- Fixed testing issue where too many sockets where open thus leading to a connection reset issue (GH3982, GH3985, GH4028, GH4054)
- Fixed failing tests in test_yahoo, test_google where symbols were not retrieved but were being accessed (GH3982, GH3985, GH4028, GH4054)
- `Series.hist` will now take the figure from the current environment if one is not passed
- Fixed bug where a 1xN DataFrame would barf on a 1xN mask (GH4071)
- Fixed running of `tox` under python3 where the pickle import was getting rewritten in an incompatible way (GH4062, GH4063)
- Fixed bug where sharex and sharey were not being passed to grouped_hist (GH4089)
- Fixed bug in `DataFrame.replace` where a nested dict wasn't being iterated over when regex=False (GH4115)
- Fixed bug in the parsing of microseconds when using the `format` argument in `to_datetime` (GH4152)
- Fixed bug in `PandasAutoDateLocator` where `invert_xaxis` triggered incorrectly `MilliSecondLocator` (GH3990)
- Fixed bug in plotting that wasn't raising on invalid colormap for matplotlib 1.1.1 (GH4215)
- Fixed the legend displaying in `DataFrame.plot(kind='kde')` (GH4216)
- Fixed bug where Index slices weren't carrying the name attribute (GH4226)
- Fixed bug in initializing `DatetimeIndex` with an array of strings in a certain time zone (GH4229)
- Fixed bug where html5lib wasn't being properly skipped (GH4265)
- Fixed bug where get_data_famafrench wasn't using the correct file edges (GH4281)

See the *full release notes* or issue tracker on GitHub for a complete list.

Contributors

A total of 50 people contributed patches to this release. People with a "+" by their names contributed a patch for the first time.

- Andy Hayden
- Chang She
- Christopher Whelan
- Damien Garaud
- Dan Allan
- Dan Birken
- Dieter Vandenbussche
- Dražen Lučanin
- Gábor Lipták +
- Jeff Mellen +
- Jeff Tratner +
- Jeffrey Tratner +
- Jonathan deWerd +

- Joris Van den Bossche +
- Juraj Niznan +
- Karmel Allison
- Kelsey Jordahl
- Kevin Stone +
- Kieran O'Mahony
- Kyle Meyer +
- Mike Kelly +
- PKEuS +
- Patrick O'Brien +
- Phillip Cloud
- Richard Hochenberger +
- Skipper Seabold
- SleepingPills +
- Tobias Brandt
- Tom Farnbauer +
- TomAugspurger +
- Trent Hauck +
- Wes McKinney
- Wouter Overmeire
- Yaroslav Halchenko
- conmai +
- danielballan +
- davidshinn +
- dieterv77
- duozhang +
- ejnens +
- gliptak +
- jniznan +
- jreback
- lexual
- nipunreddevil +
- ogiaquino +
- stonebig +
- tim smith +
- timmie

- y-p

5.16 Version 0.11

5.16.1 v0.11.0 (April 22, 2013)

This is a major release from 0.10.1 and includes many new features and enhancements along with a large number of bug fixes. The methods of Selecting Data have had quite a number of additions, and Dtype support is now full-fledged. There are also a number of important API changes that long-time pandas users should pay close attention to.

There is a new section in the documentation, *10 Minutes to Pandas*, primarily geared to new users.

There is a new section in the documentation, *Cookbook*, a collection of useful recipes in pandas (and that we want contributions!).

There are several libraries that are now *Recommended Dependencies*

Selection choices

Starting in 0.11.0, object selection has had a number of user-requested additions in order to support more explicit location based indexing. Pandas now supports three types of multi-axis indexing.

- `.loc` is strictly label based, will raise `KeyError` when the items are not found, allowed inputs are:
  - A single label, e.g. `5` or `'a'`, (note that `5` is interpreted as a *label* of the index. This use is **not** an integer position along the index)
  - A list or array of labels `['a', 'b', 'c']`
  - A slice object with labels `'a':'f'`, (note that contrary to usual python slices, **both** the start and the stop are included!)
  - A boolean array

  See more at *Selection by Label*

- `.iloc` is strictly integer position based (from `0` to `length-1` of the axis), will raise `IndexError` when the requested indices are out of bounds. Allowed inputs are:
  - An integer e.g. `5`
  - A list or array of integers `[4, 3, 0]`
  - A slice object with ints `1:7`
  - A boolean array

  See more at *Selection by Position*

- `.ix` supports mixed integer and label based access. It is primarily label based, but will fallback to integer positional access. `.ix` is the most general and will support any of the inputs to `.loc` and `.iloc`, as well as support for floating point label schemes. `.ix` is especially useful when dealing with mixed positional and label based hierarchical indexes.

  As using integer slices with `.ix` have different behavior depending on whether the slice is interpreted as position based or label based, it's usually better to be explicit and use `.iloc` or `.loc`.

  See more at *Advanced Indexing* and *Advanced Hierarchical*.

Selection deprecations

Starting in version 0.11.0, these methods *may* be deprecated in future versions.

- irow
- icol
- iget_value

See the section *Selection by Position* for substitutes.

Dtypes

Numeric dtypes will propagate and can coexist in DataFrames. If a dtype is passed (either directly via the dtype keyword, a passed ndarray, or a passed Series, then it will be preserved in DataFrame operations. Furthermore, different numeric dtypes will **NOT** be combined. The following example will give you a taste.

```
In [1]: df1 = pd.DataFrame(np.random.randn(8, 1), columns=['A'], dtype='float32')

In [2]: df1
Out[2]:
          A
0  0.469112
1 -0.282863
2 -1.509058
3 -1.135632
4  1.212112
5 -0.173215
6  0.119209
7 -1.044236

In [3]: df1.dtypes
Out[3]:
A    float32
dtype: object

In [4]: df2 = pd.DataFrame({'A': pd.Series(np.random.randn(8), dtype='float16'),
   ...:                     'B': pd.Series(np.random.randn(8)),
   ...:                     'C': pd.Series(range(8), dtype='uint8')})
   ...:

In [5]: df2
Out[5]:
          A         B  C
0 -0.861816 -0.424972  0
1 -2.105469  0.567020  1
2 -0.494873  0.276232  2
3  1.072266 -1.087401  3
4  0.721680 -0.673690  4
5 -0.706543  0.113648  5
6 -1.040039 -1.478427  6
7  0.271973  0.524988  7

In [6]: df2.dtypes
Out[6]:
A    float16
B    float64
```

(continues on next page)

(continued from previous page)

```
C      uint8
dtype: object

# here you get some upcasting
In [7]: df3 = df1.reindex_like(df2).fillna(value=0.0) + df2

In [8]: df3
Out[8]:
          A         B    C
0 -0.392704 -0.424972  0.0
1 -2.388332  0.567020  1.0
2 -2.003932  0.276232  2.0
3 -0.063367 -1.087401  3.0
4  1.933792 -0.673690  4.0
5 -0.879758  0.113648  5.0
6 -0.920830 -1.478427  6.0
7 -0.772263  0.524988  7.0

In [9]: df3.dtypes
Out[9]:
A    float32
B    float64
C    float64
dtype: object
```

Dtype conversion

This is lower-common-denominator upcasting, meaning you get the dtype which can accommodate all of the types

```
In [10]: df3.values.dtype
Out[10]: dtype('float64')
```

Conversion

```
In [11]: df3.astype('float32').dtypes
Out[11]:
A    float32
B    float32
C    float32
dtype: object
```

Mixed conversion

```
In [12]: df3['D'] = '1.'

In [13]: df3['E'] = '1'

In [14]: df3.convert_objects(convert_numeric=True).dtypes
Out[14]:
A    float32
B    float64
C    float64
D    float64
E      int64
dtype: object
```

(continues on next page)

(continued from previous page)

```
# same, but specific dtype conversion
In [15]: df3['D'] = df3['D'].astype('float16')

In [16]: df3['E'] = df3['E'].astype('int32')

In [17]: df3.dtypes
Out[17]:
A    float32
B    float64
C    float64
D    float16
E      int32
dtype: object
```

Forcing date coercion (and setting `NaT` when not datelike)

```
In [18]: import datetime

In [19]: s = pd.Series([datetime.datetime(2001, 1, 1, 0, 0), 'foo', 1.0, 1,
   ....:                pd.Timestamp('20010104'), '20010105'], dtype='O')
   ....:

In [20]: s.convert_objects(convert_dates='coerce')
Out[20]:
0   2001-01-01
1          NaT
2          NaT
3          NaT
4   2001-01-04
5   2001-01-05
dtype: datetime64[ns]
```

Dtype gotchas

Platform gotchas

Starting in 0.11.0, construction of DataFrame/Series will use default dtypes of `int64` and `float64`, *regardless of platform*. This is not an apparent change from earlier versions of pandas. If you specify dtypes, they *WILL* be respected, however (GH2837)

The following will all result in `int64` dtypes

```
In [21]: pd.DataFrame([1, 2], columns=['a']).dtypes
Out[21]:
a    int64
dtype: object

In [22]: pd.DataFrame({'a': [1, 2]}).dtypes
Out[22]:
a    int64
dtype: object

In [23]: pd.DataFrame({'a': 1}, index=range(2)).dtypes
Out[23]:
```

(continues on next page)

(continued from previous page)

```
a     int64
dtype: object
```

Keep in mind that `DataFrame(np.array([1,2]))` **WILL** result in `int32` on 32-bit platforms!

Upcasting gotchas

Performing indexing operations on integer type data can easily upcast the data. The dtype of the input data will be preserved in cases where `nans` are not introduced.

```
In [24]: dfi = df3.astype('int32')

In [25]: dfi['D'] = dfi['D'].astype('int64')

In [26]: dfi
Out[26]:
   A  B  C  D  E
0  0  0  0  1  1
1 -2  0  1  1  1
2 -2  0  2  1  1
3  0 -1  3  1  1
4  1  0  4  1  1
5  0  0  5  1  1
6  0 -1  6  1  1
7  0  0  7  1  1

In [27]: dfi.dtypes
Out[27]:
A     int32
B     int32
C     int32
D     int64
E     int32
dtype: object

In [28]: casted = dfi[dfi > 0]

In [29]: casted
Out[29]:
     A   B    C  D  E
0  NaN NaN  NaN  1  1
1  NaN NaN  1.0  1  1
2  NaN NaN  2.0  1  1
3  NaN NaN  3.0  1  1
4  1.0 NaN  4.0  1  1
5  NaN NaN  5.0  1  1
6  NaN NaN  6.0  1  1
7  NaN NaN  7.0  1  1

In [30]: casted.dtypes
Out[30]:
A     float64
B     float64
C     float64
D       int64
E       int32
dtype: object
```

While float dtypes are unchanged.

```
In [31]: df4 = df3.copy()

In [32]: df4['A'] = df4['A'].astype('float32')

In [33]: df4.dtypes
Out[33]:
A    float32
B    float64
C    float64
D    float16
E      int32
dtype: object

In [34]: casted = df4[df4 > 0]

In [35]: casted
Out[35]:
          A         B    C    D  E
0       NaN       NaN  NaN  1.0  1
1       NaN  0.567020  1.0  1.0  1
2       NaN  0.276232  2.0  1.0  1
3       NaN       NaN  3.0  1.0  1
4  1.933792       NaN  4.0  1.0  1
5       NaN  0.113648  5.0  1.0  1
6       NaN       NaN  6.0  1.0  1
7       NaN  0.524988  7.0  1.0  1

In [36]: casted.dtypes
Out[36]:
A    float32
B    float64
C    float64
D    float16
E      int32
dtype: object
```

Datetimes conversion

Datetime64[ns] columns in a DataFrame (or a Series) allow the use of `np.nan` to indicate a nan value, in addition to the traditional `NaT`, or not-a-time. This allows convenient nan setting in a generic way. Furthermore `datetime64[ns]` columns are created by default, when passed datetimelike objects (*this change was introduced in 0.10.1*) (GH2809, GH2810)

```
In [12]: df = pd.DataFrame(np.random.randn(6, 2), pd.date_range('20010102',␣
↪periods=6),
   ....:                   columns=['A', ' B'])
   ....:

In [13]: df['timestamp'] = pd.Timestamp('20010103')

In [14]: df
Out[14]:
                   A         B  timestamp
2001-01-02  0.404705  0.577046 2001-01-03
```

(continues on next page)

(continued from previous page)

```
2001-01-03 -1.715002 -1.039268 2001-01-03
2001-01-04 -0.370647 -1.157892 2001-01-03
2001-01-05 -1.344312  0.844885 2001-01-03
2001-01-06  1.075770 -0.109050 2001-01-03
2001-01-07  1.643563 -1.469388 2001-01-03

# datetime64[ns] out of the box
In [15]: df.dtypes.value_counts()
Out[15]:
float64           2
datetime64[ns]    1
dtype: int64

# use the traditional nan, which is mapped to NaT internally
In [16]: df.loc[df.index[2:4], ['A', 'timestamp']] = np.nan

In [17]: df
Out[17]:
                   A         B  timestamp
2001-01-02  0.404705  0.577046 2001-01-03
2001-01-03 -1.715002 -1.039268 2001-01-03
2001-01-04       NaN -1.157892        NaT
2001-01-05       NaN  0.844885        NaT
2001-01-06  1.075770 -0.109050 2001-01-03
2001-01-07  1.643563 -1.469388 2001-01-03
```

Astype conversion on `datetime64[ns]` to `object`, implicitly converts `NaT` to `np.nan`

```
In [18]: s = pd.Series([datetime.datetime(2001, 1, 2, 0, 0) for i in range(3)])

In [19]: s.dtype
Out[19]: dtype('<M8[ns]')

In [20]: s[1] = np.nan

In [21]: s
Out[21]:
0   2001-01-02
1          NaT
2   2001-01-02
dtype: datetime64[ns]

In [22]: s.dtype
Out[22]: dtype('<M8[ns]')

In [23]: s = s.astype('O')

In [24]: s
Out[24]:
0    2001-01-02 00:00:00
1                    NaT
2    2001-01-02 00:00:00
dtype: object

In [25]: s.dtype
Out[25]: dtype('O')
```

API changes

- Added to_series() method to indices, to facilitate the creation of indexers (GH3275)
- HDFStore
  - added the method select_column to select a single column from a table as a Series.
  - deprecated the unique method, can be replicated by select_column(key,column).unique()
  - min_itemsize parameter to append will now automatically create data_columns for passed keys

Enhancements

- Improved performance of df.to_csv() by up to 10x in some cases. (GH3059)
- Numexpr is now a *Recommended Dependencies*, to accelerate certain types of numerical and boolean operations
- Bottleneck is now a *Recommended Dependencies*, to accelerate certain types of nan operations
- HDFStore
  - support read_hdf/to_hdf API similar to read_csv/to_csv

```
In [26]: df = pd.DataFrame({'A': range(5), 'B': range(5)})

In [27]: df.to_hdf('store.h5', 'table', append=True)

In [28]: pd.read_hdf('store.h5', 'table', where=['index > 2'])
Out[28]:
   A  B
3  3  3
4  4  4
```

  - provide dotted attribute access to get from stores, e.g. store.df == store['df']
  - new keywords iterator=boolean, and chunksize=number_in_a_chunk are provided to support iteration on select and select_as_multiple (GH3076)
- You can now select timestamps from an *unordered* timeseries similarly to an *ordered* timeseries (GH2437)
- You can now select with a string from a DataFrame with a datelike index, in a similar way to a Series (GH3070)

```
In [29]: idx = pd.date_range("2001-10-1", periods=5, freq='M')

In [30]: ts = pd.Series(np.random.rand(len(idx)), index=idx)

In [31]: ts['2001']
Out[31]:
2001-10-31    0.117967
2001-11-30    0.702184
2001-12-31    0.414034
Freq: M, dtype: float64

In [32]: df = pd.DataFrame({'A': ts})

In [33]: df['2001']
Out[33]:
                   A
2001-10-31  0.117967
```

(continues on next page)

(continued from previous page)

```
2001-11-30  0.702184
2001-12-31  0.414034
```

- `Squeeze` to possibly remove length 1 dimensions from an object.

```
>>> p = pd.Panel(np.random.randn(3, 4, 4), items=['ItemA', 'ItemB', 'ItemC'],
...              major_axis=pd.date_range('20010102', periods=4),
...              minor_axis=['A', 'B', 'C', 'D'])
>>> p
<class 'pandas.core.panel.Panel'>
Dimensions: 3 (items) x 4 (major_axis) x 4 (minor_axis)
Items axis: ItemA to ItemC
Major_axis axis: 2001-01-02 00:00:00 to 2001-01-05 00:00:00
Minor_axis axis: A to D

>>> p.reindex(items=['ItemA']).squeeze()
                   A         B         C         D
2001-01-02  0.926089 -2.026458  0.501277 -0.204683
2001-01-03 -0.076524  1.081161  1.141361  0.479243
2001-01-04  0.641817 -0.185352  1.824568  0.809152
2001-01-05  0.575237  0.669934  1.398014 -0.399338

>>> p.reindex(items=['ItemA'], minor=['B']).squeeze()
2001-01-02   -2.026458
2001-01-03    1.081161
2001-01-04   -0.185352
2001-01-05    0.669934
Freq: D, Name: B, dtype: float64
```

- In `pd.io.data.Options`,
  - Fix bug when trying to fetch data for the current month when already past expiry.
  - Now using lxml to scrape html instead of BeautifulSoup (lxml was faster).
  - New instance variables for calls and puts are automatically created when a method that creates them is called. This works for current month where the instance variables are simply `calls` and `puts`. Also works for future expiry months and save the instance variable as `callsMMYY` or `putsMMYY`, where `MMYY` are, respectively, the month and year of the option's expiry.
  - `Options.get_near_stock_price` now allows the user to specify the month for which to get relevant options data.
  - `Options.get_forward_data` now has optional kwargs `near` and `above_below`. This allows the user to specify if they would like to only return forward looking data for options near the current stock price. This just obtains the data from Options.get_near_stock_price instead of Options.get_xxx_data() (GH2758).
- Cursor coordinate information is now displayed in time-series plots.
- added option *display.max_seq_items* to control the number of elements printed per sequence pprinting it. (GH2979)
- added option *display.chop_threshold* to control display of small numerical values. (GH2739)
- added option *display.max_info_rows* to prevent verbose_info from being calculated for frames above 1M rows (configurable). (GH2807, GH2918)
- value_counts() now accepts a "normalize" argument, for normalized histograms. (GH2710).

- DataFrame.from_records now accepts not only dicts but any instance of the collections.Mapping ABC.
- added option *display.mpl_style* providing a sleeker visual style for plots. Based on https://gist.github.com/huyng/816622 (GH3075).
- Treat boolean values as integers (values 1 and 0) for numeric operations. (GH2641)
- to_html() now accepts an optional "escape" argument to control reserved HTML character escaping (enabled by default) and escapes `&`, in addition to < and >. (GH2919)

See the *full release notes* or issue tracker on GitHub for a complete list.

Contributors

A total of 50 people contributed patches to this release. People with a "+" by their names contributed a patch for the first time.

- Adam Greenhall +
- Alvaro Tejero-Cantero +
- Andy Hayden
- Brad Buran +
- Chang She
- Chapman Siu +
- Chris Withers +
- Christian Geier +
- Christopher Whelan
- Damien Garaud
- Dan Birken
- Dan Davison +
- Dieter Vandenbussche
- Drazen Lucanin +
- Dražen Lučanin +
- Garrett Drapala
- Illia Polosukhin +
- James Casbon +
- Jeff Reback
- Jeremy Wagner +
- Jonathan Chambers +
- K.-Michael Aye
- Karmel Allison +
- Loïc Estève +
- Nicholaus E. Halecky +
- Peter Prettenhofer +

- Phillip Cloud +
- Robert Gieseke +
- Skipper Seabold
- Spencer Lyon
- Stephen Lin +
- Thierry Moisan +
- Thomas Kluyver
- Tim Akinbo +
- Vytautas Jancauskas
- Vytautas Jančauskas +
- Wes McKinney
- Will Furnass +
- Wouter Overmeire
- anomrake +
- davidjameshumphreys +
- dengemann +
- dieterv77 +
- jreback
- lexual +
- stephenwlin +
- thauck +
- vytas +
- waitingkuo +
- y-p

5.17 Version 0.10

5.17.1 v0.10.1 (January 22, 2013)

This is a minor release from 0.10.0 and includes new features, enhancements, and bug fixes. In particular, there is substantial new HDFStore functionality contributed by Jeff Reback.

An undesired API breakage with functions taking the `inplace` option has been reverted and deprecation warnings added.

API changes

- Functions taking an `inplace` option return the calling object as before. A deprecation message has been added
- Groupby aggregations Max/Min no longer exclude non-numeric data (GH2700)
- Resampling an empty DataFrame now returns an empty DataFrame instead of raising an exception (GH2640)
- The file reader will now raise an exception when NA values are found in an explicitly specified integer column instead of converting the column to float (GH2631)
- DatetimeIndex.unique now returns a DatetimeIndex with the same name and
- timezone instead of an array (GH2563)

New features

- MySQL support for database (contribution from Dan Allan)

HDFStore

You may need to upgrade your existing data files. Please visit the **compatibility** section in the main docs.

You can designate (and index) certain columns that you want to be able to perform queries on a table, by passing a list to `data_columns`

```
In [1]: store = pd.HDFStore('store.h5')

In [2]: df = pd.DataFrame(np.random.randn(8, 3),
   ...:                   index=pd.date_range('1/1/2000', periods=8),
   ...:                   columns=['A', 'B', 'C'])
   ...:

In [3]: df['string'] = 'foo'

In [4]: df.loc[df.index[4:6], 'string'] = np.nan

In [5]: df.loc[df.index[7:9], 'string'] = 'bar'

In [6]: df['string2'] = 'cool'

In [7]: df
Out[7]:
                   A         B         C string string2
2000-01-01  0.469112 -0.282863 -1.509059    foo    cool
2000-01-02 -1.135632  1.212112 -0.173215    foo    cool
2000-01-03  0.119209 -1.044236 -0.861849    foo    cool
2000-01-04 -2.104569 -0.494929  1.071804    foo    cool
2000-01-05  0.721555 -0.706771 -1.039575    NaN    cool
2000-01-06  0.271860 -0.424972  0.567020    NaN    cool
2000-01-07  0.276232 -1.087401 -0.673690    foo    cool
2000-01-08  0.113648 -1.478427  0.524988    bar    cool

# on-disk operations
In [8]: store.append('df', df, data_columns=['B', 'C', 'string', 'string2'])

In [9]: store.select('df', "B>0 and string=='foo'")
Out[9]:
```

(continues on next page)

(continued from previous page)

```
                   A         B         C string string2
2000-01-02 -1.135632  1.212112 -0.173215    foo    cool

# this is in-memory version of this type of selection
In [10]: df[(df.B > 0) & (df.string == 'foo')]
Out[10]:
                   A         B         C string string2
2000-01-02 -1.135632  1.212112 -0.173215    foo    cool
```

Retrieving unique values in an indexable or data column.

```
# note that this is deprecated as of 0.14.0
# can be replicated by: store.select_column('df','index').unique()
store.unique('df', 'index')
store.unique('df', 'string')
```

You can now store `datetime64` in data columns

```
In [11]: df_mixed = df.copy()

In [12]: df_mixed['datetime64'] = pd.Timestamp('20010102')

In [13]: df_mixed.loc[df_mixed.index[3:4], ['A', 'B']] = np.nan

In [14]: store.append('df_mixed', df_mixed)

In [15]: df_mixed1 = store.select('df_mixed')

In [16]: df_mixed1
Out[16]:
                   A         B         C string string2 datetime64
2000-01-01  0.469112 -0.282863 -1.509059    foo    cool 2001-01-02
2000-01-02 -1.135632  1.212112 -0.173215    foo    cool 2001-01-02
2000-01-03  0.119209 -1.044236 -0.861849    foo    cool 2001-01-02
2000-01-04       NaN       NaN  1.071804    foo    cool 2001-01-02
2000-01-05  0.721555 -0.706771 -1.039575    NaN    cool 2001-01-02
2000-01-06  0.271860 -0.424972  0.567020    NaN    cool 2001-01-02
2000-01-07  0.276232 -1.087401 -0.673690    foo    cool 2001-01-02
2000-01-08  0.113648 -1.478427  0.524988    bar    cool 2001-01-02

In [17]: df_mixed1.dtypes.value_counts()
Out[17]:
float64           3
object            2
datetime64[ns]    1
dtype: int64
```

You can pass `columns` keyword to select to filter a list of the return columns, this is equivalent to passing a `Term('columns',list_of_columns_to_filter)`

```
In [18]: store.select('df', columns=['A', 'B'])
Out[18]:
                   A         B
2000-01-01  0.469112 -0.282863
2000-01-02 -1.135632  1.212112
2000-01-03  0.119209 -1.044236
```

(continues on next page)

(continued from previous page)

```
2000-01-04 -2.104569 -0.494929
2000-01-05  0.721555 -0.706771
2000-01-06  0.271860 -0.424972
2000-01-07  0.276232 -1.087401
2000-01-08  0.113648 -1.478427
```

HDFStore now serializes MultiIndex dataframes when appending tables.

```
In [19]: index = pd.MultiIndex(levels=[['foo', 'bar', 'baz', 'qux'],
   ....:                               ['one', 'two', 'three']],
   ....:                       labels=[[0, 0, 0, 1, 1, 2, 2, 3, 3, 3],
   ....:                               [0, 1, 2, 0, 1, 1, 2, 0, 1, 2]],
   ....:                       names=['foo', 'bar'])
   ....:

In [20]: df = pd.DataFrame(np.random.randn(10, 3), index=index,
   ....:                   columns=['A', 'B', 'C'])
   ....:

In [21]: df
Out[21]:
                  A         B         C
foo bar
foo one   -0.116619  0.295575 -1.047704
    two    1.640556  1.905836  2.772115
    three  0.088787 -1.144197 -0.633372
bar one    0.925372 -0.006438 -0.820408
    two   -0.600874 -1.039266  0.824758
baz two   -0.824095 -0.337730 -0.927764
    three -0.840123  0.248505 -0.109250
qux one    0.431977 -0.460710  0.336505
    two   -3.207595 -1.535854  0.409769
    three -0.673145 -0.741113 -0.110891

In [22]: store.append('mi', df)

In [23]: store.select('mi')
Out[23]:
                  A         B         C
foo bar
foo one   -0.116619  0.295575 -1.047704
    two    1.640556  1.905836  2.772115
    three  0.088787 -1.144197 -0.633372
bar one    0.925372 -0.006438 -0.820408
    two   -0.600874 -1.039266  0.824758
baz two   -0.824095 -0.337730 -0.927764
    three -0.840123  0.248505 -0.109250
qux one    0.431977 -0.460710  0.336505
    two   -3.207595 -1.535854  0.409769
    three -0.673145 -0.741113 -0.110891

# the levels are automatically included as data columns
In [24]: store.select('mi', "foo='bar'")
Out[24]:
                A         B         C
foo bar
bar one  0.925372 -0.006438 -0.820408
```

(continues on next page)

(continued from previous page)

```
   two -0.600874 -1.039266  0.824758
```

Multi-table creation via `append_to_multiple` and selection via `select_as_multiple` can create/select from multiple tables and return a combined result, by using `where` on a selector table.

```
In [19]: df_mt = pd.DataFrame(np.random.randn(8, 6),
   ....:                      index=pd.date_range('1/1/2000', periods=8),
   ....:                      columns=['A', 'B', 'C', 'D', 'E', 'F'])
   ....:

In [20]: df_mt['foo'] = 'bar'

# you can also create the tables individually
In [21]: store.append_to_multiple({'df1_mt': ['A', 'B'], 'df2_mt': None},
   ....:                          df_mt, selector='df1_mt')
   ....:

In [22]: store
Out[22]:
<class 'pandas.io.pytables.HDFStore'>
File path: store.h5

# individual tables were created
In [23]: store.select('df1_mt')
Out[23]:
                   A         B
2000-01-01  0.404705  0.577046
2000-01-02 -1.344312  0.844885
2000-01-03  0.357021 -0.674600
2000-01-04  0.276662 -0.472035
2000-01-05  0.895717  0.805244
2000-01-06 -1.170299 -0.226169
2000-01-07 -0.076467 -1.187678
2000-01-08  1.024180  0.569605

In [24]: store.select('df2_mt')
Out[24]:
                   C         D         E         F  foo
2000-01-01 -1.715002 -1.039268 -0.370647 -1.157892  bar
2000-01-02  1.075770 -0.109050  1.643563 -1.469388  bar
2000-01-03 -1.776904 -0.968914 -1.294524  0.413738  bar
2000-01-04 -0.013960 -0.362543 -0.006154 -0.923061  bar
2000-01-05 -1.206412  2.565646  1.431256  1.340309  bar
2000-01-06  0.410835  0.813850  0.132003 -0.827317  bar
2000-01-07  1.130127 -1.436737 -1.413681  1.607920  bar
2000-01-08  0.875906 -2.211372  0.974466 -2.006747  bar

# as a multiple
In [25]: store.select_as_multiple(['df1_mt', 'df2_mt'], where=['A>0', 'B>0'],
   ....:                          selector='df1_mt')
   ....:
Out[25]:
                   A         B         C         D         E         F  foo
2000-01-01  0.404705  0.577046 -1.715002 -1.039268 -0.370647 -1.157892  bar
2000-01-05  0.895717  0.805244 -1.206412  2.565646  1.431256  1.340309  bar
2000-01-08  1.024180  0.569605  0.875906 -2.211372  0.974466 -2.006747  bar
```

Enhancements

- `HDFStore` now can read native PyTables table format tables
- You can pass `nan_rep = 'my_nan_rep'` to append, to change the default nan representation on disk (which converts to/from *np.nan*), this defaults to *nan*.
- You can pass `index` to `append`. This defaults to `True`. This will automagically create indices on the *indexables* and *data columns* of the table
- You can pass `chunksize=an integer` to `append`, to change the writing chunksize (default is 50000). This will significantly lower your memory usage on writing.
- You can pass `expectedrows=an integer` to the first `append`, to set the TOTAL number of expected rows that `PyTables` will expected. This will optimize read/write performance.
- `Select` now supports passing `start` and `stop` to provide selection space limiting in selection.
- Greatly improved ISO8601 (e.g., yyyy-mm-dd) date parsing for file parsers (GH2698)
- Allow `DataFrame.merge` to handle combinatorial sizes too large for 64-bit integer (GH2690)
- Series now has unary negation (-series) and inversion (~series) operators (GH2686)
- DataFrame.plot now includes a `logx` parameter to change the x-axis to log scale (GH2327)
- Series arithmetic operators can now handle constant and ndarray input (GH2574)
- ExcelFile now takes a `kind` argument to specify the file type (GH2613)
- A faster implementation for Series.str methods (GH2602)

Bug Fixes

- `HDFStore` tables can now store `float32` types correctly (cannot be mixed with `float64` however)
- Fixed Google Analytics prefix when specifying request segment (GH2713).
- Function to reset Google Analytics token store so users can recover from improperly setup client secrets (GH2687).
- Fixed groupby bug resulting in segfault when passing in MultiIndex (GH2706)
- Fixed bug where passing a Series with datetime64 values into *to_datetime* results in bogus output values (GH2699)
- Fixed bug in `pattern in HDFStore` expressions when pattern is not a valid regex (GH2694)
- Fixed performance issues while aggregating boolean data (GH2692)
- When given a boolean mask key and a Series of new values, Series __setitem__ will now align the incoming values with the original Series (GH2686)
- Fixed MemoryError caused by performing counting sort on sorting MultiIndex levels with a very large number of combinatorial values (GH2684)
- Fixed bug that causes plotting to fail when the index is a DatetimeIndex with a fixed-offset timezone (GH2683)
- Corrected business day subtraction logic when the offset is more than 5 bdays and the starting date is on a weekend (GH2680)
- Fixed C file parser behavior when the file has more columns than data (GH2668)
- Fixed file reader bug that misaligned columns with data in the presence of an implicit column and a specified *usecols* value
- DataFrames with numerical or datetime indices are now sorted prior to plotting (GH2609)

- Fixed DataFrame.from_records error when passed columns, index, but empty records (GH2633)
- Several bug fixed for Series operations when dtype is datetime64 (GH2689, GH2629, GH2626)

See the *full release notes* or issue tracker on GitHub for a complete list.

Contributors

A total of 17 people contributed patches to this release. People with a "+" by their names contributed a patch for the first time.

- Andy Hayden +
- Anton I. Sipos +
- Chang She
- Christopher Whelan
- Damien Garaud +
- Dan Allan +
- Dieter Vandenbussche
- Garrett Drapala +
- Jay Parlar +
- Thouis (Ray) Jones +
- Vincent Arel-Bundock +
- Wes McKinney
- elpres
- herrfz +
- jreback
- svaksha +
- y-p

5.17.2 v0.10.0 (December 17, 2012)

This is a major release from 0.9.1 and includes many new features and enhancements along with a large number of bug fixes. There are also a number of important API changes that long-time pandas users should pay close attention to.

File parsing new features

The delimited file parsing engine (the guts of `read_csv` and `read_table`) has been rewritten from the ground up and now uses a fraction the amount of memory while parsing, while being 40% or more faster in most use cases (in some cases much faster).

There are also many new features:

- Much-improved Unicode handling via the `encoding` option.
- Column filtering (`usecols`)

- Dtype specification (`dtype` argument)
- Ability to specify strings to be recognized as True/False
- Ability to yield NumPy record arrays (`as_recarray`)
- High performance `delim_whitespace` option
- Decimal format (e.g. European format) specification
- Easier CSV dialect options: `escapechar`, `lineterminator`, `quotechar`, etc.
- More robust handling of many exceptional kinds of files observed in the wild

API changes

Deprecated DataFrame BINOP TimeSeries special case behavior

The default behavior of binary operations between a DataFrame and a Series has always been to align on the DataFrame's columns and broadcast down the rows, **except** in the special case that the DataFrame contains time series. Since there are now method for each binary operator enabling you to specify how you want to broadcast, we are phasing out this special case (Zen of Python: *Special cases aren't special enough to break the rules*). Here's what I'm talking about:

```
In [1]: import pandas as pd

In [2]: df = pd.DataFrame(np.random.randn(6, 4),
   ...:                   index=pd.date_range('1/1/2000', periods=6))
   ...:

In [3]: df
Out[3]:
                   0         1         2         3
2000-01-01  0.469112 -0.282863 -1.509059 -1.135632
2000-01-02  1.212112 -0.173215  0.119209 -1.044236
2000-01-03 -0.861849 -2.104569 -0.494929  1.071804
2000-01-04  0.721555 -0.706771 -1.039575  0.271860
2000-01-05 -0.424972  0.567020  0.276232 -1.087401
2000-01-06 -0.673690  0.113648 -1.478427  0.524988

# deprecated now
In [4]: df - df[0]
Out[4]:
            2000-01-01 00:00:00  2000-01-02 00:00:00  2000-01-03 00:00:00  2000-01-04␣
↪00:00:00  ...   0   1   2   3
2000-01-01                  NaN                  NaN                  NaN                  ␣
↪      NaN  ... NaN NaN NaN NaN
2000-01-02                  NaN                  NaN                  NaN                  ␣
↪      NaN  ... NaN NaN NaN NaN
2000-01-03                  NaN                  NaN                  NaN                  ␣
↪      NaN  ... NaN NaN NaN NaN
2000-01-04                  NaN                  NaN                  NaN                  ␣
↪      NaN  ... NaN NaN NaN NaN
2000-01-05                  NaN                  NaN                  NaN                  ␣
↪      NaN  ... NaN NaN NaN NaN
2000-01-06                  NaN                  NaN                  NaN                  ␣
↪      NaN  ... NaN NaN NaN NaN

[6 rows x 10 columns]
```

(continues on next page)

(continued from previous page)

```
# Change your code to
In [5]: df.sub(df[0], axis=0)  # align on axis 0 (rows)
Out[5]:
              0         1         2         3
2000-01-01  0.0 -0.751976 -1.978171 -1.604745
2000-01-02  0.0 -1.385327 -1.092903 -2.256348
2000-01-03  0.0 -1.242720  0.366920  1.933653
2000-01-04  0.0 -1.428326 -1.761130 -0.449695
2000-01-05  0.0  0.991993  0.701204 -0.662428
2000-01-06  0.0  0.787338 -0.804737  1.198677
```

You will get a deprecation warning in the 0.10.x series, and the deprecated functionality will be removed in 0.11 or later.

Altered resample default behavior

The default time series `resample` binning behavior of daily `D` and *higher* frequencies has been changed to `closed='left', label='left'`. Lower nfrequencies are unaffected. The prior defaults were causing a great deal of confusion for users, especially resampling data to daily frequency (which labeled the aggregated group with the end of the interval: the next day).

```
In [1]: dates = pd.date_range('1/1/2000', '1/5/2000', freq='4h')

In [2]: series = pd.Series(np.arange(len(dates)), index=dates)

In [3]: series
Out[3]:
2000-01-01 00:00:00     0
2000-01-01 04:00:00     1
2000-01-01 08:00:00     2
2000-01-01 12:00:00     3
2000-01-01 16:00:00     4
2000-01-01 20:00:00     5
2000-01-02 00:00:00     6
2000-01-02 04:00:00     7
2000-01-02 08:00:00     8
2000-01-02 12:00:00     9
2000-01-02 16:00:00    10
2000-01-02 20:00:00    11
2000-01-03 00:00:00    12
2000-01-03 04:00:00    13
2000-01-03 08:00:00    14
2000-01-03 12:00:00    15
2000-01-03 16:00:00    16
2000-01-03 20:00:00    17
2000-01-04 00:00:00    18
2000-01-04 04:00:00    19
2000-01-04 08:00:00    20
2000-01-04 12:00:00    21
2000-01-04 16:00:00    22
2000-01-04 20:00:00    23
2000-01-05 00:00:00    24
Freq: 4H, dtype: int64

In [4]: series.resample('D', how='sum')
Out[4]:
```

(continues on next page)

(continued from previous page)

```
2000-01-01     15
2000-01-02     51
2000-01-03     87
2000-01-04    123
2000-01-05     24
Freq: D, dtype: int64

In [5]: # old behavior
In [6]: series.resample('D', how='sum', closed='right', label='right')
Out[6]:
2000-01-01      0
2000-01-02     21
2000-01-03     57
2000-01-04     93
2000-01-05    129
Freq: D, dtype: int64
```

- Infinity and negative infinity are no longer treated as NA by `isnull` and `notnull`. That they ever were was a relic of early pandas. This behavior can be re-enabled globally by the `mode.use_inf_as_null` option:

```
In [6]: s = pd.Series([1.5, np.inf, 3.4, -np.inf])

In [7]: pd.isnull(s)
Out[7]:
0    False
1    False
2    False
3    False
Length: 4, dtype: bool

In [8]: s.fillna(0)
Out[8]:
0    1.500000
1         inf
2    3.400000
3        -inf
Length: 4, dtype: float64

In [9]: pd.set_option('use_inf_as_null', True)

In [10]: pd.isnull(s)
Out[10]:
0    False
1     True
2    False
3     True
Length: 4, dtype: bool

In [11]: s.fillna(0)
Out[11]:
0    1.5
1    0.0
2    3.4
3    0.0
Length: 4, dtype: float64

In [12]: pd.reset_option('use_inf_as_null')
```

- Methods with the `inplace` option now all return `None` instead of the calling object. E.g. code written like `df = df.fillna(0, inplace=True)` may stop working. To fix, simply delete the unnecessary variable assignment.
- `pandas.merge` no longer sorts the group keys (`sort=False`) by default. This was done for performance reasons: the group-key sorting is often one of the more expensive parts of the computation and is often unnecessary.
- The default column names for a file with no header have been changed to the integers `0` through `N - 1`. This is to create consistency with the DataFrame constructor with no columns specified. The v0.9.0 behavior (names `X0`, `X1`, ...) can be reproduced by specifying `prefix='X'`:

```
In [6]: import io

In [7]: data = ('a,b,c\n'
   ...:         '1,Yes,2\n'
   ...:         '3,No,4')
   ...:

In [8]: print(data)
a,b,c
1,Yes,2
3,No,4

In [9]: pd.read_csv(io.StringIO(data), header=None)
Out[9]:
   0    1  2
0  a    b  c
1  1  Yes  2
2  3   No  4

In [10]: pd.read_csv(io.StringIO(data), header=None, prefix='X')
Out[10]:
  X0   X1 X2
0  a    b  c
1  1  Yes  2
2  3   No  4
```

- Values like `'Yes'` and `'No'` are not interpreted as boolean by default, though this can be controlled by new `true_values` and `false_values` arguments:

```
In [11]: print(data)
a,b,c
1,Yes,2
3,No,4

In [12]: pd.read_csv(io.StringIO(data))
Out[12]:
   a    b  c
0  1  Yes  2
1  3   No  4

In [13]: pd.read_csv(io.StringIO(data), true_values=['Yes'], false_values=['No'])
Out[13]:
   a      b  c
0  1   True  2
1  3  False  4
```

- The file parsers will not recognize non-string values arising from a converter function as NA if passed in the na_values argument. It's better to do post-processing using the replace function instead.

- Calling fillna on Series or DataFrame with no arguments is no longer valid code. You must either specify a fill value or an interpolation method:

```
In [14]: s = pd.Series([np.nan, 1., 2., np.nan, 4])

In [15]: s
Out[15]:
0    NaN
1    1.0
2    2.0
3    NaN
4    4.0
dtype: float64

In [16]: s.fillna(0)
Out[16]:
0    0.0
1    1.0
2    2.0
3    0.0
4    4.0
dtype: float64

In [17]: s.fillna(method='pad')
Out[17]:
0    NaN
1    1.0
2    2.0
3    2.0
4    4.0
dtype: float64
```

Convenience methods ffill and bfill have been added:

```
In [18]: s.ffill()
Out[18]:
0    NaN
1    1.0
2    2.0
3    2.0
4    4.0
dtype: float64
```

- Series.apply will now operate on a returned value from the applied function, that is itself a series, and possibly upcast the result to a DataFrame

```
In [19]: def f(x):
   ....:     return pd.Series([x, x**2], index=['x', 'x^2'])
   ....:

In [20]: s = pd.Series(np.random.rand(5))

In [21]: s
Out[21]:
0    0.340445
1    0.984729
```

(continues on next page)

(continued from previous page)

```
2    0.919540
3    0.037772
4    0.861549
dtype: float64

In [22]: s.apply(f)
Out[22]:
          x       x^2
0  0.340445  0.115903
1  0.984729  0.969691
2  0.919540  0.845555
3  0.037772  0.001427
4  0.861549  0.742267
```

- New API functions for working with pandas options (GH2097):
  - `get_option` / `set_option` - get/set the value of an option. Partial names are accepted. - `reset_option` - reset one or more options to their default value. Partial names are accepted. - `describe_option` - print a description of one or more options. When called with no arguments. print all registered options.

  Note: `set_printoptions`/ `reset_printoptions` are now deprecated (but functioning), the print options now live under "display.XYZ". For example:

```
In [23]: pd.get_option("display.max_rows")
Out[23]: 15
```

- to_string() methods now always return unicode strings (GH2224).

New features

Wide DataFrame printing

Instead of printing the summary information, pandas now splits the string representation across multiple rows by default:

```
In [24]: wide_frame = pd.DataFrame(np.random.randn(5, 16))

In [25]: wide_frame
Out[25]:
         0         1         2         3         4         5         6  ...         9␣
↪        10        11        12        13        14        15
0 -0.548702  1.467327 -1.015962 -0.483075  1.637550 -1.217659 -0.291519  ...  0.
↪991460 -0.919069  0.266046 -0.709661  1.669052  1.037882 -1.705775
1 -0.919854 -0.042379  1.247642 -0.009920  0.290213  0.495767  0.362949  ... -0.
↪089329  0.337863 -0.945867 -0.932132  1.956030  0.017587 -0.016692
2 -0.575247  0.254161 -1.143704  0.215897  1.193555 -0.077118 -0.408530  ...  1.
↪511763  1.627081 -0.990582 -0.441652  1.211526  0.268520  0.024580
3 -1.577585  0.396823 -0.105381 -0.532532  1.453749  1.208843 -0.080952  ... -0.
↪589346  0.339969 -0.693205 -0.339355  0.593616  0.884345  1.591431
4  0.141809  0.220390  0.435589  0.192451 -0.096701  0.803351  1.715071  ... -1.
↪814470  1.018601 -0.595447  1.395433 -0.392670  0.007207  1.928123

[5 rows x 16 columns]
```

The old behavior of printing out summary information can be achieved via the 'expand_frame_repr' print option: