- Other Python objects: `'object'`

The `numpy_type` is the physical storage type of the column, which is the result of `str(dtype)` for the underlying NumPy array that holds the data. So for `datetimetz` this is `datetime64[ns]` and for categorical, it may be any of the supported integer categorical types.

The `metadata` field is `None` except for:

- `datetimetz`: `{'timezone': zone, 'unit': 'ns'}`, e.g. `{'timezone', 'America/New_York', 'unit': 'ns'}`. The `'unit'` is optional, and if omitted it is assumed to be nanoseconds.
- `categorical`: `{'num_categories': K, 'ordered': is_ordered, 'type': $TYPE}`
  - Here `'type'` is optional, and can be a nested pandas type specification here (but not categorical)
- `unicode`: `{'encoding': encoding}`
  - The encoding is optional, and if not present is UTF-8
- `object`: `{'encoding': encoding}`. Objects can be serialized and stored in `BYTE_ARRAY` Parquet columns. The encoding can be one of:
  - `'pickle'`
  - `'bson'`
  - `'json'`
- `timedelta`: `{'unit': 'ns'}`. The `'unit'` is optional, and if omitted it is assumed to be nanoseconds. This metadata is optional altogether

For types other than these, the `'metadata'` key can be omitted. Implementations can assume `None` if the key is not present.

As an example of fully-formed metadata:

```
{'index_columns': ['__index_level_0__'],
 'column_indexes': [
     {'name': None,
      'field_name': 'None',
      'pandas_type': 'unicode',
      'numpy_type': 'object',
      'metadata': {'encoding': 'UTF-8'}}
 ],
 'columns': [
     {'name': 'c0',
      'field_name': 'c0',
      'pandas_type': 'int8',
      'numpy_type': 'int8',
      'metadata': None},
     {'name': 'c1',
      'field_name': 'c1',
      'pandas_type': 'bytes',
      'numpy_type': 'object',
      'metadata': None},
     {'name': 'c2',
      'field_name': 'c2',
      'pandas_type': 'categorical',
      'numpy_type': 'int16',
      'metadata': {'num_categories': 1000, 'ordered': False}},
     {'name': 'c3',
      'field_name': 'c3',
      'pandas_type': 'datetimetz',
```

(continues on next page)

(continued from previous page)

```
      'numpy_type': 'datetime64[ns]',
      'metadata': {'timezone': 'America/Los_Angeles'}},
     {'name': 'c4',
      'field_name': 'c4',
      'pandas_type': 'object',
      'numpy_type': 'object',
      'metadata': {'encoding': 'pickle'}},
     {'name': None,
      'field_name': '__index_level_0__',
      'pandas_type': 'int64',
      'numpy_type': 'int64',
      'metadata': None}
 ],
 'pandas_version': '0.20.0',
 'creator': {
   'library': 'pyarrow',
   'version': '0.13.0'
 }}
```

4.7 Policies

4.7.1 Version Policy

Changed in version 1.0.0.

Pandas uses a loose variant of semantic versioning (SemVer) to govern deprecations, API compatibility, and version numbering.

A pandas release number is made up of `MAJOR.MINOR.PATCH`.

API breaking changes should only occur in **major** releases. Theses changes will be documented, with clear guidance on what is changing, why it's changing, and how to migrate existing code to the new behavior.

Whenever possible, a deprecation path will be provided rather than an outright breaking change.

Pandas will introduce deprecations in **minor** releases. These deprecations will preserve the existing behavior while emitting a warning that provide guidance on:

- How to achieve similar behavior if an alternative is available
- The pandas version in which the deprecation will be enforced.

We will not introduce new deprecations in patch releases.

Deprecations will only be enforced in **major** releases. For example, if a behavior is deprecated in pandas 1.2.0, it will continue to work, with a warning, for all releases in the 1.x series. The behavior will change and the deprecation removed in the next next major release (2.0.0).

Note: Pandas will sometimes make *behavior changing* bug fixes, as part of minor or patch releases. Whether or not a change is a bug fix or an API-breaking change is a judgement call. We'll do our best, and we invite you to participate in development discussion on the issue tracker or mailing list.

These policies do not apply to features marked as **experimental** in the documentation. Pandas may change the behavior of experimental features at any time.

4.7.2 Python Support

Pandas will only drop support for specific Python versions (e.g. 3.6.x, 3.7.x) in pandas **major** releases.

4.8 Roadmap

This page provides an overview of the major themes in pandas' development. Each of these items requires a relatively large amount of effort to implement. These may be achieved more quickly with dedicated funding or interest from contributors.

An item being on the roadmap does not mean that it will *necessarily* happen, even with unlimited funding. During the implementation period we may discover issues preventing the adoption of the feature.

Additionally, an item *not* being on the roadmap does not exclude it from inclusion in pandas. The roadmap is intended for larger, fundamental changes to the project that are likely to take months or years of developer time. Smaller-scoped items will continue to be tracked on our issue tracker.

See *Roadmap Evolution* for proposing changes to this document.

4.8.1 Extensibility

Pandas *Extension types* allow for extending NumPy types with custom data types and array storage. Pandas uses extension types internally, and provides an interface for 3rd-party libraries to define their own custom data types.

Many parts of pandas still unintentionally convert data to a NumPy array. These problems are especially pronounced for nested data.

We'd like to improve the handling of extension arrays throughout the library, making their behavior more consistent with the handling of NumPy arrays. We'll do this by cleaning up pandas' internals and adding new methods to the extension array interface.

4.8.2 String data type

Currently, pandas stores text data in an `object` -dtype NumPy array. The current implementation has two primary drawbacks: First, `object` -dtype is not specific to strings: any Python object can be stored in an `object` -dtype array, not just strings. Second: this is not efficient. The NumPy memory model isn't especially well-suited to variable width text data.

To solve the first issue, we propose a new extension type for string data. This will initially be opt-in, with users explicitly requesting `dtype="string"`. The array backing this string dtype may initially be the current implementation: an `object` -dtype NumPy array of Python strings.

To solve the second issue (performance), we'll explore alternative in-memory array libraries (for example, Apache Arrow). As part of the work, we may need to implement certain operations expected by pandas users (for example the algorithm used in, `Series.str.upper`). That work may be done outside of pandas.

4.8.3 Apache Arrow interoperability

Apache Arrow is a cross-language development platform for in-memory data. The Arrow logical types are closely aligned with typical pandas use cases.

We'd like to provide better-integrated support for Arrow memory and data types within pandas. This will let us take advantage of its I/O capabilities and provide for better interoperability with other languages and libraries using Arrow.

4.8.4 Block manager rewrite

We'd like to replace pandas current internal data structures (a collection of 1 or 2-D arrays) with a simpler collection of 1-D arrays.

Pandas internal data model is quite complex. A DataFrame is made up of one or more 2-dimensional "blocks", with one or more blocks per dtype. This collection of 2-D arrays is managed by the BlockManager.

The primary benefit of the BlockManager is improved performance on certain operations (construction from a 2D array, binary operations, reductions across the columns), especially for wide DataFrames. However, the BlockManager substantially increases the complexity and maintenance burden of pandas.

By replacing the BlockManager we hope to achieve

- Substantially simpler code
- Easier extensibility with new logical types
- Better user control over memory use and layout
- Improved micro-performance
- Option to provide a C / Cython API to pandas' internals

See these design documents for more.

4.8.5 Decoupling of indexing and internals

The code for getting and setting values in pandas' data structures needs refactoring. In particular, we must clearly separate code that converts keys (e.g., the argument to `DataFrame.loc`) to positions from code that uses these positions to get or set values. This is related to the proposed BlockManager rewrite. Currently, the BlockManager sometimes uses label-based, rather than position-based, indexing. We propose that it should only work with positional indexing, and the translation of keys to positions should be entirely done at a higher level.

Indexing is a complicated API with many subtleties. This refactor will require care and attention. More details are discussed at https://github.com/pandas-dev/pandas/wiki/(Tentative)-rules-for-restructuring-indexing-code

4.8.6 Numba-accelerated operations

Numba is a JIT compiler for Python code. We'd like to provide ways for users to apply their own Numba-jitted functions where pandas accepts user-defined functions (for example, `Series.apply()`, `DataFrame.apply()`, `DataFrame.applymap()`, and in groupby and window contexts). This will improve the performance of user-defined-functions in these operations by staying within compiled code.

4.8.7 Documentation improvements

We'd like to improve the content, structure, and presentation of the pandas documentation. Some specific goals include

- Overhaul the HTML theme with a modern, responsive design (GH15556)
- Improve the "Getting Started" documentation, designing and writing learning paths for users different backgrounds (e.g. brand new to programming, familiar with other languages like R, already familiar with Python).
- Improve the overall organization of the documentation and specific subsections of the documentation to make navigation and finding content easier.

4.8.8 Package docstring validation

To improve the quality and consistency of pandas docstrings, we've developed tooling to check docstrings in a variety of ways. https://github.com/pandas-dev/pandas/blob/master/scripts/validate_docstrings.py contains the checks.

Like many other projects, pandas uses the numpydoc style for writing docstrings. With the collaboration of the numpydoc maintainers, we'd like to move the checks to a package other than pandas so that other projects can easily use them as well.

4.8.9 Performance monitoring

Pandas uses airspeed velocity to monitor for performance regressions. ASV itself is a fabulous tool, but requires some additional work to be integrated into an open source project's workflow.

The asv-runner organization, currently made up of pandas maintainers, provides tools built on top of ASV. We have a physical machine for running a number of project's benchmarks, and tools managing the benchmark runs and reporting on results.

We'd like to fund improvements and maintenance of these tools to

- Be more stable. Currently, they're maintained on the nights and weekends when a maintainer has free time.
- Tune the system for benchmarks to improve stability, following https://pyperf.readthedocs.io/en/latest/system.html
- Build a GitHub bot to request ASV runs *before* a PR is merged. Currently, the benchmarks are only run nightly.

4.8.10 Roadmap Evolution

Pandas continues to evolve. The direction is primarily determined by community interest. Everyone is welcome to review existing items on the roadmap and to propose a new item.

Each item on the roadmap should be a short summary of a larger design proposal. The proposal should include

1. Short summary of the changes, which would be appropriate for inclusion in the roadmap if accepted.
2. Motivation for the changes.
3. An explanation of why the change is in scope for pandas.
4. Detailed design: Preferably with example-usage (even if not implemented yet) and API documentation
5. API Change: Any API changes that may result from the proposal.

That proposal may then be submitted as a GitHub issue, where the pandas maintainers can review and comment on the design. The pandas mailing list should be notified of the proposal.

When there's agreement that an implementation would be welcome, the roadmap should be updated to include the summary and a link to the discussion issue.

4.9 Developer Meetings

We hold regular developer meetings on the second Wednesday of each month at 18:00 UTC. These meetings and their minutes are open to the public. All are welcome to join.

4.9.1 Minutes

The minutes of past meetings are available in this Google Document.

4.9.2 Calendar

This calendar shows all the developer meetings.

You can subscribe to this calendar with the following links:

- iCal
- Google calendar

Additionally, we'll sometimes have one-off meetings on specific topics. These will be published on the same calendar.

CHAPTER

FIVE

RELEASE NOTES

This is the list of changes to pandas between each release. For full details, see the commit logs at http://github.com/pandas-dev/pandas. For install and upgrade instructions, see *Installation*.

5.1 Version 1.0

5.1.1 What's new in 1.0.5 (June 17, 2020)

These are the changes in pandas 1.0.5. See *Release Notes* for a full changelog including other versions of pandas.

Fixed regressions

- Fix regression in `read_parquet()` when reading from file-like objects (GH34467).
- Fix regression in reading from public S3 buckets (GH34626).

Note this disables the ability to read Parquet files from directories on S3 again (GH26388, GH34632), which was added in the 1.0.4 release, but is now targeted for pandas 1.1.0.

- Fixed regression in `replace()` raising an `AssertionError` when replacing values in an extension dtype with values of a different dtype (GH34530)

Bug fixes

- Fixed building from source with Python 3.8 fetching the wrong version of NumPy (GH34666)

Contributors

A total of 8 people contributed patches to this release. People with a "+" by their names contributed a patch for the first time.

- Joris Van den Bossche
- MeeseeksMachine
- Natalie Jann +
- Pandas Development Team
- Simon Hawkins
- Tom Augspurger

- William Ayd
- alimcmaster1

5.1.2 What's new in 1.0.4 (May 28, 2020)

These are the changes in pandas 1.0.4. See *Release Notes* for a full changelog including other versions of pandas.

Fixed regressions

- Fix regression where *Series.isna()* and *DataFrame.isna()* would raise for categorical dtype when `pandas.options.mode.use_inf_as_na` was set to `True` (GH33594)
- Fix regression in `GroupBy.first()` and `GroupBy.last()` where None is not preserved in object dtype (GH32800)
- Fix regression in DataFrame reductions using `numeric_only=True` and ExtensionArrays (GH33256).
- Fix performance regression in `memory_usage(deep=True)` for object dtype (GH33012)
- Fix regression where `Categorical.replace()` would replace with `NaN` whenever the new value and replacement value were equal (GH33288)
- Fix regression where an ordered *Categorical* containing only `NaN` values would raise rather than returning `NaN` when taking the minimum or maximum (GH33450)
- Fix regression in `DataFrameGroupBy.agg()` with dictionary input losing `ExtensionArray` dtypes (GH32194)
- Fix to preserve the ability to index with the "nearest" method with xarray's CFTimeIndex, an *Index* subclass (pydata/xarray#3751, GH32905).
- Fix regression in *DataFrame.describe()* raising `TypeError: unhashable type: 'dict'` (GH32409)
- Fix regression in *DataFrame.replace()* casts columns to `object` dtype if items in `to_replace` not in values (GH32988)
- Fix regression in *Series.groupby()* would raise `ValueError` when grouping by *PeriodIndex* level (GH34010)
- Fix regression in `GroupBy.rolling.apply()` ignores args and kwargs parameters (GH33433)
- Fix regression in error message with `np.min` or `np.max` on unordered *Categorical* (GH33115)
- Fix regression in *DataFrame.loc()* and *Series.loc()* throwing an error when a `datetime64[ns, tz]` value is provided (GH32395)

Bug fixes

- Bug in `SeriesGroupBy.first()`, `SeriesGroupBy.last()`, `SeriesGroupBy.min()`, and `SeriesGroupBy.max()` returning floats when applied to nullable Booleans (GH33071)
- Bug in `Rolling.min()` and `Rolling.max()`: Growing memory usage after multiple calls when using a fixed window (GH30726)
- Bug in *to_parquet()* was not raising `PermissionError` when writing to a private s3 bucket with invalid creds. (GH27679)
- Bug in *to_csv()* was silently failing when writing to an invalid s3 bucket. (GH32486)

- Bug in *read_parquet()* was raising a FileNotFoundError when passed an s3 directory path. (GH26388)
- Bug in *to_parquet()* was throwing an AttributeError when writing a partitioned parquet file to s3 (GH27596)
- Bug in GroupBy.quantile() causes the quantiles to be shifted when the by axis contains NaN (GH33200, GH33569)

Contributors

A total of 18 people contributed patches to this release. People with a "+" by their names contributed a patch for the first time.

- Daniel Saxton
- JDkuba +
- Joris Van den Bossche
- Kaiqi Dong
- Mabel Villalba
- MeeseeksMachine
- MomIsBestFriend
- Pandas Development Team
- Simon Hawkins
- Spencer Clark +
- Tom Augspurger
- Vikas Pandey +
- alimcmaster1
- h-vishal +
- jbrockmendel
- mproszewska +
- neilkg +
- rebecca-palmer +

5.1.3 What's new in 1.0.3 (March 17, 2020)

These are the changes in pandas 1.0.3. See *Release Notes* for a full changelog including other versions of pandas.

Fixed regressions

- Fixed regression in `resample.agg` when the underlying data is non-writeable (GH31710)
- Fixed regression in *`DataFrame`* exponentiation with reindexing (GH32685)

Bug fixes

Contributors

A total of 5 people contributed patches to this release. People with a "+" by their names contributed a patch for the first time.

- MeeseeksMachine
- Pandas Development Team
- Tom Augspurger
- William Ayd
- jbrockmendel

5.1.4 What's new in 1.0.2 (March 12, 2020)

These are the changes in pandas 1.0.2. See *Release Notes* for a full changelog including other versions of pandas.

Fixed regressions

Groupby

- Fixed regression in *`groupby(..).agg()`* which was failing on frames with MultiIndex columns and a custom function (GH31777)
- Fixed regression in `groupby(..).rolling(..).apply()` (`RollingGroupby`) where the `raw` parameter was ignored (GH31754)
- Fixed regression in *`rolling(..).corr()`* when using a time offset (GH31789)
- Fixed regression in *`groupby(..).nunique()`* which was modifying the original values if `NaN` values were present (GH31950)
- Fixed regression in `DataFrame.groupby` raising a `ValueError` from an internal operation (GH31802)
- Fixed regression in *`groupby(..).agg()`* calling a user-provided function an extra time on an empty input (GH31760)

I/O

- Fixed regression in *`read_csv()`* in which the `encoding` option was not recognized with certain file-like objects (GH31819)
- Fixed regression in *`DataFrame.to_excel()`* when the `columns` keyword argument is passed (GH31677)
- Fixed regression in `ExcelFile` where the stream passed into the function was closed by the destructor. (GH31467)
- Fixed regression where *`read_pickle()`* raised a `UnicodeDecodeError` when reading a py27 pickle with *`MultiIndex`* column (GH31988).

Reindexing/alignment

- Fixed regression in *Series.align()* when `other` is a DataFrame and `method` is not None (GH31785)
- Fixed regression in *DataFrame.reindex()* and *Series.reindex()* when reindexing with (tz-aware) index and `method=nearest` (GH26683)
- Fixed regression in *DataFrame.reindex_like()* on a *DataFrame* subclass raised an `AssertionError` (GH31925)
- Fixed regression in *DataFrame* arithmetic operations with mis-matched columns (GH31623)

Other

- Fixed regression in joining on *DatetimeIndex* or *TimedeltaIndex* to preserve `freq` in simple cases (GH32166)
- Fixed regression in *Series.shift()* with `datetime64` dtype when passing an integer `fill_value` (GH32591)
- Fixed regression in the repr of an object-dtype *Index* with bools and missing values (GH32146)

Indexing with Nullable Boolean Arrays

Previously indexing with a nullable Boolean array containing `NA` would raise a `ValueError`, however this is now permitted with `NA` being treated as `False`. (GH31503)

```
In [1]: s = pd.Series([1, 2, 3, 4])

In [2]: mask = pd.array([True, True, False, None], dtype="boolean")

In [3]: s
Out[3]:
0    1
1    2
2    3
3    4
Length: 4, dtype: int64

In [4]: mask
Out[4]:
<BooleanArray>
[True, True, False, <NA>]
Length: 4, dtype: boolean
```

pandas 1.0.0-1.0.1

```
>>> s[mask]
Traceback (most recent call last):
...
ValueError: cannot mask with array containing NA / NaN values
```

pandas 1.0.2

```
In [5]: s[mask]
Out[5]:
0    1
1    2
Length: 2, dtype: int64
```

Bug fixes

Datetimelike

- Bug in *Series.astype()* not copying for tz-naive and tz-aware datetime64 dtype (GH32490)
- Bug where *to_datetime()* would raise when passed `pd.NA` (GH32213)
- Improved error message when subtracting two *Timestamp* that result in an out-of-bounds *Timedelta* (GH31774)

Categorical

- Fixed bug where *Categorical.from_codes()* improperly raised a `ValueError` when passed nullable integer codes. (GH31779)
- Fixed bug where *Categorical()* constructor would raise a `TypeError` when given a numpy array containing `pd.NA`. (GH31927)
- Bug in *Categorical* that would ignore or crash when calling *Series.replace()* with a list-like `to_replace` (GH31720)

I/O

- Using `pd.NA` with *DataFrame.to_json()* now correctly outputs a null value instead of an empty object (GH31615)
- Bug in *pandas.json_normalize()* when value in meta path is not iterable (GH31507)
- Fixed pickling of `pandas.NA`. Previously a new object was returned, which broke computations relying on `NA` being a singleton (GH31847)
- Fixed bug in parquet roundtrip with nullable unsigned integer dtypes (GH31896).

Experimental dtypes

- Fixed bug in *DataFrame.convert_dtypes()* for columns that were already using the `"string"` dtype (GH31731).
- Fixed bug in *DataFrame.convert_dtypes()* for series with mix of integers and strings (GH32117)
- Fixed bug in *DataFrame.convert_dtypes()* where `BooleanDtype` columns were converted to `Int64` (GH32287)
- Fixed bug in setting values using a slice indexer with string dtype (GH31772)
- Fixed bug where *pandas.core.groupby.GroupBy.first()* and *pandas.core.groupby.GroupBy.last()* would raise a `TypeError` when groups contained `pd.NA` in a column of object dtype (GH32123)
- Fix bug in *Series.convert_dtypes()* for series with mix of integers and strings (GH32117)
- Fixed bug where `DataFrameGroupBy.mean()`, `DataFrameGroupBy.median()`, `DataFrameGroupBy.var()`, and `DataFrameGroupBy.std()` would raise a `TypeError` on `Int64` dtype columns (GH32219)

Strings

- Using `pd.NA` with *Series.str.repeat()* now correctly outputs a null value instead of raising error for vector inputs (GH31632)

Rolling

- Fixed rolling operations with variable window (defined by time duration) on decreasing time index (GH32385).

Contributors

A total of 25 people contributed patches to this release. People with a "+" by their names contributed a patch for the first time.

- Anna Daglis +
- Daniel Saxton
- Irv Lustig
- Jan Škoda
- Joris Van den Bossche
- Justin Zheng
- Kaiqi Dong
- Kendall Masse
- Marco Gorelli
- Matthew Roeschke
- MeeseeksMachine
- MomIsBestFriend
- Pandas Development Team
- Pedro Reys +
- Prakhar Pandey
- Robert de Vries +
- Rushabh Vasani
- Simon Hawkins
- Stijn Van Hoey
- Terji Petersen
- Tom Augspurger
- William Ayd
- alimcmaster1
- gfyoung
- jbrockmendel

5.1.5 What's new in 1.0.1 (February 5, 2020)

These are the changes in pandas 1.0.1. See *Release Notes* for a full changelog including other versions of pandas.

Fixed regressions

- Fixed regression in *DataFrame* setting values with a slice (e.g. `df[-4:] = 1`) indexing by label instead of position (GH31469)
- Fixed regression when indexing a `Series` or `DataFrame` indexed by `DatetimeIndex` with a slice containg a `datetime.date` (GH31501)
- Fixed regression in `DataFrame.__setitem__` raising an `AttributeError` with a *MultiIndex* and a non-monotonic indexer (GH31449)
- Fixed regression in *Series* multiplication when multiplying a numeric *Series* with >10000 elements with a timedelta-like scalar (GH31457)
- Fixed regression in `.groupby().agg()` raising an `AssertionError` for some reductions like `min` on object-dtype columns (GH31522)
- Fixed regression in `.groupby()` aggregations with categorical dtype using Cythonized reduction functions (e.g. `first`) (GH31450)
- Fixed regression in `GroupBy.apply()` if called with a function which returned a non-pandas non-scalar object (e.g. a list or numpy array) (GH31441)
- Fixed regression in *DataFrame.groupby()* whereby taking the minimum or maximum of a column with period dtype would raise a `TypeError`. (GH31471)
- Fixed regression in *DataFrame.groupby()* with an empty DataFrame grouping by a level of a MultiIndex (GH31670).
- Fixed regression in *DataFrame.apply()* with object dtype and non-reducing function (GH31505)
- Fixed regression in *to_datetime()* when parsing non-nanosecond resolution datetimes (GH31491)
- Fixed regression in *to_csv()* where specifying an `na_rep` might truncate the values written (GH31447)
- Fixed regression in *Categorical* construction with `numpy.str_` categories (GH31499)
- Fixed regression in *DataFrame.loc()* and *DataFrame.iloc()* when selecting a row containing a single `datetime64` or `timedelta64` column (GH31649)
- Fixed regression where setting `pd.options.display.max_colwidth` was not accepting negative integer. In addition, this behavior has been deprecated in favor of using `None` (GH31532)
- Fixed regression in objTOJSON.c fix return-type warning (GH31463)
- Fixed regression in *qcut()* when passed a nullable integer. (GH31389)
- Fixed regression in assigning to a *Series* using a nullable integer dtype (GH31446)
- Fixed performance regression when indexing a `DataFrame` or `Series` with a *MultiIndex* for the index using a list of labels (GH31648)
- Fixed regression in *read_csv()* used in file like object `RawIOBase` is not recognize `encoding` option (GH31575)

Deprecations

- Support for negative integer for `pd.options.display.max_colwidth` is deprecated in favor of using `None` (GH31532)

Bug fixes

Datetimelike

- Fixed bug in *`to_datetime()`* raising when `cache=True` and out-of-bound values are present (GH31491)

Numeric

- Bug in dtypes being lost in `DataFrame.__invert__` (~ operator) with mixed dtypes (GH31183) and for extension-array backed `Series` and `DataFrame` (GH23087)

Plotting

- Plotting tz-aware timeseries no longer gives UserWarning (GH31205)

Interval

- Bug in *`Series.shift()`* with `interval` dtype raising a `TypeError` when shifting an interval array of integers or datetimes (GH34195)

Contributors

A total of 15 people contributed patches to this release. People with a "+" by their names contributed a patch for the first time.

- Daniel Saxton
- Guillaume Lemaitre
- Jeff Reback
- Joris Van den Bossche
- Kaiqi Dong
- Marco Gorelli
- MeeseeksMachine
- Pandas Development Team
- Sebastián Vanrell +
- Tom Augspurger
- William Ayd
- alimcmaster1
- jbrockmendel
- paihu +
- proost

5.1.6 What's new in 1.0.0 (January 29, 2020)

These are the changes in pandas 1.0.0. See *Release Notes* for a full changelog including other versions of pandas.

Note: The pandas 1.0 release removed a lot of functionality that was deprecated in previous releases (see *below* for an overview). It is recommended to first upgrade to pandas 0.25 and to ensure your code is working without warnings, before upgrading to pandas 1.0.

New Deprecation Policy

Starting with Pandas 1.0.0, pandas will adopt a variant of SemVer to version releases. Briefly,

- Deprecations will be introduced in minor releases (e.g. 1.1.0, 1.2.0, 2.1.0, ...)
- Deprecations will be enforced in major releases (e.g. 1.0.0, 2.0.0, 3.0.0, ...)
- API-breaking changes will be made only in major releases (except for experimental features)

See *Version Policy* for more.

Enhancements

Using Numba in `rolling.apply` and `expanding.apply`

We've added an `engine` keyword to *`apply()`* and *`apply()`* that allows the user to execute the routine using Numba instead of Cython. Using the Numba engine can yield significant performance gains if the apply function can operate on numpy arrays and the data set is larger (1 million rows or greater). For more details, see *rolling apply documentation* (GH28987, GH30936)

Defining custom windows for rolling operations

We've added a *`pandas.api.indexers.BaseIndexer()`* class that allows users to define how window bounds are created during `rolling` operations. Users can define their own `get_window_bounds` method on a *`pandas.api.indexers.BaseIndexer()`* subclass that will generate the start and end indices used for each window during the rolling aggregation. For more details and example usage, see the *custom window rolling documentation*

Converting to Markdown

We've added *`to_markdown()`* for creating a markdown table (GH11052)

```
In [1]: df = pd.DataFrame({"A": [1, 2, 3], "B": [1, 2, 3]}, index=['a', 'a', 'b'])

In [2]: print(df.to_markdown())
|    |   A |   B |
|:---|----:|----:|
| a  |   1 |   1 |
| a  |   2 |   2 |
| b  |   3 |   3 |
```

Experimental new features

Experimental `NA` scalar to denote missing values

A new `pd.NA` value (singleton) is introduced to represent scalar missing values. Up to now, pandas used several values to represent missing data: `np.nan` is used for this for float data, `np.nan` or `None` for object-dtype data and `pd.NaT` for datetime-like data. The goal of `pd.NA` is to provide a "missing" indicator that can be used consistently across data types. `pd.NA` is currently used by the nullable integer and boolean data types and the new string data type (GH28095).

> **Warning:** Experimental: the behaviour of `pd.NA` can still change without warning.

For example, creating a Series using the nullable integer dtype:

```
In [3]: s = pd.Series([1, 2, None], dtype="Int64")

In [4]: s
Out[4]:
0       1
1       2
2    <NA>
Length: 3, dtype: Int64

In [5]: s[2]
Out[5]: <NA>
```

Compared to `np.nan`, `pd.NA` behaves differently in certain operations. In addition to arithmetic operations, `pd.NA` also propagates as "missing" or "unknown" in comparison operations:

```
In [6]: np.nan > 1
Out[6]: False

In [7]: pd.NA > 1
Out[7]: <NA>
```

For logical operations, `pd.NA` follows the rules of the three-valued logic (or *Kleene logic*). For example:

```
In [8]: pd.NA | True
Out[8]: True
```

For more, see *NA section* in the user guide on missing data.

Dedicated string data type

We've added *`StringDtype`*, an extension type dedicated to string data. Previously, strings were typically stored in object-dtype NumPy arrays. (GH29975)

> **Warning:** `StringDtype` is currently considered experimental. The implementation and parts of the API may change without warning.

The `'string'` extension type solves several issues with object-dtype NumPy arrays:

1. You can accidentally store a *mixture* of strings and non-strings in an `object` dtype array. A `StringArray` can only store strings.
2. `object` dtype breaks dtype-specific operations like *`DataFrame.select_dtypes()`*. There isn't a clear way to select *just* text while excluding non-text, but still object-dtype columns.
3. When reading code, the contents of an `object` dtype array is less clear than `string`.

```
In [9]: pd.Series(['abc', None, 'def'], dtype=pd.StringDtype())
Out[9]:
0     abc
1    <NA>
2     def
Length: 3, dtype: string
```

You can use the alias `"string"` as well.

```
In [10]: s = pd.Series(['abc', None, 'def'], dtype="string")

In [11]: s
Out[11]:
0     abc
1    <NA>
2     def
Length: 3, dtype: string
```

The usual string accessor methods work. Where appropriate, the return type of the Series or columns of a DataFrame will also have string dtype.

```
In [12]: s.str.upper()
Out[12]:
0     ABC
1    <NA>
2     DEF
Length: 3, dtype: string

In [13]: s.str.split('b', expand=True).dtypes
Out[13]:
0    string
1    string
Length: 2, dtype: object
```

String accessor methods returning integers will return a value with *`Int64Dtype`*

```
In [14]: s.str.count("a")
Out[14]:
0       1
1    <NA>
2       0
Length: 3, dtype: Int64
```

We recommend explicitly using the `string` data type when working with strings. See *Text Data Types* for more.

Boolean data type with missing values support

We've added *BooleanDtype* / *BooleanArray*, an extension type dedicated to boolean data that can hold missing values. The default `bool` data type based on a bool-dtype NumPy array, the column can only hold `True` or `False`, and not missing values. This new *BooleanArray* can store missing values as well by keeping track of this in a separate mask. (GH29555, GH30095, GH31131)

```
In [15]: pd.Series([True, False, None], dtype=pd.BooleanDtype())
Out[15]:
0     True
1    False
2     <NA>
Length: 3, dtype: boolean
```

You can use the alias `"boolean"` as well.

```
In [16]: s = pd.Series([True, False, None], dtype="boolean")

In [17]: s
Out[17]:
0     True
1    False
2     <NA>
Length: 3, dtype: boolean
```

`convert_dtypes` method to ease use of supported extension dtypes

In order to encourage use of the extension dtypes `StringDtype`, `BooleanDtype`, `Int64Dtype`, `Int32Dtype`, etc., that support `pd.NA`, the methods *DataFrame.convert_dtypes()* and *Series.convert_dtypes()* have been introduced. (GH29752) (GH30929)

Example:

```
In [18]: df = pd.DataFrame({'x': ['abc', None, 'def'],
   ....:                    'y': [1, 2, np.nan],
   ....:                    'z': [True, False, True]})
   ....:

In [19]: df
Out[19]:
      x    y      z
0   abc  1.0   True
1  None  2.0  False
2   def  NaN   True

[3 rows x 3 columns]

In [20]: df.dtypes
Out[20]:
x     object
y    float64
z       bool
Length: 3, dtype: object
```

```
In [21]: converted = df.convert_dtypes()

In [22]: converted
Out[22]:
      x     y      z
0   abc     1   True
1  <NA>     2  False
2   def  <NA>   True

[3 rows x 3 columns]

In [23]: converted.dtypes
Out[23]:
x     string
y      Int64
z    boolean
Length: 3, dtype: object
```

This is especially useful after reading in data using readers such as *read_csv()* and *read_excel()*. See *here* for a description.

Other enhancements

- *DataFrame.to_string()* added the `max_colwidth` parameter to control when wide columns are truncated (GH9784)
- Added the `na_value` argument to *Series.to_numpy()*, *Index.to_numpy()* and *DataFrame.to_numpy()* to control the value used for missing data (GH30322)
- *MultiIndex.from_product()* infers level names from inputs if not explicitly provided (GH27292)
- *DataFrame.to_latex()* now accepts `caption` and `label` arguments (GH25436)
- DataFrames with *nullable integer*, the *new string dtype* and period data type can now be converted to `pyarrow` (>=0.15.0), which means that it is supported in writing to the Parquet file format when using the `pyarrow` engine (GH28368). Full roundtrip to parquet (writing and reading back in with *to_parquet()* / *read_parquet()*) is supported starting with pyarrow >= 0.16 (GH20612).
- `to_parquet()` now appropriately handles the `schema` argument for user defined schemas in the pyarrow engine. (GH30270)
- *DataFrame.to_json()* now accepts an `indent` integer argument to enable pretty printing of JSON output (GH12004)
- *read_stata()* can read Stata 119 dta files. (GH28250)
- Implemented `pandas.core.window.Window.var()` and `pandas.core.window.Window.std()` functions (GH26597)
- Added `encoding` argument to *DataFrame.to_string()* for non-ascii text (GH28766)
- Added `encoding` argument to *DataFrame.to_html()* for non-ascii text (GH28663)
- `Styler.background_gradient()` now accepts `vmin` and `vmax` arguments (GH12145)
- `Styler.format()` added the `na_rep` parameter to help format the missing values (GH21527, GH28358)
- *read_excel()* now can read binary Excel (`.xlsb`) files by passing `engine='pyxlsb'`. For more details and example usage, see the *Binary Excel files documentation*. Closes GH8540.
- The `partition_cols` argument in *DataFrame.to_parquet()* now accepts a string (GH27117)

- *pandas.read_json()* now parses `NaN`, `Infinity` and `-Infinity` (GH12213)
- DataFrame constructor preserve *ExtensionArray* dtype with *ExtensionArray* (GH11363)
- *DataFrame.sort_values()* and *Series.sort_values()* have gained `ignore_index` keyword to be able to reset index after sorting (GH30114)
- *DataFrame.sort_index()* and *Series.sort_index()* have gained `ignore_index` keyword to reset index (GH30114)
- *DataFrame.drop_duplicates()* has gained `ignore_index` keyword to reset index (GH30114)
- Added new writer for exporting Stata dta files in versions 118 and 119, `StataWriterUTF8`. These files formats support exporting strings containing Unicode characters. Format 119 supports data sets with more than 32,767 variables (GH23573, GH30959)
- *Series.map()* now accepts `collections.abc.Mapping` subclasses as a mapper (GH29733)
- Added an experimental *attrs* for storing global metadata about a dataset (GH29062)
- `Timestamp.fromisocalendar()` is now compatible with python 3.8 and above (GH28115)
- *DataFrame.to_pickle()* and *read_pickle()* now accept URL (GH30163)

Backwards incompatible API changes

Avoid using names from `MultiIndex.levels`

As part of a larger refactor to *MultiIndex* the level names are now stored separately from the levels (GH27242). We recommend using *MultiIndex.names* to access the names, and *Index.set_names()* to update the names.

For backwards compatibility, you can still *access* the names via the levels.

```
In [24]: mi = pd.MultiIndex.from_product([[1, 2], ['a', 'b']], names=['x', 'y'])

In [25]: mi.levels[0].name
Out[25]: 'x'
```

However, it is no longer possible to *update* the names of the `MultiIndex` via the level.

```
In [26]: mi.levels[0].name = "new name"
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
<ipython-input-26-65f4400a0c97> in <module>
----> 1 mi.levels[0].name = "new name"

/pandas-release/pandas/pandas/core/indexes/base.py in name(self, value)
   1189             # Used in MultiIndex.levels to avoid silently ignoring name␣
↪updates.
   1190             raise RuntimeError(
-> 1191                 "Cannot set name on a level of a MultiIndex. Use "
   1192                 "'MultiIndex.set_names' instead."
   1193             )

RuntimeError: Cannot set name on a level of a MultiIndex. Use 'MultiIndex.set_names'␣
↪instead.

In [27]: mi.names
Out[27]: FrozenList(['x', 'y'])
```

To update, use `MultiIndex.set_names`, which returns a new `MultiIndex`.

```
In [28]: mi2 = mi.set_names("new name", level=0)

In [29]: mi2.names
Out[29]: FrozenList(['new name', 'y'])
```

New repr for `IntervalArray`

pandas.arrays.IntervalArray adopts a new `__repr__` in accordance with other array classes (GH25022)

pandas 0.25.x

```
In [1]: pd.arrays.IntervalArray.from_tuples([(0, 1), (2, 3)])
Out[2]:
IntervalArray([(0, 1], (2, 3]],
              closed='right',
              dtype='interval[int64]')
```

pandas 1.0.0

```
In [30]: pd.arrays.IntervalArray.from_tuples([(0, 1), (2, 3)])
Out[30]:
<IntervalArray>
[(0, 1], (2, 3]]
Length: 2, closed: right, dtype: interval[int64]
```

`DataFrame.rename` now only accepts one positional argument

DataFrame.rename() would previously accept positional arguments that would lead to ambiguous or undefined behavior. From pandas 1.0, only the very first argument, which maps labels to their new names along the default axis, is allowed to be passed by position (GH29136).

pandas 0.25.x

```
>>> df = pd.DataFrame([[1]])
>>> df.rename({0: 1}, {0: 2})
FutureWarning: ...Use named arguments to resolve ambiguity...
   2
1  1
```

pandas 1.0.0

```
>>> df.rename({0: 1}, {0: 2})
Traceback (most recent call last):
...
TypeError: rename() takes from 1 to 2 positional arguments but 3 were given
```

Note that errors will now be raised when conflicting or potentially ambiguous arguments are provided.

pandas 0.25.x

```
>>> df.rename({0: 1}, index={0: 2})
   0
1  1
```

(continues on next page)

(continued from previous page)

```
>>> df.rename(mapper={0: 1}, index={0: 2})
   0
2  1
```

pandas 1.0.0

```
>>> df.rename({0: 1}, index={0: 2})
Traceback (most recent call last):
...
TypeError: Cannot specify both 'mapper' and any of 'index' or 'columns'

>>> df.rename(mapper={0: 1}, index={0: 2})
Traceback (most recent call last):
...
TypeError: Cannot specify both 'mapper' and any of 'index' or 'columns'
```

You can still change the axis along which the first positional argument is applied by supplying the `axis` keyword argument.

```
In [31]: df.rename({0: 1})
Out[31]:
   0
1  1

[1 rows x 1 columns]

In [32]: df.rename({0: 1}, axis=1)
Out[32]:
   1
0  1

[1 rows x 1 columns]
```

If you would like to update both the index and column labels, be sure to use the respective keywords.

```
In [33]: df.rename(index={0: 1}, columns={0: 2})
Out[33]:
   2
1  1

[1 rows x 1 columns]
```

Extended verbose info output for `DataFrame`

`DataFrame.info()` now shows line numbers for the columns summary (GH17304)

pandas 0.25.x

```
>>> df = pd.DataFrame({"int_col": [1, 2, 3],
...                    "text_col": ["a", "b", "c"],
...                    "float_col": [0.0, 0.1, 0.2]})
>>> df.info(verbose=True)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3 entries, 0 to 2
```

(continues on next page)

(continued from previous page)

```
Data columns (total 3 columns):
int_col      3 non-null int64
text_col     3 non-null object
float_col    3 non-null float64
dtypes: float64(1), int64(1), object(1)
memory usage: 152.0+ bytes
```

pandas 1.0.0

```
In [34]: df = pd.DataFrame({"int_col": [1, 2, 3],
   ....:                    "text_col": ["a", "b", "c"],
   ....:                    "float_col": [0.0, 0.1, 0.2]})
   ....:

In [35]: df.info(verbose=True)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3 entries, 0 to 2
Data columns (total 3 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   int_col    3 non-null      int64
 1   text_col   3 non-null      object
 2   float_col  3 non-null      float64
dtypes: float64(1), int64(1), object(1)
memory usage: 200.0+ bytes
```

`pandas.array()` inference changes

pandas.array() now infers pandas' new extension types in several cases (GH29791):

1. String data (including missing values) now returns a *arrays.StringArray*.
2. Integer data (including missing values) now returns a *arrays.IntegerArray*.
3. Boolean data (including missing values) now returns the new *arrays.BooleanArray*

pandas 0.25.x

```
>>> pd.array(["a", None])
<PandasArray>
['a', None]
Length: 2, dtype: object

>>> pd.array([1, None])
<PandasArray>
[1, None]
Length: 2, dtype: object
```

pandas 1.0.0

```
In [36]: pd.array(["a", None])
Out[36]:
<StringArray>
['a', <NA>]
Length: 2, dtype: string
```

(continues on next page)

(continued from previous page)

```
In [37]: pd.array([1, None])
Out[37]:
<IntegerArray>
[1, <NA>]
Length: 2, dtype: Int64
```

As a reminder, you can specify the `dtype` to disable all inference.

`arrays.IntegerArray` now uses `pandas.NA`

arrays.IntegerArray now uses `pandas.NA` rather than `numpy.nan` as its missing value marker (GH29964).

pandas 0.25.x

```
>>> a = pd.array([1, 2, None], dtype="Int64")
>>> a
<IntegerArray>
[1, 2, NaN]
Length: 3, dtype: Int64

>>> a[2]
nan
```

pandas 1.0.0

```
In [38]: a = pd.array([1, 2, None], dtype="Int64")

In [39]: a
Out[39]:
<IntegerArray>
[1, 2, <NA>]
Length: 3, dtype: Int64

In [40]: a[2]
Out[40]: <NA>
```

This has a few API-breaking consequences.

Converting to a NumPy ndarray

When converting to a NumPy array missing values will be `pd.NA`, which cannot be converted to a float. So calling `np.asarray(integer_array, dtype="float")` will now raise.

pandas 0.25.x

```
>>> np.asarray(a, dtype="float")
array([ 1.,  2., nan])
```

pandas 1.0.0

```
In [41]: np.asarray(a, dtype="float")
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-41-4578f04da2b3> in <module>
----> 1 np.asarray(a, dtype="float")
```

(continues on next page)