- Suggested tutorials in new *Tutorials* section.
- Our pandas ecosystem is growing, We now feature related projects in a new Pandas Ecosystem section.
- Much work has been taking place on improving the docs, and a new *Contributing* section has been added.
- Even though it may only be of interest to devs, we <3 our new CI status page: ScatterCI.

Warning: 0.13.1 fixes a bug that was caused by a combination of having numpy < 1.8, and doing chained assignment on a string-like array. Please review *the docs*, chained indexing can have unexpected results and should generally be avoided.

This would previously segfault:

```
In [1]: df = pd.DataFrame({'A': np.array(['foo', 'bar', 'bah', 'foo', 'bar'])})

In [2]: df['A'].iloc[0] = np.nan

In [3]: df
Out[3]:
     A
0  NaN
1  bar
2  bah
3  foo
4  bar
```

The recommended way to do this type of assignment is:

```
In [4]: df = pd.DataFrame({'A': np.array(['foo', 'bar', 'bah', 'foo', 'bar'])})

In [5]: df.loc[0, 'A'] = np.nan

In [6]: df
Out[6]:
     A
0  NaN
1  bar
2  bah
3  foo
4  bar
```

Output formatting enhancements

- df.info() view now display dtype info per column (GH5682)
- df.info() now honors the option `max_info_rows`, to disable null counts for large frames (GH5974)

```
In [7]: max_info_rows = pd.get_option('max_info_rows')

In [8]: df = pd.DataFrame({'A': np.random.randn(10),
   ...:                    'B': np.random.randn(10),
   ...:                    'C': pd.date_range('20130101', periods=10)
   ...:                    })
   ...:

In [9]: df.iloc[3:6, [0, 2]] = np.nan
```

```
# set to not display the null counts
In [10]: pd.set_option('max_info_rows', 0)

In [11]: df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 3 columns):
 #   Column  Dtype
---  ------  -----
 0   A       float64
 1   B       float64
 2   C       datetime64[ns]
dtypes: datetime64[ns](1), float64(2)
memory usage: 368.0 bytes
```

```
# this is the default (same as in 0.13.0)
In [12]: pd.set_option('max_info_rows', max_info_rows)

In [13]: df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   A       7 non-null      float64
 1   B       10 non-null     float64
 2   C       7 non-null      datetime64[ns]
dtypes: datetime64[ns](1), float64(2)
memory usage: 368.0 bytes
```

- Add `show_dimensions` display option for the new DataFrame repr to control whether the dimensions print.

```
In [14]: df = pd.DataFrame([[1, 2], [3, 4]])

In [15]: pd.set_option('show_dimensions', False)

In [16]: df
Out[16]:
   0  1
0  1  2
1  3  4

In [17]: pd.set_option('show_dimensions', True)

In [18]: df
Out[18]:
   0  1
0  1  2
1  3  4

[2 rows x 2 columns]
```

- The `ArrayFormatter` for `datetime` and `timedelta64` now intelligently limit precision based on the values in the array (GH3401)

  Previously output might look like:

```
  age                 today               diff
0 2001-01-01 00:00:00 2013-04-19 00:00:00 4491 days, 00:00:00
1 2004-06-01 00:00:00 2013-04-19 00:00:00 3244 days, 00:00:00
```

Now the output looks like:

```
In [19]: df = pd.DataFrame([pd.Timestamp('20010101'),
   ....:                    pd.Timestamp('20040601')], columns=['age'])
   ....:

In [20]: df['today'] = pd.Timestamp('20130419')

In [21]: df['diff'] = df['today'] - df['age']

In [22]: df
Out[22]:
         age      today      diff
0 2001-01-01 2013-04-19 4491 days
1 2004-06-01 2013-04-19 3244 days

[2 rows x 3 columns]
```

API changes

- Add `-NaN` and `-nan` to the default set of NA values (GH5952). See *NA Values*.
- Added `Series.str.get_dummies` vectorized string method (GH6021), to extract dummy/indicator variables for separated string columns:

```
In [23]: s = pd.Series(['a', 'a|b', np.nan, 'a|c'])

In [24]: s.str.get_dummies(sep='|')
Out[24]:
   a  b  c
0  1  0  0
1  1  1  0
2  0  0  0
3  1  0  1

[4 rows x 3 columns]
```

- Added the `NDFrame.equals()` method to compare if two NDFrames are equal have equal axes, dtypes, and values. Added the `array_equivalent` function to compare if two ndarrays are equal. NaNs in identical locations are treated as equal. (GH5283) See also *the docs* for a motivating example.

```
df = pd.DataFrame({'col': ['foo', 0, np.nan]})
df2 = pd.DataFrame({'col': [np.nan, 0, 'foo']}, index=[2, 1, 0])
df.equals(df2)
df.equals(df2.sort_index())
```

- `DataFrame.apply` will use the `reduce` argument to determine whether a `Series` or a `DataFrame` should be returned when the `DataFrame` is empty (GH6007).

  Previously, calling `DataFrame.apply` an empty `DataFrame` would return either a `DataFrame` if there were no columns, or the function being applied would be called with an empty `Series` to guess whether a `Series` or `DataFrame` should be returned:

```
In [32]: def applied_func(col):
  ....:     print("Apply function being called with: ", col)
  ....:     return col.sum()
  ....:

In [33]: empty = DataFrame(columns=['a', 'b'])

In [34]: empty.apply(applied_func)
Apply function being called with:  Series([], Length: 0, dtype: float64)
Out[34]:
a   NaN
b   NaN
Length: 2, dtype: float64
```

Now, when `apply` is called on an empty `DataFrame`: if the `reduce` argument is `True` a `Series` will returned, if it is `False` a `DataFrame` will be returned, and if it is `None` (the default) the function being applied will be called with an empty series to try and guess the return type.

```
In [35]: empty.apply(applied_func, reduce=True)
Out[35]:
a   NaN
b   NaN
Length: 2, dtype: float64

In [36]: empty.apply(applied_func, reduce=False)
Out[36]:
Empty DataFrame
Columns: [a, b]
Index: []

[0 rows x 2 columns]
```

Prior version deprecations/changes

There are no announced changes in 0.13 or prior that are taking effect as of 0.13.1

Deprecations

There are no deprecations of prior behavior in 0.13.1

Enhancements

- `pd.read_csv` and `pd.to_datetime` learned a new `infer_datetime_format` keyword which greatly improves parsing perf in many cases. Thanks to @lexual for suggesting and @danbirken for rapidly implementing. (GH5490, GH6021)

  If `parse_dates` is enabled and this flag is set, pandas will attempt to infer the format of the datetime strings in the columns, and if it can be inferred, switch to a faster method of parsing them. In some cases this can increase the parsing speed by ~5-10x.

```
# Try to infer the format for the index column
df = pd.read_csv('foo.csv', index_col=0, parse_dates=True,
                 infer_datetime_format=True)
```

- `date_format` and `datetime_format` keywords can now be specified when writing to `excel` files (GH4133)
- `MultiIndex.from_product` convenience function for creating a MultiIndex from the cartesian product of a set of iterables (GH6055):

```
In [25]: shades = ['light', 'dark']

In [26]: colors = ['red', 'green', 'blue']

In [27]: pd.MultiIndex.from_product([shades, colors], names=['shade', 'color'])
Out[27]:
MultiIndex([('light',   'red'),
            ('light', 'green'),
            ('light',  'blue'),
            ( 'dark',   'red'),
            ( 'dark', 'green'),
            ( 'dark',  'blue')],
           names=['shade', 'color'])
```

- Panel `apply()` will work on non-ufuncs. See *the docs*.

```
In [28]: import pandas._testing as tm

In [29]: panel = tm.makePanel(5)

In [30]: panel
Out[30]:
<class 'pandas.core.panel.Panel'>
Dimensions: 3 (items) x 5 (major_axis) x 4 (minor_axis)
Items axis: ItemA to ItemC
Major_axis axis: 2000-01-03 00:00:00 to 2000-01-07 00:00:00
Minor_axis axis: A to D

In [31]: panel['ItemA']
Out[31]:
                   A         B         C         D
2000-01-03 -0.673690  0.577046 -1.344312 -1.469388
2000-01-04  0.113648 -1.715002  0.844885  0.357021
2000-01-05 -1.478427 -1.039268  1.075770 -0.674600
2000-01-06  0.524988 -0.370647 -0.109050 -1.776904
2000-01-07  0.404705 -1.157892  1.643563 -0.968914

[5 rows x 4 columns]
```

Specifying an `apply` that operates on a Series (to return a single element)

```
In [32]: panel.apply(lambda x: x.dtype, axis='items')
Out[32]:
                A        B        C        D
2000-01-03  float64  float64  float64  float64
2000-01-04  float64  float64  float64  float64
2000-01-05  float64  float64  float64  float64
2000-01-06  float64  float64  float64  float64
2000-01-07  float64  float64  float64  float64

[5 rows x 4 columns]
```

A similar reduction type operation

```
In [33]: panel.apply(lambda x: x.sum(), axis='major_axis')
Out[33]:
      ItemA     ItemB     ItemC
A -1.108775 -1.090118 -2.984435
B -3.705764  0.409204  1.866240
C  2.110856  2.960500 -0.974967
D -4.532785  0.303202 -3.685193

[4 rows x 3 columns]
```

This is equivalent to

```
In [34]: panel.sum('major_axis')
Out[34]:
      ItemA     ItemB     ItemC
A -1.108775 -1.090118 -2.984435
B -3.705764  0.409204  1.866240
C  2.110856  2.960500 -0.974967
D -4.532785  0.303202 -3.685193

[4 rows x 3 columns]
```

A transformation operation that returns a Panel, but is computing the z-score across the major_axis

```
In [35]: result = panel.apply(lambda x: (x - x.mean()) / x.std(),
  ....:                       axis='major_axis')
  ....:

In [36]: result
Out[36]:
<class 'pandas.core.panel.Panel'>
Dimensions: 3 (items) x 5 (major_axis) x 4 (minor_axis)
Items axis: ItemA to ItemC
Major_axis axis: 2000-01-03 00:00:00 to 2000-01-07 00:00:00
Minor_axis axis: A to D

In [37]: result['ItemA']                          # noqa E999
Out[37]:
                   A         B         C         D
2000-01-03 -0.535778  1.500802 -1.506416 -0.681456
2000-01-04  0.397628 -1.108752  0.360481  1.529895
2000-01-05 -1.489811 -0.339412  0.557374  0.280845
2000-01-06  0.885279  0.421830 -0.453013 -1.053785
2000-01-07  0.742682 -0.474468  1.041575 -0.075499

[5 rows x 4 columns]
```

- Panel `apply()` operating on cross-sectional slabs. (GH1148)

```
In [38]: def f(x):
   ....:     return ((x.T - x.mean(1)) / x.std(1)).T
   ....:

In [39]: result = panel.apply(f, axis=['items', 'major_axis'])

In [40]: result
Out[40]:
```

(continues on next page)

(continued from previous page)

```
<class 'pandas.core.panel.Panel'>
Dimensions: 4 (items) x 5 (major_axis) x 3 (minor_axis)
Items axis: A to D
Major_axis axis: 2000-01-03 00:00:00 to 2000-01-07 00:00:00
Minor_axis axis: ItemA to ItemC

In [41]: result.loc[:, :, 'ItemA']
Out[41]:
                   A         B         C         D
2000-01-03  0.012922 -0.030874 -0.629546 -0.757034
2000-01-04  0.392053 -1.071665  0.163228  0.548188
2000-01-05 -1.093650 -0.640898  0.385734 -1.154310
2000-01-06  1.005446 -1.154593 -0.595615 -0.809185
2000-01-07  0.783051 -0.198053  0.919339 -1.052721

[5 rows x 4 columns]
```

This is equivalent to the following

```
In [42]: result = pd.Panel({ax: f(panel.loc[:, :, ax]) for ax in panel.minor_axis}
↪)

In [43]: result
Out[43]:
<class 'pandas.core.panel.Panel'>
Dimensions: 4 (items) x 5 (major_axis) x 3 (minor_axis)
Items axis: A to D
Major_axis axis: 2000-01-03 00:00:00 to 2000-01-07 00:00:00
Minor_axis axis: ItemA to ItemC

In [44]: result.loc[:, :, 'ItemA']
Out[44]:
                   A         B         C         D
2000-01-03  0.012922 -0.030874 -0.629546 -0.757034
2000-01-04  0.392053 -1.071665  0.163228  0.548188
2000-01-05 -1.093650 -0.640898  0.385734 -1.154310
2000-01-06  1.005446 -1.154593 -0.595615 -0.809185
2000-01-07  0.783051 -0.198053  0.919339 -1.052721

[5 rows x 4 columns]
```

Performance

Performance improvements for 0.13.1

- Series datetime/timedelta binary operations (GH5801)
- DataFrame `count/dropna` for `axis=1`
- Series.str.contains now has a *regex=False* keyword which can be faster for plain (non-regex) string patterns. (GH5879)
- Series.str.extract (GH5944)
- `dtypes/ftypes` methods (GH5968)
- indexing with object dtypes (GH5968)

- `DataFrame.apply` (GH6013)
- Regression in JSON IO (GH5765)
- Index construction from Series (GH6150)

Experimental

There are no experimental changes in 0.13.1

Bug fixes

- Bug in `io.wb.get_countries` not including all countries (GH6008)
- Bug in Series replace with timestamp dict (GH5797)
- read_csv/read_table now respects the *prefix* kwarg (GH5732).
- Bug in selection with missing values via `.ix` from a duplicate indexed DataFrame failing (GH5835)
- Fix issue of boolean comparison on empty DataFrames (GH5808)
- Bug in isnull handling `NaT` in an object array (GH5443)
- Bug in `to_datetime` when passed a `np.nan` or integer datelike and a format string (GH5863)
- Bug in groupby dtype conversion with datetimelike (GH5869)
- Regression in handling of empty Series as indexers to Series (GH5877)
- Bug in internal caching, related to (GH5727)
- Testing bug in reading JSON/msgpack from a non-filepath on windows under py3 (GH5874)
- Bug when assigning to .ix[tuple(...)] (GH5896)
- Bug in fully reindexing a Panel (GH5905)
- Bug in idxmin/max with object dtypes (GH5914)
- Bug in `BusinessDay` when adding n days to a date not on offset when n>5 and n%5==0 (GH5890)
- Bug in assigning to chained series with a series via ix (GH5928)
- Bug in creating an empty DataFrame, copying, then assigning (GH5932)
- Bug in DataFrame.tail with empty frame (GH5846)
- Bug in propagating metadata on `resample` (GH5862)
- Fixed string-representation of `NaT` to be "NaT" (GH5708)
- Fixed string-representation for Timestamp to show nanoseconds if present (GH5912)
- `pd.match` not returning passed sentinel
- `Panel.to_frame()` no longer fails when `major_axis` is a `MultiIndex` (GH5402).
- Bug in `pd.read_msgpack` with inferring a `DateTimeIndex` frequency incorrectly (GH5947)
- Fixed `to_datetime` for array with both Tz-aware datetimes and `NaT`'s (GH5961)
- Bug in rolling skew/kurtosis when passed a Series with bad data (GH5749)
- Bug in scipy `interpolate` methods with a datetime index (GH5975)
- Bug in NaT comparison if a mixed datetime/np.datetime64 with NaT were passed (GH5968)

- Fixed bug with `pd.concat` losing dtype information if all inputs are empty (GH5742)
- Recent changes in IPython cause warnings to be emitted when using previous versions of pandas in QTConsole, now fixed. If you're using an older version and need to suppress the warnings, see (GH5922).
- Bug in merging `timedelta` dtypes (GH5695)
- Bug in plotting.scatter_matrix function. Wrong alignment among diagonal and off-diagonal plots, see (GH5497).
- Regression in Series with a MultiIndex via ix (GH6018)
- Bug in Series.xs with a MultiIndex (GH6018)
- Bug in Series construction of mixed type with datelike and an integer (which should result in object type and not automatic conversion) (GH6028)
- Possible segfault when chained indexing with an object array under NumPy 1.7.1 (GH6026, GH6056)
- Bug in setting using fancy indexing a single element with a non-scalar (e.g. a list), (GH6043)
- `to_sql` did not respect `if_exists` (GH4110 GH4304)
- Regression in `.get(None)` indexing from 0.12 (GH5652)
- Subtle `iloc` indexing bug, surfaced in (GH6059)
- Bug with insert of strings into DatetimeIndex (GH5818)
- Fixed unicode bug in to_html/HTML repr (GH6098)
- Fixed missing arg validation in get_options_data (GH6105)
- Bug in assignment with duplicate columns in a frame where the locations are a slice (e.g. next to each other) (GH6120)
- Bug in propagating _ref_locs during construction of a DataFrame with dups index/columns (GH6121)
- Bug in `DataFrame.apply` when using mixed datelike reductions (GH6125)
- Bug in `DataFrame.append` when appending a row with different columns (GH6129)
- Bug in DataFrame construction with recarray and non-ns datetime dtype (GH6140)
- Bug in `.loc` setitem indexing with a dataframe on rhs, multiple item setting, and a datetimelike (GH6152)
- Fixed a bug in `query`/`eval` during lexicographic string comparisons (GH6155).
- Fixed a bug in `query` where the index of a single-element `Series` was being thrown away (GH6148).
- Bug in `HDFStore` on appending a dataframe with MultiIndexed columns to an existing table (GH6167)
- Consistency with dtypes in setting an empty DataFrame (GH6171)
- Bug in selecting on a MultiIndex `HDFStore` even in the presence of under specified column spec (GH6169)
- Bug in `nanops.var` with `ddof=1` and 1 elements would sometimes return `inf` rather than `nan` on some platforms (GH6136)
- Bug in Series and DataFrame bar plots ignoring the `use_index` keyword (GH6209)
- Bug in groupby with mixed str/int under python3 fixed; `argsort` was failing (GH6212)

Contributors

A total of 52 people contributed patches to this release. People with a "+" by their names contributed a patch for the first time.

- Alex Rothberg
- Alok Singhal +
- Andrew Burrows +
- Andy Hayden
- Bjorn Arneson +
- Brad Buran
- Caleb Epstein
- Chapman Siu
- Chase Albert +
- Clark Fitzgerald +
- DSM
- Dan Birken
- Daniel Waeber +
- David Wolever +
- Doran Deluz +
- Douglas McNeil +
- Douglas Rudd +
- Drazen Lucanin
- Elliot S +
- Felix Lawrence +
- George Kuan +
- Guillaume Gay +
- Jacob Schaer
- Jan Wagner +
- Jeff Tratner
- John McNamara
- Joris Van den Bossche
- Julia Evans +
- Kieran O'Mahony
- Michael Schatzow +
- Naveen Michaud-Agrawal +
- Patrick O'Keeffe +
- Phillip Cloud

- Roman Pekar
- Skipper Seabold
- Spencer Lyon
- Tom Augspurger +
- TomAugspurger
- acorbe +
- akittredge +
- bmu +
- bwignall +
- chapman siu
- danielballan
- david +
- davidshinn
- immerrr +
- jreback
- lexual
- mwaskom +
- unutbu
- y-p

5.14.2 v0.13.0 (January 3, 2014)

This is a major release from 0.12.0 and includes a number of API changes, several new features and enhancements along with a large number of bug fixes.

Highlights include:

- support for a new index type `Float64Index`, and other Indexing enhancements
- `HDFStore` has a new string based syntax for query specification
- support for new methods of interpolation
- updated `timedelta` operations
- a new string manipulation method `extract`
- Nanosecond support for Offsets
- `isin` for DataFrames

Several experimental features are added, including:

- new `eval/query` methods for expression evaluation
- support for `msgpack` serialization
- an i/o interface to Google's `BigQuery`

Their are several new or updated docs sections including:

- *Comparison with SQL*, which should be useful for those familiar with SQL but still learning pandas.
- *Comparison with R*, idiom translations from R to pandas.
- *Enhancing Performance*, ways to enhance pandas performance with `eval/query`.

Warning: In 0.13.0 `Series` has internally been refactored to no longer sub-class `ndarray` but instead subclass `NDFrame`, similar to the rest of the pandas containers. This should be a transparent change with only very limited API implications. See *Internal Refactoring*

API changes

- `read_excel` now supports an integer in its `sheetname` argument giving the index of the sheet to read in (GH4301).
- Text parser now treats anything that reads like inf ("inf", "Inf", "-Inf", "iNf", etc.) as infinity. (GH4220, GH4219), affecting `read_table`, `read_csv`, etc.
- `pandas` now is Python 2/3 compatible without the need for 2to3 thanks to @jtratner. As a result, pandas now uses iterators more extensively. This also led to the introduction of substantive parts of the Benjamin Peterson's `six` library into compat. (GH4384, GH4375, GH4372)
- `pandas.util.compat` and `pandas.util.py3compat` have been merged into `pandas.compat`. `pandas.compat` now includes many functions allowing 2/3 compatibility. It contains both list and iterator versions of range, filter, map and zip, plus other necessary elements for Python 3 compatibility. `lmap`, `lzip`, `lrange` and `lfilter` all produce lists instead of iterators, for compatibility with `numpy`, subscripting and `pandas` constructors.(GH4384, GH4375, GH4372)
- `Series.get` with negative indexers now returns the same as `[]` (GH4390)
- Changes to how `Index` and `MultiIndex` handle metadata (`levels`, `labels`, and `names`) (GH4039):

```
# previously, you would have set levels or labels directly
>>> pd.index.levels = [[1, 2, 3, 4], [1, 2, 4, 4]]

# now, you use the set_levels or set_labels methods
>>> index = pd.index.set_levels([[1, 2, 3, 4], [1, 2, 4, 4]])

# similarly, for names, you can rename the object
# but setting names is not deprecated
>>> index = pd.index.set_names(["bob", "cranberry"])

# and all methods take an inplace kwarg - but return None
>>> pd.index.set_names(["bob", "cranberry"], inplace=True)
```

- **All** division with `NDFrame` objects is now *truedivision*, regardless of the future import. This means that operating on pandas objects will by default use *floating point* division, and return a floating point dtype. You can use `//` and `floordiv` to do integer division.

  Integer division

```
In [3]: arr = np.array([1, 2, 3, 4])

In [4]: arr2 = np.array([5, 3, 2, 1])

In [5]: arr / arr2
Out[5]: array([0, 0, 1, 4])
```

(continues on next page)

(continued from previous page)

```
In [6]: pd.Series(arr) // pd.Series(arr2)
Out[6]:
0    0
1    0
2    1
3    4
dtype: int64
```

True Division

```
In [7]: pd.Series(arr) / pd.Series(arr2)  # no future import required
Out[7]:
0    0.200000
1    0.666667
2    1.500000
3    4.000000
dtype: float64
```

- Infer and downcast dtype if `downcast='infer'` is passed to `fillna/ffill/bfill` (GH4604)
- `__nonzero__` for all NDFrame objects, will now raise a `ValueError`, this reverts back to (GH1073, GH4633) behavior. See *gotchas* for a more detailed discussion.

  This prevents doing boolean comparison on *entire* pandas objects, which is inherently ambiguous. These all will raise a `ValueError`.

```
>>> df = pd.DataFrame({'A': np.random.randn(10),
...                    'B': np.random.randn(10),
...                    'C': pd.date_range('20130101', periods=10)
...                    })
...
>>> if df:
...     pass
...
Traceback (most recent call last):
    ...
ValueError: The truth value of a DataFrame is ambiguous.  Use a.empty,
a.bool(), a.item(), a.any() or a.all().

>>> df1 = df
>>> df2 = df
>>> df1 and df2
Traceback (most recent call last):
    ...
ValueError: The truth value of a DataFrame is ambiguous.  Use a.empty,
a.bool(), a.item(), a.any() or a.all().

>>> d = [1, 2, 3]
>>> s1 = pd.Series(d)
>>> s2 = pd.Series(d)
>>> s1 and s2
Traceback (most recent call last):
    ...
ValueError: The truth value of a DataFrame is ambiguous.  Use a.empty,
a.bool(), a.item(), a.any() or a.all().
```

  Added the `.bool()` method to `NDFrame` objects to facilitate evaluating of single-element boolean Series:

```
In [1]: pd.Series([True]).bool()
Out[1]: True

In [2]: pd.Series([False]).bool()
Out[2]: False

In [3]: pd.DataFrame([[True]]).bool()
Out[3]: True

In [4]: pd.DataFrame([[False]]).bool()
Out[4]: False
```

- All non-Index NDFrames (`Series`, `DataFrame`, `Panel`, `Panel4D`, `SparsePanel`, etc.), now support the entire set of arithmetic operators and arithmetic flex methods (add, sub, mul, etc.). `SparsePanel` does not support `pow` or `mod` with non-scalars. (GH3765)
- `Series` and `DataFrame` now have a `mode()` method to calculate the statistical mode(s) by axis/Series. (GH5367)
- Chained assignment will now by default warn if the user is assigning to a copy. This can be changed with the option `mode.chained_assignment`, allowed options are `raise/warn/None`. See *the docs*.

```
In [5]: dfc = pd.DataFrame({'A': ['aaa', 'bbb', 'ccc'], 'B': [1, 2, 3]})

In [6]: pd.set_option('chained_assignment', 'warn')
```

The following warning / exception will show if this is attempted.

```
In [7]: dfc.loc[0]['A'] = 1111
```

```
Traceback (most recent call last)
   ...
SettingWithCopyWarning:
   A value is trying to be set on a copy of a slice from a DataFrame.
   Try using .loc[row_index,col_indexer] = value instead
```

Here is the correct method of assignment.

```
In [8]: dfc.loc[0, 'A'] = 11

In [9]: dfc
Out[9]:
     A  B
0   11  1
1  bbb  2
2  ccc  3
```

- **`Panel.reindex` has the following call signature `Panel.reindex(items=None, major_axis=None, minor_ax`** to conform with other `NDFrame` objects. See *Internal Refactoring* for more information.
- **`Series.argmin` and `Series.argmax` are now aliased to `Series.idxmin` and `Series.idxmax`. These return the *i*** min or max element respectively. Prior to 0.13.0 these would return the position of the min / max element. (GH6214)

Prior version deprecations/changes

These were announced changes in 0.12 or prior that are taking effect as of 0.13.0

- Remove deprecated `Factor` (GH3650)
- Remove deprecated `set_printoptions/reset_printoptions` (GH3046)
- Remove deprecated `_verbose_info` (GH3215)
- Remove deprecated `read_clipboard/to_clipboard/ExcelFile/ExcelWriter` from `pandas.io.parsers` (GH3717) These are available as functions in the main pandas namespace (e.g. `pd.read_clipboard`)
- default for `tupleize_cols` is now `False` for both `to_csv` and `read_csv`. Fair warning in 0.12 (GH3604)
- default for *display.max_seq_len* is now 100 rather then *None*. This activates truncated display ("...") of long sequences in various places. (GH3391)

Deprecations

Deprecated in 0.13.0

- deprecated `iterkv`, which will be removed in a future release (this was an alias of iteritems used to bypass `2to3`'s changes). (GH4384, GH4375, GH4372)
- deprecated the string method `match`, whose role is now performed more idiomatically by `extract`. In a future release, the default behavior of `match` will change to become analogous to `contains`, which returns a boolean indexer. (Their distinction is strictness: `match` relies on `re.match` while `contains` relies on `re.search`.) In this release, the deprecated behavior is the default, but the new behavior is available through the keyword argument `as_indexer=True`.

Indexing API changes

Prior to 0.13, it was impossible to use a label indexer (`.loc/.ix`) to set a value that was not contained in the index of a particular axis. (GH2578). See *the docs*

In the `Series` case this is effectively an appending operation

```
In [10]: s = pd.Series([1, 2, 3])

In [11]: s
Out[11]:
0    1
1    2
2    3
dtype: int64

In [12]: s[5] = 5.

In [13]: s
Out[13]:
0    1.0
1    2.0
2    3.0
5    5.0
dtype: float64
```

```
In [14]: dfi = pd.DataFrame(np.arange(6).reshape(3, 2),
   ....:                    columns=['A', 'B'])
   ....:

In [15]: dfi
Out[15]:
   A  B
0  0  1
1  2  3
2  4  5
```

This would previously `KeyError`

```
In [16]: dfi.loc[:, 'C'] = dfi.loc[:, 'A']

In [17]: dfi
Out[17]:
   A  B  C
0  0  1  0
1  2  3  2
2  4  5  4
```

This is like an `append` operation.

```
In [18]: dfi.loc[3] = 5

In [19]: dfi
Out[19]:
   A  B  C
0  0  1  0
1  2  3  2
2  4  5  4
3  5  5  5
```

A Panel setting operation on an arbitrary axis aligns the input to the Panel

```
In [20]: p = pd.Panel(np.arange(16).reshape(2, 4, 2),
   ....:              items=['Item1', 'Item2'],
   ....:              major_axis=pd.date_range('2001/1/12', periods=4),
   ....:              minor_axis=['A', 'B'], dtype='float64')
   ....:

In [21]: p
Out[21]:
<class 'pandas.core.panel.Panel'>
Dimensions: 2 (items) x 4 (major_axis) x 2 (minor_axis)
Items axis: Item1 to Item2
Major_axis axis: 2001-01-12 00:00:00 to 2001-01-15 00:00:00
Minor_axis axis: A to B

In [22]: p.loc[:, :, 'C'] = pd.Series([30, 32], index=p.items)

In [23]: p
Out[23]:
<class 'pandas.core.panel.Panel'>
Dimensions: 2 (items) x 4 (major_axis) x 3 (minor_axis)
Items axis: Item1 to Item2
```

(continues on next page)

(continued from previous page)

```
Major_axis axis: 2001-01-12 00:00:00 to 2001-01-15 00:00:00
Minor_axis axis: A to C

In [24]: p.loc[:, :, 'C']
Out[24]:
            Item1  Item2
2001-01-12   30.0   32.0
2001-01-13   30.0   32.0
2001-01-14   30.0   32.0
2001-01-15   30.0   32.0
```

Float64Index API change

- Added a new index type, `Float64Index`. This will be automatically created when passing floating values in index creation. This enables a pure label-based slicing paradigm that makes `[],ix,loc` for scalar indexing and slicing work exactly the same. See *the docs*, (GH263)

  Construction is by default for floating type values.

```
In [20]: index = pd.Index([1.5, 2, 3, 4.5, 5])

In [21]: index
Out[21]: Float64Index([1.5, 2.0, 3.0, 4.5, 5.0], dtype='float64')

In [22]: s = pd.Series(range(5), index=index)

In [23]: s
Out[23]:
1.5    0
2.0    1
3.0    2
4.5    3
5.0    4
dtype: int64
```

  Scalar selection for `[],.ix,.loc` will always be label based. An integer will match an equal float index (e.g. `3` is equivalent to `3.0`)

```
In [24]: s[3]
Out[24]: 2

In [25]: s.loc[3]
Out[25]: 2
```

  The only positional indexing is via `iloc`

```
In [26]: s.iloc[3]
Out[26]: 3
```

  A scalar index that is not found will raise `KeyError`

  Slicing is ALWAYS on the values of the index, for `[],ix,loc` and ALWAYS positional with `iloc`

```
In [27]: s[2:4]
Out[27]:
```

(continues on next page)

(continued from previous page)

```
2.0    1
3.0    2
dtype: int64

In [28]: s.loc[2:4]
Out[28]:
2.0    1
3.0    2
dtype: int64

In [29]: s.iloc[2:4]
Out[29]:
3.0    2
4.5    3
dtype: int64
```

In float indexes, slicing using floats are allowed

```
In [30]: s[2.1:4.6]
Out[30]:
3.0    2
4.5    3
dtype: int64

In [31]: s.loc[2.1:4.6]
Out[31]:
3.0    2
4.5    3
dtype: int64
```

- Indexing on other index types are preserved (and positional fallback for `[],ix`), with the exception, that floating point slicing on indexes on non `Float64Index` will now raise a `TypeError`.

```
In [1]: pd.Series(range(5))[3.5]
TypeError: the label [3.5] is not a proper indexer for this index type␣
↪(Int64Index)

In [1]: pd.Series(range(5))[3.5:4.5]
TypeError: the slice start [3.5] is not a proper indexer for this index type␣
↪(Int64Index)
```

Using a scalar float indexer will be deprecated in a future version, but is allowed for now.

```
In [3]: pd.Series(range(5))[3.0]
Out[3]: 3
```

HDFStore API changes

- Query Format Changes. A much more string-like query format is now supported. See *the docs*.

```
In [32]: path = 'test.h5'

In [33]: dfq = pd.DataFrame(np.random.randn(10, 4),
   ....:                    columns=list('ABCD'),
   ....:                    index=pd.date_range('20130101', periods=10))
   ....:

In [34]: dfq.to_hdf(path, 'dfq', format='table', data_columns=True)
```

Use boolean expressions, with in-line function evaluation.

```
In [35]: pd.read_hdf(path, 'dfq',
   ....:             where="index>Timestamp('20130104') & columns=['A', 'B']")
   ....:
Out[35]:
                   A         B
2013-01-05 -0.424972  0.567020
2013-01-06 -0.673690  0.113648
2013-01-07  0.404705  0.577046
2013-01-08 -0.370647 -1.157892
2013-01-09  1.075770 -0.109050
2013-01-10  0.357021 -0.674600
```

Use an inline column reference

```
In [36]: pd.read_hdf(path, 'dfq',
   ....:             where="A>0 or C>0")
   ....:
Out[36]:
                   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
2013-01-04  0.721555 -0.706771 -1.039575  0.271860
2013-01-05 -0.424972  0.567020  0.276232 -1.087401
2013-01-07  0.404705  0.577046 -1.715002 -1.039268
2013-01-09  1.075770 -0.109050  1.643563 -1.469388
2013-01-10  0.357021 -0.674600 -1.776904 -0.968914
```

- the `format` keyword now replaces the `table` keyword; allowed values are `fixed(f)` or `table(t)` the same defaults as prior < 0.13.0 remain, e.g. `put` implies `fixed` format and `append` implies `table` format. This default format can be set as an option by setting `io.hdf.default_format`.

```
In [37]: path = 'test.h5'

In [38]: df = pd.DataFrame(np.random.randn(10, 2))

In [39]: df.to_hdf(path, 'df_table', format='table')

In [40]: df.to_hdf(path, 'df_table2', append=True)

In [41]: df.to_hdf(path, 'df_fixed')

In [42]: with pd.HDFStore(path) as store:
   ....:     print(store)
```

(continues on next page)

(continued from previous page)

```
   ....:
<class 'pandas.io.pytables.HDFStore'>
File path: test.h5
```

- Significant table writing performance improvements
- handle a passed `Series` in table format (GH4330)
- can now serialize a `timedelta64[ns]` dtype in a table (GH3577), See *the docs*.
- added an `is_open` property to indicate if the underlying file handle is_open; a closed store will now report 'CLOSED' when viewing the store (rather than raising an error) (GH4409)
- a close of a `HDFStore` now will close that instance of the `HDFStore` but will only close the actual file if the ref count (by `PyTables`) w.r.t. all of the open handles are 0. Essentially you have a local instance of `HDFStore` referenced by a variable. Once you close it, it will report closed. Other references (to the same file) will continue to operate until they themselves are closed. Performing an action on a closed file will raise `ClosedFileError`

```
In [43]: path = 'test.h5'

In [44]: df = pd.DataFrame(np.random.randn(10, 2))

In [45]: store1 = pd.HDFStore(path)

In [46]: store2 = pd.HDFStore(path)

In [47]: store1.append('df', df)

In [48]: store2.append('df2', df)

In [49]: store1
Out[49]:
<class 'pandas.io.pytables.HDFStore'>
File path: test.h5

In [50]: store2
Out[50]:
<class 'pandas.io.pytables.HDFStore'>
File path: test.h5

In [51]: store1.close()

In [52]: store2
Out[52]:
<class 'pandas.io.pytables.HDFStore'>
File path: test.h5

In [53]: store2.close()

In [54]: store2
Out[54]:
<class 'pandas.io.pytables.HDFStore'>
File path: test.h5
```

- removed the `_quiet` attribute, replace by a `DuplicateWarning` if retrieving duplicate rows from a table (GH4367)

- removed the `warn` argument from `open`. Instead a `PossibleDataLossError` exception will be raised if you try to use `mode='w'` with an OPEN file handle (GH4367)
- allow a passed locations array or mask as a `where` condition (GH4467). See *the docs* for an example.
- add the keyword `dropna=True` to `append` to change whether ALL nan rows are not written to the store (default is `True`, ALL nan rows are NOT written), also settable via the option `io.hdf.dropna_table` (GH4625)
- pass through store creation arguments; can be used to support in-memory stores

DataFrame repr changes

The HTML and plain text representations of *`DataFrame`* now show a truncated view of the table once it exceeds a certain size, rather than switching to the short info view (GH4886, GH5550). This makes the representation more consistent as small DataFrames get larger.

2010-03-29	13.70	13.88	13.39	13.57	158225000	12.98
2010-03-30	13.55	13.64	13.18	13.28	142055200	12.70
	...	...	...	...	...	...

771 rows × 6 columns

To get the info view, call *`DataFrame.info()`*. If you prefer the info view as the repr for large DataFrames, you can set this by running `set_option('display.large_repr', 'info')`.

Enhancements

- `df.to_clipboard()` learned a new `excel` keyword that let's you paste df data directly into excel (enabled by default). (GH5070).
- `read_html` now raises a `URLError` instead of catching and raising a `ValueError` (GH4303, GH4305)
- Added a test for `read_clipboard()` and `to_clipboard()` (GH4282)
- Clipboard functionality now works with PySide (GH4282)
- Added a more informative error message when plot arguments contain overlapping color and style arguments (GH4402)
- `to_dict` now takes `records` as a possible out type. Returns an array of column-keyed dictionaries. (GH4936)
- `NaN` handing in get_dummies (GH4446) with *dummy_na*

```
# previously, nan was erroneously counted as 2 here
# now it is not counted at all
In [55]: pd.get_dummies([1, 2, np.nan])
Out[55]:
   1.0  2.0
0    1    0
1    0    1
2    0    0

# unless requested
```

(continues on next page)

(continued from previous page)

```
In [56]: pd.get_dummies([1, 2, np.nan], dummy_na=True)
Out[56]:
   1.0  2.0  NaN
0    1    0    0
1    0    1    0
2    0    0    1
```

- `timedelta64[ns]` operations. See *the docs*.

> **Warning:** Most of these operations require `numpy >= 1.7`

Using the new top-level `to_timedelta`, you can convert a scalar or array from the standard timedelta format (produced by `to_csv`) into a timedelta type (`np.timedelta64` in `nanoseconds`).

```
In [57]: pd.to_timedelta('1 days 06:05:01.00003')
Out[57]: Timedelta('1 days 06:05:01.000030')

In [58]: pd.to_timedelta('15.5us')
Out[58]: Timedelta('0 days 00:00:00.000015')

In [59]: pd.to_timedelta(['1 days 06:05:01.00003', '15.5us', 'nan'])
Out[59]: TimedeltaIndex(['1 days 06:05:01.000030', '0 days 00:00:00.000015', NaT],
 ↪ dtype='timedelta64[ns]', freq=None)

In [60]: pd.to_timedelta(np.arange(5), unit='s')
Out[60]: TimedeltaIndex(['00:00:00', '00:00:01', '00:00:02', '00:00:03', '00:00:04
 ↪'], dtype='timedelta64[ns]', freq=None)

In [61]: pd.to_timedelta(np.arange(5), unit='d')
Out[61]: TimedeltaIndex(['0 days', '1 days', '2 days', '3 days', '4 days'], dtype=
 ↪'timedelta64[ns]', freq=None)
```

A Series of dtype `timedelta64[ns]` can now be divided by another `timedelta64[ns]` object, or astyped to yield a `float64` dtyped Series. This is frequency conversion. See *the docs* for the docs.

```
In [62]: import datetime

In [63]: td = pd.Series(pd.date_range('20130101', periods=4)) - pd.Series(
   ....:     pd.date_range('20121201', periods=4))
   ....:

In [64]: td[2] += np.timedelta64(datetime.timedelta(minutes=5, seconds=3))

In [65]: td[3] = np.nan

In [66]: td
Out[66]:
0   31 days 00:00:00
1   31 days 00:00:00
2   31 days 00:05:03
3                NaT
dtype: timedelta64[ns]

# to days
In [67]: td / np.timedelta64(1, 'D')
```

(continues on next page)

(continued from previous page)

```
Out[67]:
0    31.000000
1    31.000000
2    31.003507
3          NaN
dtype: float64

In [68]: td.astype('timedelta64[D]')
Out[68]:
0    31.0
1    31.0
2    31.0
3     NaN
dtype: float64

# to seconds
In [69]: td / np.timedelta64(1, 's')
Out[69]:
0    2678400.0
1    2678400.0
2    2678703.0
3          NaN
dtype: float64

In [70]: td.astype('timedelta64[s]')
Out[70]:
0    2678400.0
1    2678400.0
2    2678703.0
3          NaN
dtype: float64
```

Dividing or multiplying a `timedelta64[ns]` Series by an integer or integer Series

```
In [71]: td * -1
Out[71]:
0   -31 days +00:00:00
1   -31 days +00:00:00
2   -32 days +23:54:57
3                  NaT
dtype: timedelta64[ns]

In [72]: td * pd.Series([1, 2, 3, 4])
Out[72]:
0   31 days 00:00:00
1   62 days 00:00:00
2   93 days 00:15:09
3               NaT
dtype: timedelta64[ns]
```

Absolute `DateOffset` objects can act equivalently to `timedeltas`

```
In [73]: from pandas import offsets

In [74]: td + offsets.Minute(5) + offsets.Milli(5)
Out[74]:
0   31 days 00:05:00.005000
```

(continues on next page)

(continued from previous page)

```
1    31 days 00:05:00.005000
2    31 days 00:10:03.005000
3                        NaT
dtype: timedelta64[ns]
```

Fillna is now supported for timedeltas

```
In [75]: td.fillna(pd.Timedelta(0))
Out[75]:
0    31 days 00:00:00
1    31 days 00:00:00
2    31 days 00:05:03
3     0 days 00:00:00
dtype: timedelta64[ns]

In [76]: td.fillna(datetime.timedelta(days=1, seconds=5))
Out[76]:
0    31 days 00:00:00
1    31 days 00:00:00
2    31 days 00:05:03
3     1 days 00:00:05
dtype: timedelta64[ns]
```

You can do numeric reduction operations on timedeltas.

```
In [77]: td.mean()
Out[77]: Timedelta('31 days 00:01:41')

In [78]: td.quantile(.1)
Out[78]: Timedelta('31 days 00:00:00')
```

- `plot(kind='kde')` now accepts the optional parameters `bw_method` and `ind`, passed to scipy.stats.gaussian_kde() (for scipy >= 0.11.0) to set the bandwidth, and to gkde.evaluate() to specify the indices at which it is evaluated, respectively. See scipy docs. (GH4298)
- DataFrame constructor now accepts a numpy masked record array (GH3478)
- The new vectorized string method `extract` return regular expression matches more conveniently.

```
In [79]: pd.Series(['a1', 'b2', 'c3']).str.extract('[ab](\\d)')
Out[79]:
     0
0    1
1    2
2  NaN
```

Elements that do not match return `NaN`. Extracting a regular expression with more than one group returns a DataFrame with one column per group.

```
In [80]: pd.Series(['a1', 'b2', 'c3']).str.extract('([ab])(\\d)')
Out[80]:
     0    1
0    a    1
1    b    2
2  NaN  NaN
```

Elements that do not match return a row of `NaN`. Thus, a Series of messy strings can be *converted* into a like-indexed Series or DataFrame of cleaned-up or more useful strings, without necessitating `get()` to access tuples

or `re.match` objects.

Named groups like

```
In [81]: pd.Series(['a1', 'b2', 'c3']).str.extract(
   ....:     '(?P<letter>[ab])(?P<digit>\\d)')
   ....:
Out[81]:
  letter digit
0      a     1
1      b     2
2    NaN   NaN
```

and optional groups can also be used.

```
In [82]: pd.Series(['a1', 'b2', '3']).str.extract(
   ....:      '(?P<letter>[ab])?(?P<digit>\\d)')
   ....:
Out[82]:
  letter digit
0      a     1
1      b     2
2    NaN     3
```

- `read_stata` now accepts Stata 13 format (GH4291)
- `read_fwf` now infers the column specifications from the first 100 rows of the file if the data has correctly separated and properly aligned columns using the delimiter provided to the function (GH4488).
- support for nanosecond times as an offset

> **Warning:** These operations require `numpy >= 1.7`

Period conversions in the range of seconds and below were reworked and extended up to nanoseconds. Periods in the nanosecond range are now available.

```
In [83]: pd.date_range('2013-01-01', periods=5, freq='5N')
Out[83]:
DatetimeIndex([          '2013-01-01 00:00:00',
               '2013-01-01 00:00:00.000000005',
               '2013-01-01 00:00:00.000000010',
               '2013-01-01 00:00:00.000000015',
               '2013-01-01 00:00:00.000000020'],
              dtype='datetime64[ns]', freq='5N')
```

or with frequency as offset

```
In [84]: pd.date_range('2013-01-01', periods=5, freq=pd.offsets.Nano(5))
Out[84]:
DatetimeIndex([          '2013-01-01 00:00:00',
               '2013-01-01 00:00:00.000000005',
               '2013-01-01 00:00:00.000000010',
               '2013-01-01 00:00:00.000000015',
               '2013-01-01 00:00:00.000000020'],
              dtype='datetime64[ns]', freq='5N')
```

Timestamps can be modified in the nanosecond range