Pandas Development Roadmap

12/29/15 Initial Creation

This document show some recent changes in the pandas library and some directions for future work. Please add helpful comments / major features that should be considered.

Some previous discussions are here:
Pandas Meeting Notes thru June 2015
Scipy 2015 BOF

Table of contents

- Pandas Development Roadmap
  - Some recent changes in pandas
  - Some on-deck enhancements are:
  - Future Possibilities:
- pandas and NumPy compatibility
- Replacing BlockManager with new C++ data structures
- Missing data support in integer and boolean data
- Interfacing pandas with numba, dask, and other Python projects

Some recent changes in pandas

- dtype enhancements (`Categorical, Timedelta, Datetime` w/tz) & making these first class objects
- code refactoring to remove subclassing of ndarrays for Series & Index
- carving out / deprecating non-core parts of pandas
  - datareader
  - `SparsePanel, WidePanel` & other aliases (`TimeSeries`)
  - rpy, rplot, irow et al.
  - google-analytics
  - removed residual statsmodels type things (`ols, VAR`)
- API changes to make things more consistent
  - `pd.rolling/expanding` * -> `.rolling/expanding` (this is in master now)
  - `.resample` becoming a full deferred like groupby (PR up)
  - multi-index slicing along any level (obviates need for `.xs`) and allows assignment
  - `.loc/.iloc` - for the most part obviates use of `.ix`
  - `.pipe` & `.assign`
  - plotting accessors
  - fixing of the sorting API
- many performance enhancements both micro & macro (e.g. release GIL)

Some on-deck enhancements are:

- `IntervalIndex` (and eventually make `PeriodIndex` just a sub-class of this) https://github.com/pydata/pandas/pull/8707
- `RangeIndex` https://github.com/pydata/pandas/pull/11892

Future Possibilities:

- IO support
  - parquet
  - avro https://github.com/pydata/pandas/issues/11752
  - BSON https://github.com/pydata/pandas/issues/9166 (more of a refactor here)
  - SFrame format?
- what is public API? / do we need lower-level 'dev' API?
- Impl API
  - currently only support an actual numpy API, ideally support a numpy-like API
  - if this is fleshed out a bit, could support memap/bcolz/DyND/SFrame/new c-API, IOW swap out for specific dtypes, providing easy transition
- pandas 1.0
  - Some ideas are here: https://github.com/pydata/pandas/issues/10000
  - Fix []/__getitem__ (#9595)
  - Make the index/column distinction less painful (#5677, #8162)
  - Clean up the Index vs MultiIndex API (#3268)
  - DataFrame constructor reindexes, but every expects overwriting the index instead (everyone finds this confusing)
  - Sparse
    - remove / deprecate?
    - carve to own library?
    - support more? mostly functional, but need champion
  - Panel
    - defer to x-ray? / deprecate?
      - +1 most of the pandas logic just doesn't generalize well beyond 1D/2D (shoyer)
      - +1 I think x-ray is a better approach to ND labeled arrays. (Travis)
      - Some user commentary here: https://github.com/pydata/pandas/issues/8906
    - fix some warts (indexing when reducing dimensions is a bit non-intuitive)
- crazy things that would probably break too much but might be worth thinking about / opinions:
  - enforce immutability in group by apply

- use `numba` for all aggregations instead of python-generated `cython` (would need AOT to avoid explict `numba` dep; this is currently available in `numba >= 0.22`).
    - shoyer: I have some proof of concept for this in numbagg. In my tests, performance is very good.
- view numpy as a necessary evil and *only* compatibility reasons and where that doesn't work we don't bend over backwards to make it work (dynd has a similar view IIUC)
- factory functions for Series & DataFrame constructors
    - WM: This would reduce constructor complexity (to separate internal object construction from user construction) and result in better microperformance. As long as you have a fastpath option for internal use I think it's fine to leave it as is because the API breakage would be too severe.
    - breaks API, what benefit is this?

pandas and NumPy compatibility

Per above, what NumPy compatibility should we actually care about?

Replacing BlockManager with new C++ data structures

- Internals rewrite in c++ (`libpandas`)
    - pros:
        - Allows better micro performance (what exactly are we expecting here)?
        - c-api (potential to create multi-lang bindings)
        - better maintainability (*how is this true?*)
        - New dtypes could be added and treated as first-class citizens rather than glued on
    - cons:
        - this would make the base library *harder* for newcomers to contribute (not that the internals are easy now, but they are mostly in python)
        - this is a fair amount of work, so have to weigh roi
        - not re-using a lot of the low-level work that is already in DyND
- consolidation policy
    - allow `policy=block|column|split`
        - block is backwards compat
        - column is keep blocks separate if passed that way or created (e.g. a dict now will support direct view access), this would be the default
        - split is forced splitting of ndarrays on pass in to dict-like (this can be expensive so its optional)

        - column/split provide superior row aggregations as the cache is fully utilized (as these are contiguous in memory)
        - https://github.com/jreback/pandas/tree/consolidate, not a PR yet (and a bit behind master)
    - related is supporting `copy-on-write` to allow chained indexing to work (rather than either fail silently in older versions of pandas of showing a `SettingWithCopyWarning`, which is a bit unintuitive for users), https://github.com/pydata/pandas/pull/11500
- Adding new data types

Missing data support in integer and boolean data

Decision: depends on BlockManager / libpandas investigation and decision

- Missing value support for `int NA & boolean NA`
    - Use bitfields
        - pros
            - no additional dependencies
            - much easier to understand than DyND for non-c++ wizards
            - allows missing value support for basically any type that we can come up with (though presumably the set of types that are useful isn't that large)
            - precedence in at least one other system (postgres) for doing it this way (not the strongest argument in favor, but evidence that it's a valid solution to the problem)
            - this works with nested data, not sure how sentinels would work with that (dynd keeps an extra bool in the char* indicating whether the value is missing or not but only does this for complex types. it uses sentinels for scalar types)
        - cons
            - probably a fair amount of work up front
            - less easier to understand than DyND for non-bitfield wizards
            - different than using sentinel (like all other dtypes in pandas ATM)
            - need to implement our own `bitarray` or find a good implementation
            - how would this jive with the buffer interface which is presumably the interface we'd target for numpy compatibility
            - numpy compat is very difficult because numpy doesn't support the notion of missing values, though i guess we could go to a masked array (blah)
    - Use `DyND`
        - pros

        - already supports missing values
        - have proof-of-concept impl
        - no more heavyweight than NumPy
        - DyND has explored much of the space here already, has learned from its mistakes
    - cons
        - fairly heavyweight to have as a hard dependency (its currently `pip` (wheels) and `conda` installable, on all platforms, though)
        - DyND needs to flesh out numpy compat.
        - Requires waiting on another project to do something -- we can also collaborate with the DyND developers, they are fairly approachable
        - the way dynd allocates memory is exactly like C and NumPy, meaning any in-memory layout optimizations specific to analytics applications, like using a columnar layout are basically not possible because how memory is allocated is part of the user interface (this is what datashape is). *How is this a con*?

Interfacing pandas with numba, dask, and other Python projects

- defer to `numba` in `.apply`
    - provides easy way to handle UDF's
    - numba is adding extension dtypes so can interoperate with `Series`
    - adding 'real' indexing support within udfs is possible but tricky. would have to extend the Series dtype so that the c-indexing routines could be directly called (e.g. if someone does say `self.iloc[]` within a udf)
    - this would need benchmarking
    - biggest API issue is that we need to handle what happens when the udf goes to `nopython` mode, should we just continue on with regular apply, or raise (need a kw arg for this I think).
    - makes sense to have `engine='numba'` kw arg here (are there other engines we could defer to?)
- interfacing to `dask`
    - via groupby with `engine='dask'` default (if installed) for auto-parallelism
    - `.to_parallel` to simply return a `dask.dataframe` (easy way for people to then parallelize)

Supporting new data types

- Other dtype support (just putting these out there for discussion)

- ragged arrays (e.g. lists in cells)
- json-like
- images
- variable len strings
- pd.String (make all string-likes into an impl of categorical)
- datetimes with non-ns units
- unit support https://github.com/pydata/pandas/issues/10349