```
  10  4  5  1
8 12  7  8  9
```

New behavior:

```
In [54]: df = pd.DataFrame(np.array(range(1,10)).reshape(3,3),
   ....:                   columns=list('abc'),
   ....:                   index=[[4,4,8], [8,10,12]])
   ....:

In [55]: df
Out[55]:
      a  b  c
4 8   1  2  3
  10  4  5  6
8 12  7  8  9

[3 rows x 3 columns]

In [56]: df.loc[4, 'c'] = np.array([0., 1.])

In [57]: df
Out[57]:
      a  b    c
4 8   1  2  0.0
  10  4  5  1.0
8 12  7  8  9.0

[3 rows x 3 columns]
```

to_xarray

In a future version of pandas, we will be deprecating `Panel` and other > 2 ndim objects. In order to provide for continuity, all `NDFrame` objects have gained the `.to_xarray()` method in order to convert to `xarray` objects, which has a pandas-like interface for > 2 ndim. (GH11972)

See the xarray full-documentation here.

```
In [1]: p = Panel(np.arange(2*3*4).reshape(2,3,4))

In [2]: p.to_xarray()
Out[2]:
<xarray.DataArray (items: 2, major_axis: 3, minor_axis: 4)>
array([[[ 0,  1,  2,  3],
        [ 4,  5,  6,  7],
        [ 8,  9, 10, 11]],

       [[12, 13, 14, 15],
        [16, 17, 18, 19],
        [20, 21, 22, 23]]])
Coordinates:
  * items       (items) int64 0 1
  * major_axis  (major_axis) int64 0 1 2
  * minor_axis  (minor_axis) int64 0 1 2 3
```

Latex representation

`DataFrame` has gained a `._repr_latex_()` method in order to allow for conversion to latex in a ipython/jupyter notebook using nbconvert. (GH11778)

Note that this must be activated by setting the option `pd.display.latex.repr=True` (GH12182)

For example, if you have a jupyter notebook you plan to convert to latex using nbconvert, place the statement `pd.display.latex.repr=True` in the first cell to have the contained DataFrame output also stored as latex.

The options `display.latex.escape` and `display.latex.longtable` have also been added to the configuration and are used automatically by the `to_latex` method. See the *available options docs* for more info.

`pd.read_sas()` changes

`read_sas` has gained the ability to read SAS7BDAT files, including compressed files. The files can be read in entirety, or incrementally. For full details see *here*. (GH4052)

Other enhancements

- Handle truncated floats in SAS xport files (GH11713)
- Added option to hide index in `Series.to_string` (GH11729)
- `read_excel` now supports s3 urls of the format `s3://bucketname/filename` (GH11447)
- add support for `AWS_S3_HOST` env variable when reading from s3 (GH12198)
- A simple version of `Panel.round()` is now implemented (GH11763)
- For Python 3.x, `round(DataFrame)`, `round(Series)`, `round(Panel)` will work (GH11763)
- `sys.getsizeof(obj)` returns the memory usage of a pandas object, including the values it contains (GH11597)
- `Series` gained an `is_unique` attribute (GH11946)
- `DataFrame.quantile` and `Series.quantile` now accept `interpolation` keyword (GH10174).
- Added `DataFrame.style.format` for more flexible formatting of cell values (GH11692)
- `DataFrame.select_dtypes` now allows the `np.float16` type code (GH11990)
- `pivot_table()` now accepts most iterables for the `values` parameter (GH12017)
- Added Google `BigQuery` service account authentication support, which enables authentication on remote servers. (GH11881, GH12572). For further details see here
- `HDFStore` is now iterable: `for k in store` is equivalent to `for k in store.keys()` (GH12221).
- Add missing methods/fields to `.dt` for `Period` (GH8848)
- The entire code base has been `PEP`-ified (GH12096)

Backwards incompatible API changes

- the leading white spaces have been removed from the output of `.to_string(index=False)` method (GH11833)
- the `out` parameter has been removed from the `Series.round()` method. (GH11763)
- `DataFrame.round()` leaves non-numeric columns unchanged in its return, rather than raises. (GH11885)
- `DataFrame.head(0)` and `DataFrame.tail(0)` return empty frames, rather than `self`. (GH11937)
- `Series.head(0)` and `Series.tail(0)` return empty series, rather than `self`. (GH11937)
- `to_msgpack` and `read_msgpack` encoding now defaults to `'utf-8'`. (GH12170)
- the order of keyword arguments to text file parsing functions (`.read_csv()`, `.read_table()`, `.read_fwf()`) changed to group related arguments. (GH11555)
- `NaTType.isoformat` now returns the string `'NaT` to allow the result to be passed to the constructor of `Timestamp`. (GH12300)

NaT and Timedelta operations

`NaT` and `Timedelta` have expanded arithmetic operations, which are extended to `Series` arithmetic where applicable. Operations defined for `datetime64[ns]` or `timedelta64[ns]` are now also defined for `NaT` (GH11564).

`NaT` now supports arithmetic operations with integers and floats.

```
In [58]: pd.NaT * 1
Out[58]: NaT

In [59]: pd.NaT * 1.5
Out[59]: NaT

In [60]: pd.NaT / 2
Out[60]: NaT

In [61]: pd.NaT * np.nan
Out[61]: NaT
```

`NaT` defines more arithmetic operations with `datetime64[ns]` and `timedelta64[ns]`.

```
In [62]: pd.NaT / pd.NaT
Out[62]: nan

In [63]: pd.Timedelta('1s') / pd.NaT
Out[63]: nan
```

`NaT` may represent either a `datetime64[ns]` null or a `timedelta64[ns]` null. Given the ambiguity, it is treated as a `timedelta64[ns]`, which allows more operations to succeed.

```
In [64]: pd.NaT + pd.NaT
Out[64]: NaT

# same as
In [65]: pd.Timedelta('1s') + pd.Timedelta('1s')
Out[65]: Timedelta('0 days 00:00:02')
```

as opposed to

```
In [3]: pd.Timestamp('19900315') + pd.Timestamp('19900315')
TypeError: unsupported operand type(s) for +: 'Timestamp' and 'Timestamp'
```

However, when wrapped in a `Series` whose `dtype` is `datetime64[ns]` or `timedelta64[ns]`, the `dtype` information is respected.

```
In [1]: pd.Series([pd.NaT], dtype='<M8[ns]') + pd.Series([pd.NaT], dtype='<M8[ns]')
TypeError: can only operate on a datetimes for subtraction,
           but the operator [__add__] was passed
```

```
In [66]: pd.Series([pd.NaT], dtype='<m8[ns]') + pd.Series([pd.NaT], dtype='<m8[ns]')
Out[66]:
0   NaT
Length: 1, dtype: timedelta64[ns]
```

`Timedelta` division by `floats` now works.

```
In [67]: pd.Timedelta('1s') / 2.0
Out[67]: Timedelta('0 days 00:00:00.500000')
```

Subtraction by `Timedelta` in a `Series` by a `Timestamp` works (GH11925)

```
In [68]: ser = pd.Series(pd.timedelta_range('1 day', periods=3))

In [69]: ser
Out[69]:
0   1 days
1   2 days
2   3 days
Length: 3, dtype: timedelta64[ns]

In [70]: pd.Timestamp('2012-01-01') - ser
Out[70]:
0   2011-12-31
1   2011-12-30
2   2011-12-29
Length: 3, dtype: datetime64[ns]
```

`NaT.isoformat()` now returns `'NaT'`. This change allows allows `pd.Timestamp` to rehydrate any timestamp like object from its isoformat (GH12300).

Changes to msgpack

Forward incompatible changes in `msgpack` writing format were made over 0.17.0 and 0.18.0; older versions of pandas cannot read files packed by newer versions (GH12129, GH10527)

Bugs in `to_msgpack` and `read_msgpack` introduced in 0.17.0 and fixed in 0.18.0, caused files packed in Python 2 unreadable by Python 3 (GH12142). The following table describes the backward and forward compat of msgpacks.

Warning:

Packed with	Can be unpacked with
pre-0.17 / Python 2	any
pre-0.17 / Python 3	any
0.17 / Python 2	• ==0.17 / Python 2 • >=0.18 / any Python
0.17 / Python 3	>=0.18 / any Python
0.18	>= 0.18

0.18.0 is backward-compatible for reading files packed by older versions, except for files packed with 0.17 in Python 2, in which case only they can only be unpacked in Python 2.

Signature change for .rank

`Series.rank` and `DataFrame.rank` now have the same signature (GH11759)

Previous signature

```
In [3]: pd.Series([0,1]).rank(method='average', na_option='keep',
                              ascending=True, pct=False)
Out[3]:
0     1
1     2
dtype: float64

In [4]: pd.DataFrame([0,1]).rank(axis=0, numeric_only=None,
                                 method='average', na_option='keep',
                                 ascending=True, pct=False)
Out[4]:
   0
0  1
1  2
```

New signature

```
In [71]: pd.Series([0,1]).rank(axis=0, method='average', numeric_only=None,
   ....:                       na_option='keep', ascending=True, pct=False)
   ....:
Out[71]:
0    1.0
1    2.0
Length: 2, dtype: float64

In [72]: pd.DataFrame([0,1]).rank(axis=0, method='average', numeric_only=None,
   ....:                          na_option='keep', ascending=True, pct=False)
   ....:
Out[72]:
     0
0  1.0
1  2.0

[2 rows x 1 columns]
```

Bug in QuarterBegin with n=0

In previous versions, the behavior of the QuarterBegin offset was inconsistent depending on the date when the `n` parameter was 0. (GH11406)

The general semantics of anchored offsets for `n=0` is to not move the date when it is an anchor point (e.g., a quarter start date), and otherwise roll forward to the next anchor point.