- Nigel Markey +
- Noritada Kobayashi +
- Oktay Sabak +
- Oliver Hofkens +
- Oluokun Adedayo +
- Osman +
- Oğuzhan Öğreden +
- Pandas Development Team +
- Patrik Hlobil +
- Paul Lee +
- Paul Siegel +
- Petr Baev +
- Pietro Battiston
- Prakhar Pandey +
- Puneeth K +
- Raghav +
- Rajat +
- Rajhans Jadhao +
- Rajiv Bharadwaj +
- Rik-de-Kort +
- Roei.r
- Rohit Sanjay +
- Ronan Lamy +
- Roshni +
- Roymprog +
- Rushabh Vasani +
- Ryan Grout +
- Ryan Nazareth
- Samesh Lakhotia +
- Samuel Sinayoko
- Samyak Jain +
- Sarah Donehower +
- Sarah Masud +
- Saul Shanabrook +
- Scott Cole +
- SdgJlbl +

- Seb +
- Sergei Ivko +
- Shadi Akiki
- Shorokhov Sergey
- Siddhesh Poyarekar +
- Sidharthan Nair +
- Simon Gibbons
- Simon Hawkins
- Simon-Martin Schröder +
- Sofiane Mahiou +
- Sourav kumar +
- Souvik Mandal +
- Soyoun Kim +
- Sparkle Russell-Puleri +
- Srinivas Reddy Thatiparthy ( )
- Stuart Berg +
- Sumanau Sareen
- Szymon Bednarek +
- Tambe Tabitha Achere +
- Tan Tran
- Tang Heyi +
- Tanmay Daripa +
- Tanya Jain
- Terji Petersen
- Thomas Li +
- Tirth Jain +
- Tola A +
- Tom Augspurger
- Tommy Lynch +
- Tomoyuki Suzuki +
- Tony Lorenzo
- Unprocessable +
- Uwe L. Korn
- Vaibhav Vishal
- Victoria Zdanovskaya +
- Vijayant +

- Vishwak Srinivasan +
- WANG Aiyong
- Wenhuan
- Wes McKinney
- Will Ayd
- Will Holmgren
- William Ayd
- William Blan +
- Wouter Overmeire
- Wuraola Oyewusi +
- YaOzI +
- Yash Shukla +
- Yu Wang +
- Yusei Tahara +
- alexander135 +
- alimcmaster1
- avelineg +
- bganglia +
- bolkedebruin
- bravech +
- chinhwee +
- cruzzoe +
- dalgarno +
- daniellebrown +
- danielplawrence
- est271 +
- francisco souza +
- ganevgv +
- garanews +
- gfyoung
- h-vetinari
- hasnain2808 +
- ianzur +
- jalbritt +
- jbrockmendel
- jeschwar +

- jlamborn324 +
- joy-rosie +
- kernc
- killerontherun1
- krey +
- lexy-lixinyu +
- lucyleeow +
- lukasbk +
- maheshbapatu +
- mck619 +
- nathalier
- naveenkaushik2504 +
- nlepleux +
- nrebena
- ohad83 +
- pilkibun
- pqzx +
- proost +
- pv8493013j +
- qudade +
- rhstanton +
- rmunjal29 +
- sangarshanan +
- sardonick +
- saskakarsi +
- shaido987 +
- ssikdar1
- steveayers124 +
- tadashigaki +
- timcera +
- tlaytongoogle +
- tobycheese
- tonywu1999 +
- tsvikas +
- yogendrasoni +
- zys5945 +

5.2 Version 0.25

5.2.1 What's new in 0.25.3 (October 31, 2019)

These are the changes in pandas 0.25.3. See *Release Notes* for a full changelog including other versions of pandas.

Bug fixes

Groupby/resample/rolling

- Bug in `DataFrameGroupBy.quantile()` where NA values in the grouping could cause segfaults or incorrect results (GH28882)

Contributors

A total of 2 people contributed patches to this release. People with a "+" by their names contributed a patch for the first time.

- Will Ayd
- William Ayd

5.2.2 What's new in 0.25.2 (October 15, 2019)

These are the changes in pandas 0.25.2. See *Release Notes* for a full changelog including other versions of pandas.

Note: Pandas 0.25.2 adds compatibility for Python 3.8 (GH28147).

Bug fixes

Indexing

- Fix regression in `DataFrame.reindex()` not following the `limit` argument (GH28631).
- Fix regression in `RangeIndex.get_indexer()` for decreasing `RangeIndex` where target values may be improperly identified as missing/present (GH28678)

I/O

- Fix regression in notebook display where `<th>` tags were missing for `DataFrame.index` values (GH28204).
- Regression in `to_csv()` where writing a `Series` or `DataFrame` indexed by an `IntervalIndex` would incorrectly raise a `TypeError` (GH28210)
- Fix `to_csv()` with `ExtensionArray` with list-like values (GH28840).

Groupby/resample/rolling

- Bug incorrectly raising an `IndexError` when passing a list of quantiles to *pandas.core.groupby.DataFrameGroupBy.quantile()* (GH28113).
- Bug in `pandas.core.groupby.GroupBy.shift()`, *pandas.core.groupby.GroupBy.bfill()* and *pandas.core.groupby.GroupBy.ffill()* where timezone information would be dropped (GH19995, GH27992)

Other

- Compatibility with Python 3.8 in `DataFrame.query()` (GH27261)
- Fix to ensure that tab-completion in an IPython console does not raise warnings for deprecated attributes (GH27900).

Contributors

A total of 6 people contributed patches to this release. People with a "+" by their names contributed a patch for the first time.

- Felix Divo +
- Jeremy Schendel
- Joris Van den Bossche
- MeeseeksMachine
- Tom Augspurger
- jbrockmendel

5.2.3 What's new in 0.25.1 (August 21, 2019)

These are the changes in pandas 0.25.1. See *Release Notes* for a full changelog including other versions of pandas.

I/O and LZMA

Some users may unknowingly have an incomplete Python installation lacking the *lzma* module from the standard library. In this case, *import pandas* failed due to an *ImportError* (GH27575). Pandas will now warn, rather than raising an *ImportError* if the *lzma* module is not present. Any subsequent attempt to use *lzma* methods will raise a *RuntimeError*. A possible fix for the lack of the *lzma* module is to ensure you have the necessary libraries and then re-install Python. For example, on MacOS installing Python with *pyenv* may lead to an incomplete Python installation due to unmet system dependencies at compilation time (like *xz*). Compilation will succeed, but Python might fail at run time. The issue can be solved by installing the necessary dependencies and then re-installing Python.

Bug fixes

Categorical

- Bug in `Categorical.fillna()` that would replace all values, not just those that are `NaN` (GH26215)

Datetimelike

- Bug in `to_datetime()` where passing a timezone-naive `DatetimeArray` or `DatetimeIndex` and `utc=True` would incorrectly return a timezone-naive result (GH27733)
- Bug in `Period.to_timestamp()` where a `Period` outside the `Timestamp` implementation bounds (roughly 1677-09-21 to 2262-04-11) would return an incorrect `Timestamp` instead of raising `OutOfBoundsDatetime` (GH19643)
- Bug in iterating over `DatetimeIndex` when the underlying data is read-only (GH28055)

Timezones

- Bug in `Index` where a numpy object array with a timezone aware `Timestamp` and `np.nan` would not return a `DatetimeIndex` (GH27011)

Numeric

- Bug in `Series.interpolate()` when using a timezone aware `DatetimeIndex` (GH27548)
- Bug when printing negative floating point complex numbers would raise an `IndexError` (GH27484)
- Bug where `DataFrame` arithmetic operators such as `DataFrame.mul()` with a `Series` with axis=1 would raise an `AttributeError` on `DataFrame` larger than the minimum threshold to invoke numexpr (GH27636)
- Bug in `DataFrame` arithmetic where missing values in results were incorrectly masked with `NaN` instead of `Inf` (GH27464)

Conversion

- Improved the warnings for the deprecated methods `Series.real()` and `Series.imag()` (GH27610)

Interval

- Bug in `IntervalIndex` where *dir(obj)* would raise `ValueError` (GH27571)

Indexing

- Bug in partial-string indexing returning a NumPy array rather than a `Series` when indexing with a scalar like `.loc['2015']` (GH27516)
- Break reference cycle involving `Index` and other index classes to allow garbage collection of index objects without running the GC. (GH27585, GH27840)
- Fix regression in assigning values to a single column of a DataFrame with a `MultiIndex` columns (GH27841).
- Fix regression in `.ix` fallback with an `IntervalIndex` (GH27865).

Missing

- Bug in *pandas.isnull()* or *pandas.isna()* when the input is a type e.g. `type(pandas.Series())` (GH27482)

I/O

- Avoid calling `S3File.s3` when reading parquet, as this was removed in s3fs version 0.3.0 (GH27756)
- Better error message when a negative header is passed in *pandas.read_csv()* (GH27779)
- Follow the `min_rows` display option (introduced in v0.25.0) correctly in the HTML repr in the notebook (GH27991).

Plotting

- Added a `pandas_plotting_backends` entrypoint group for registering plot backends. See *Plotting backends* for more (GH26747).
- Fixed the re-instatement of Matplotlib datetime converters after calling *pandas.plotting.deregister_matplotlib_converters()* (GH27481).
- Fix compatibility issue with matplotlib when passing a pandas `Index` to a plot call (GH27775).

Groupby/resample/rolling

- Fixed regression in `pands.core.groupby.DataFrameGroupBy.quantile()` raising when multiple quantiles are given (GH27526)
- Bug in `pandas.core.groupby.DataFrameGroupBy.transform()` where applying a timezone conversion lambda function would drop timezone information (GH27496)
- Bug in *pandas.core.groupby.GroupBy.nth()* where `observed=False` was being ignored for Categorical groupers (GH26385)
- Bug in windowing over read-only arrays (GH27766)
- Fixed segfault in *pandas.core.groupby.DataFrameGroupBy.quantile* when an invalid quantile was passed (GH27470)

Reshaping

- A `KeyError` is now raised if `.unstack()` is called on a `Series` or `DataFrame` with a flat `Index` passing a name which is not the correct one (GH18303)
- Bug `merge_asof()` could not merge `Timedelta` objects when passing *tolerance* kwarg (GH27642)
- Bug in `DataFrame.crosstab()` when `margins` set to `True` and `normalize` is not `False`, an error is raised. (GH27500)
- `DataFrame.join()` now suppresses the `FutureWarning` when the sort parameter is specified (GH21952)
- Bug in `DataFrame.join()` raising with readonly arrays (GH27943)

Sparse

- Bug in reductions for `Series` with Sparse dtypes (GH27080)

Other

- Bug in `Series.replace()` and `DataFrame.replace()` when replacing timezone-aware timestamps using a dict-like replacer (GH27720)
- Bug in `Series.rename()` when using a custom type indexer. Now any value that isn't callable or dict-like is treated as a scalar. (GH27814)

Contributors

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

- Jeff Reback
- Joris Van den Bossche
- MeeseeksMachine +
- Tom Augspurger
- jbrockmendel

5.2.4 What's new in 0.25.0 (July 18, 2019)

Warning: Starting with the 0.25.x series of releases, pandas only supports Python 3.5.3 and higher. See Dropping Python 2.7 for more details.

Warning: The minimum supported Python version will be bumped to 3.6 in a future release.

Warning: *Panel* has been fully removed. For N-D labeled data structures, please use xarray

Warning: `read_pickle()` and `read_msgpack()` are only guaranteed backwards compatible back to pandas version 0.20.3 (GH27082)

These are the changes in pandas 0.25.0. See *Release Notes* for a full changelog including other versions of pandas.

Enhancements

Groupby aggregation with relabeling

Pandas has added special groupby behavior, known as "named aggregation", for naming the output columns when applying multiple aggregation functions to specific columns (GH18366, GH26512).

```
In [1]: animals = pd.DataFrame({'kind': ['cat', 'dog', 'cat', 'dog'],
   ...:                         'height': [9.1, 6.0, 9.5, 34.0],
   ...:                         'weight': [7.9, 7.5, 9.9, 198.0]})
   ...:

In [2]: animals
Out[2]:
  kind  height  weight
0  cat     9.1     7.9
1  dog     6.0     7.5
2  cat     9.5     9.9
3  dog    34.0   198.0

[4 rows x 3 columns]

In [3]: animals.groupby("kind").agg(
   ...:     min_height=pd.NamedAgg(column='height', aggfunc='min'),
   ...:     max_height=pd.NamedAgg(column='height', aggfunc='max'),
   ...:     average_weight=pd.NamedAgg(column='weight', aggfunc=np.mean),
   ...: )
   ...:
Out[3]:
      min_height  max_height  average_weight
kind
cat          9.1         9.5            8.90
dog          6.0        34.0          102.75

[2 rows x 3 columns]
```

Pass the desired columns names as the `**kwargs` to `.agg`. The values of `**kwargs` should be tuples where the first element is the column selection, and the second element is the aggregation function to apply. Pandas provides the `pandas.NamedAgg` namedtuple to make it clearer what the arguments to the function are, but plain tuples are accepted as well.

```
In [4]: animals.groupby("kind").agg(
   ...:     min_height=('height', 'min'),
   ...:     max_height=('height', 'max'),
   ...:     average_weight=('weight', np.mean),
   ...: )
   ...:
Out[4]:
      min_height  max_height  average_weight
```

(continues on next page)

(continued from previous page)

```
kind
cat          9.1         9.5            8.90
dog          6.0        34.0          102.75

[2 rows x 3 columns]
```

Named aggregation is the recommended replacement for the deprecated "dict-of-dicts" approach to naming the output of column-specific aggregations (*Deprecate groupby.agg() with a dictionary when renaming*).

A similar approach is now available for Series groupby objects as well. Because there's no need for column selection, the values can just be the functions to apply

```
In [5]: animals.groupby("kind").height.agg(
   ...:     min_height="min",
   ...:     max_height="max",
   ...: )
   ...:
Out[5]:
      min_height  max_height
kind
cat          9.1         9.5
dog          6.0        34.0

[2 rows x 2 columns]
```

This type of aggregation is the recommended alternative to the deprecated behavior when passing a dict to a Series groupby aggregation (*Deprecate groupby.agg() with a dictionary when renaming*).

See *Named aggregation* for more.

Groupby Aggregation with multiple lambdas

You can now provide multiple lambda functions to a list-like aggregation in *pandas.core.groupby.GroupBy.agg* (GH26430).

```
In [6]: animals.groupby('kind').height.agg([
   ...:     lambda x: x.iloc[0], lambda x: x.iloc[-1]
   ...: ])
   ...:
Out[6]:
      <lambda_0>  <lambda_1>
kind
cat          9.1         9.5
dog          6.0        34.0

[2 rows x 2 columns]

In [7]: animals.groupby('kind').agg([
   ...:     lambda x: x.iloc[0] - x.iloc[1],
   ...:     lambda x: x.iloc[0] + x.iloc[1]
   ...: ])
   ...:
Out[7]:
         height                  weight
     <lambda_0> <lambda_1> <lambda_0> <lambda_1>
```

(continues on next page)

(continued from previous page)

```
kind
cat        -0.4       18.6       -2.0       17.8
dog       -28.0       40.0     -190.5      205.5

[2 rows x 4 columns]
```

Previously, these raised a `SpecificationError`.

Better repr for MultiIndex

Printing of *MultiIndex* instances now shows tuples of each row and ensures that the tuple items are vertically aligned, so it's now easier to understand the structure of the `MultiIndex`. (GH13480):

The repr now looks like this:

```
In [8]: pd.MultiIndex.from_product([['a', 'abc'], range(500)])
Out[8]:
MultiIndex([(  'a',   0),
            (  'a',   1),
            (  'a',   2),
            (  'a',   3),
            (  'a',   4),
            (  'a',   5),
            (  'a',   6),
            (  'a',   7),
            (  'a',   8),
            (  'a',   9),
            ...
            ('abc', 490),
            ('abc', 491),
            ('abc', 492),
            ('abc', 493),
            ('abc', 494),
            ('abc', 495),
            ('abc', 496),
            ('abc', 497),
            ('abc', 498),
            ('abc', 499)],
           length=1000)
```

Previously, outputting a *MultiIndex* printed all the `levels` and `codes` of the `MultiIndex`, which was visually unappealing and made the output more difficult to navigate. For example (limiting the range to 5):

```
In [1]: pd.MultiIndex.from_product([['a', 'abc'], range(5)])
Out[1]: MultiIndex(levels=[['a', 'abc'], [0, 1, 2, 3]],
   ...:            codes=[[0, 0, 0, 0, 1, 1, 1, 1], [0, 1, 2, 3, 0, 1, 2, 3]])
```

In the new repr, all values will be shown, if the number of rows is smaller than `options.display.max_seq_items` (default: 100 items). Horizontally, the output will truncate, if it's wider than `options.display.width` (default: 80 characters).

Shorter truncated repr for Series and DataFrame

Currently, the default display options of pandas ensure that when a Series or DataFrame has more than 60 rows, its repr gets truncated to this maximum of 60 rows (the `display.max_rows` option). However, this still gives a repr that takes up a large part of the vertical screen estate. Therefore, a new option `display.min_rows` is introduced with a default of 10 which determines the number of rows showed in the truncated repr:

- For small Series or DataFrames, up to `max_rows` number of rows is shown (default: 60).
- For larger Series of DataFrame with a length above `max_rows`, only `min_rows` number of rows is shown (default: 10, i.e. the first and last 5 rows).

This dual option allows to still see the full content of relatively small objects (e.g. `df.head(20)` shows all 20 rows), while giving a brief repr for large objects.

To restore the previous behaviour of a single threshold, set `pd.options.display.min_rows = None`.

Json normalize with max_level param support

`json_normalize()` normalizes the provided input dict to all nested levels. The new max_level parameter provides more control over which level to end normalization (GH23843):

The repr now looks like this:

```
from pandas.io.json import json_normalize
data = [{
    'CreatedBy': {'Name': 'User001'},
    'Lookup': {'TextField': 'Some text',
               'UserField': {'Id': 'ID001', 'Name': 'Name001'}},
    'Image': {'a': 'b'}
}]
json_normalize(data, max_level=1)
```

Series.explode to split list-like values to rows

`Series` and *`DataFrame`* have gained the *`DataFrame.explode()`* methods to transform list-likes to individual rows. See *section on Exploding list-like column* in docs for more information (GH16538, GH10511)

Here is a typical usecase. You have comma separated string in a column.

```
In [9]: df = pd.DataFrame([{'var1': 'a,b,c', 'var2': 1},
   ...:                    {'var1': 'd,e,f', 'var2': 2}])
   ...:

In [10]: df
Out[10]:
    var1  var2
0  a,b,c     1
1  d,e,f     2

[2 rows x 2 columns]
```

Creating a long form `DataFrame` is now straightforward using chained operations

```
In [11]: df.assign(var1=df.var1.str.split(',')).explode('var1')
Out[11]:
  var1  var2
0    a     1
0    b     1
0    c     1
1    d     2
1    e     2
1    f     2

[6 rows x 2 columns]
```

Other enhancements

- *DataFrame.plot()* keywords `logy`, `logx` and `loglog` can now accept the value `'sym'` for symlog scaling. (GH24867)
- Added support for ISO week year format ('%G-%V-%u') when parsing datetimes using *to_datetime()* (GH16607)
- Indexing of `DataFrame` and `Series` now accepts zerodim `np.ndarray` (GH24919)
- *Timestamp.replace()* now supports the `fold` argument to disambiguate DST transition times (GH25017)
- *DataFrame.at_time()* and *Series.at_time()* now support `datetime.time` objects with timezones (GH24043)
- *DataFrame.pivot_table()* now accepts an `observed` parameter which is passed to underlying calls to *DataFrame.groupby()* to speed up grouping categorical data. (GH24923)
- `Series.str` has gained *Series.str.casefold()* method to removes all case distinctions present in a string (GH25405)
- *DataFrame.set_index()* now works for instances of `abc.Iterator`, provided their output is of the same length as the calling frame (GH22484, GH24984)
- `DatetimeIndex.union()` now supports the `sort` argument. The behavior of the sort parameter matches that of *Index.union()* (GH24994)
- `RangeIndex.union()` now supports the `sort` argument. If `sort=False` an unsorted `Int64Index` is always returned. `sort=None` is the default and returns a monotonically increasing `RangeIndex` if possible or a sorted `Int64Index` if not (GH24471)
- `TimedeltaIndex.intersection()` now also supports the `sort` keyword (GH24471)
- *DataFrame.rename()* now supports the `errors` argument to raise errors when attempting to rename nonexistent keys (GH13473)
- Added *Sparse accessor* for working with a `DataFrame` whose values are sparse (GH25681)
- *RangeIndex* has gained *start*, *stop*, and *step* attributes (GH25710)
- `datetime.timezone` objects are now supported as arguments to timezone methods and constructors (GH25065)
- *DataFrame.query()* and *DataFrame.eval()* now supports quoting column names with backticks to refer to names with spaces (GH6508)
- *merge_asof()* now gives a more clear error message when merge keys are categoricals that are not equal (GH26136)

- `pandas.core.window.Rolling()` supports exponential (or Poisson) window type (GH21303)
- Error message for missing required imports now includes the original import error's text (GH23868)
- *`DatetimeIndex`* and *`TimedeltaIndex`* now have a `mean` method (GH24757)
- *`DataFrame.describe()`* now formats integer percentiles without decimal point (GH26660)
- Added support for reading SPSS .sav files using *`read_spss()`* (GH26537)
- Added new option `plotting.backend` to be able to select a plotting backend different than the existing `matplotlib` one. Use `pandas.set_option('plotting.backend', '<backend-module>')` where `<backend-module` is a library implementing the pandas plotting API (GH14130)
- `pandas.offsets.BusinessHour` supports multiple opening hours intervals (GH15481)
- *`read_excel()`* can now use `openpyxl` to read Excel files via the `engine='openpyxl'` argument. This will become the default in a future release (GH11499)
- `pandas.io.excel.read_excel()` supports reading OpenDocument tables. Specify `engine='odf'` to enable. Consult the *IO User Guide* for more details (GH9070)
- *`Interval`*, *`IntervalIndex`*, and *`IntervalArray`* have gained an *`is_empty`* attribute denoting if the given interval(s) are empty (GH27219)

Backwards incompatible API changes

Indexing with date strings with UTC offsets

Indexing a *`DataFrame`* or *`Series`* with a *`DatetimeIndex`* with a date string with a UTC offset would previously ignore the UTC offset. Now, the UTC offset is respected in indexing. (GH24076, GH16785)

```
In [12]: df = pd.DataFrame([0], index=pd.DatetimeIndex(['2019-01-01'], tz='US/Pacific
↪'))

In [13]: df
Out[13]:
                           0
2019-01-01 00:00:00-08:00  0

[1 rows x 1 columns]
```

Previous behavior:

```
In [3]: df['2019-01-01 00:00:00+04:00':'2019-01-01 01:00:00+04:00']
Out[3]:
                           0
2019-01-01 00:00:00-08:00  0
```

New behavior:

```
In [14]: df['2019-01-01 12:00:00+04:00':'2019-01-01 13:00:00+04:00']
Out[14]:
                           0
2019-01-01 00:00:00-08:00  0

[1 rows x 1 columns]
```

`MultiIndex` constructed from levels and codes

Constructing a *`MultiIndex`* with `NaN` levels or codes value < -1 was allowed previously. Now, construction with codes value < -1 is not allowed and `NaN` levels' corresponding codes would be reassigned as -1. (GH19387)

Previous behavior:

```
In [1]: pd.MultiIndex(levels=[[np.nan, None, pd.NaT, 128, 2]],
   ...:               codes=[[0, -1, 1, 2, 3, 4]])
   ...:
Out[1]: MultiIndex(levels=[[nan, None, NaT, 128, 2]],
                   codes=[[0, -1, 1, 2, 3, 4]])

In [2]: pd.MultiIndex(levels=[[1, 2]], codes=[[0, -2]])
Out[2]: MultiIndex(levels=[[1, 2]],
                   codes=[[0, -2]])
```

New behavior:

```
In [15]: pd.MultiIndex(levels=[[np.nan, None, pd.NaT, 128, 2]],
   ....:               codes=[[0, -1, 1, 2, 3, 4]])
   ....:
Out[15]:
MultiIndex([(nan,),
            (nan,),
            (nan,),
            (nan,),
            (128,),
            (  2,)],
           )

In [16]: pd.MultiIndex(levels=[[1, 2]], codes=[[0, -2]])
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-16-225a01af3975> in <module>
----> 1 pd.MultiIndex(levels=[[1, 2]], codes=[[0, -2]])

/pandas-release/pandas/pandas/core/indexes/multi.py in __new__(cls, levels, codes,␣
↪sortorder, names, dtype, copy, name, verify_integrity, _set_identity)
    278
    279         if verify_integrity:
--> 280             new_codes = result._verify_integrity()
    281             result._codes = new_codes
    282

/pandas-release/pandas/pandas/core/indexes/multi.py in _verify_integrity(self, codes,␣
↪levels)
    352                 )
    353             if len(level_codes) and level_codes.min() < -1:
--> 354                 raise ValueError(f"On level {i}, code value ({level_codes.
↪min()}) < -1")
    355             if not level.is_unique:
    356                 raise ValueError(

ValueError: On level 0, code value (-2) < -1
```

`Groupby.apply` on `DataFrame` evaluates first group only once

The implementation of `DataFrameGroupBy.apply()` previously evaluated the supplied function consistently twice on the first group to infer if it is safe to use a fast code path. Particularly for functions with side effects, this was an undesired behavior and may have led to surprises. (GH2936, GH2656, GH7739, GH10519, GH12155, GH20084, GH21417)

Now every group is evaluated only a single time.

```
In [17]: df = pd.DataFrame({"a": ["x", "y"], "b": [1, 2]})

In [18]: df
Out[18]:
   a  b
0  x  1
1  y  2

[2 rows x 2 columns]

In [19]: def func(group):
   ....:     print(group.name)
   ....:     return group
   ....:
```

Previous behavior:

```
In [3]: df.groupby('a').apply(func)
x
x
y
Out[3]:
   a  b
0  x  1
1  y  2
```

New behavior:

```
In [20]: df.groupby("a").apply(func)
x
y
Out[20]:
   a  b
0  x  1
1  y  2

[2 rows x 2 columns]
```

Concatenating sparse values

When passed DataFrames whose values are sparse, *concat()* will now return a *Series* or *DataFrame* with sparse values, rather than a SparseDataFrame (GH25702).

```
In [21]: df = pd.DataFrame({"A": pd.SparseArray([0, 1])})
```

Previous behavior:

```
In [2]: type(pd.concat([df, df]))
pandas.core.sparse.frame.SparseDataFrame
```

New behavior:

```
In [22]: type(pd.concat([df, df]))
Out[22]: pandas.core.frame.DataFrame
```

This now matches the existing behavior of *concat* on Series with sparse values. *concat()* will continue to return a SparseDataFrame when all the values are instances of SparseDataFrame.

This change also affects routines using *concat()* internally, like *get_dummies()*, which now returns a *DataFrame* in all cases (previously a SparseDataFrame was returned if all the columns were dummy encoded, and a *DataFrame* otherwise).

Providing any SparseSeries or SparseDataFrame to *concat()* will cause a SparseSeries or SparseDataFrame to be returned, as before.

The `.str`-accessor performs stricter type checks

Due to the lack of more fine-grained dtypes, *Series.str* so far only checked whether the data was of object dtype. *Series.str* will now infer the dtype data *within* the Series; in particular, 'bytes'-only data will raise an exception (except for *Series.str.decode()*, *Series.str.get()*, *Series.str.len()*, *Series.str.slice()*), see GH23163, GH23011, GH23551.

Previous behavior:

```
In [1]: s = pd.Series(np.array(['a', 'ba', 'cba'], 'S'), dtype=object)

In [2]: s
Out[2]:
0      b'a'
1     b'ba'
2    b'cba'
dtype: object

In [3]: s.str.startswith(b'a')
Out[3]:
0     True
1    False
2    False
dtype: bool
```

New behavior:

```
In [23]: s = pd.Series(np.array(['a', 'ba', 'cba'], 'S'), dtype=object)
```

(continues on next page)

(continued from previous page)

```
In [24]: s
Out[24]:
0      b'a'
1     b'ba'
2    b'cba'
Length: 3, dtype: object

In [25]: s.str.startswith(b'a')
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-25-ac784692b361> in <module>
----> 1 s.str.startswith(b'a')

/pandas-release/pandas/pandas/core/strings.py in wrapper(self, *args, **kwargs)
   1951                       f"inferred dtype '{self._inferred_dtype}'."
   1952                   )
-> 1953                   raise TypeError(msg)
   1954               return func(self, *args, **kwargs)
   1955

TypeError: Cannot use .str.startswith with values of inferred dtype 'bytes'.
```

Categorical dtypes are preserved during groupby

Previously, columns that were categorical, but not the groupby key(s) would be converted to `object` dtype during groupby operations. Pandas now will preserve these dtypes. (GH18502)

```
In [26]: cat = pd.Categorical(["foo", "bar", "bar", "qux"], ordered=True)

In [27]: df = pd.DataFrame({'payload': [-1, -2, -1, -2], 'col': cat})

In [28]: df
Out[28]:
   payload  col
0       -1  foo
1       -2  bar
2       -1  bar
3       -2  qux

[4 rows x 2 columns]

In [29]: df.dtypes
Out[29]:
payload       int64
col        category
Length: 2, dtype: object
```

Previous Behavior:

```
In [5]: df.groupby('payload').first().col.dtype
Out[5]: dtype('O')
```

New Behavior:

```
In [30]: df.groupby('payload').first().col.dtype
Out[30]: CategoricalDtype(categories=['bar', 'foo', 'qux'], ordered=True)
```

Incompatible Index type unions

When performing *Index.union()* operations between objects of incompatible dtypes, the result will be a base *Index* of dtype `object`. This behavior holds true for unions between *Index* objects that previously would have been prohibited. The dtype of empty *Index* objects will now be evaluated before performing union operations rather than simply returning the other *Index* object. *Index.union()* can now be considered commutative, such that `A.union(B) == B.union(A)` (GH23525).

Previous behavior:

```
In [1]: pd.period_range('19910905', periods=2).union(pd.Int64Index([1, 2, 3]))
...
ValueError: can only call with other PeriodIndex-ed objects

In [2]: pd.Index([], dtype=object).union(pd.Index([1, 2, 3]))
Out[2]: Int64Index([1, 2, 3], dtype='int64')
```

New behavior:

```
In [31]: pd.period_range('19910905', periods=2).union(pd.Int64Index([1, 2, 3]))
Out[31]: Index([1991-09-05, 1991-09-06, 1, 2, 3], dtype='object')

In [32]: pd.Index([], dtype=object).union(pd.Index([1, 2, 3]))
Out[32]: Index([1, 2, 3], dtype='object')
```

Note that integer- and floating-dtype indexes are considered "compatible". The integer values are coerced to floating point, which may result in loss of precision. See *Set operations on Index objects* for more.

`DataFrame` groupby ffill/bfill no longer return group labels

The methods `ffill`, `bfill`, `pad` and `backfill` of `DataFrameGroupBy` previously included the group labels in the return value, which was inconsistent with other groupby transforms. Now only the filled values are returned. (GH21521)

```
In [33]: df = pd.DataFrame({"a": ["x", "y"], "b": [1, 2]})

In [34]: df
Out[34]:
   a  b
0  x  1
1  y  2

[2 rows x 2 columns]
```

Previous behavior:

```
In [3]: df.groupby("a").ffill()
Out[3]:
   a  b
0  x  1
1  y  2
```

New behavior:

```
In [35]: df.groupby("a").ffill()
Out[35]:
   b
0  1
1  2

[2 rows x 1 columns]
```

`DataFrame` describe on an empty categorical / object column will return top and freq

When calling *`DataFrame.describe()`* with an empty categorical / object column, the 'top' and 'freq' columns were previously omitted, which was inconsistent with the output for non-empty columns. Now the 'top' and 'freq' columns will always be included, with `numpy.nan` in the case of an empty *`DataFrame`* (GH26397)

```
In [36]: df = pd.DataFrame({"empty_col": pd.Categorical([])})

In [37]: df
Out[37]:
Empty DataFrame
Columns: [empty_col]
Index: []

[0 rows x 1 columns]
```

Previous behavior:

```
In [3]: df.describe()
Out[3]:
        empty_col
count           0
unique          0
```

New behavior:

```
In [38]: df.describe()
Out[38]:
       empty_col
count          0
unique         0
top          NaN
freq         NaN

[4 rows x 1 columns]
```

`__str__` methods now call `__repr__` rather than vice versa

Pandas has until now mostly defined string representations in a Pandas objects's `__str__`/`__unicode__`/`__bytes__` methods, and called `__str__` from the `__repr__` method, if a specific `__repr__` method is not found. This is not needed for Python3. In Pandas 0.25, the string representations of Pandas objects are now generally defined in `__repr__`, and calls to `__str__` in general now pass the call on to the `__repr__`, if a specific `__str__` method doesn't exist, as is standard for Python. This change is backward compatible for direct usage of Pandas, but if you subclass Pandas objects *and* give your subclasses specific `__str__`/`__repr__` methods, you may have to adjust your `__str__`/`__repr__` methods (GH26495).

Indexing an `IntervalIndex` with `Interval` objects

Indexing methods for *`IntervalIndex`* have been modified to require exact matches only for *`Interval`* queries. `IntervalIndex` methods previously matched on any overlapping `Interval`. Behavior with scalar points, e.g. querying with an integer, is unchanged (GH16316).

```
In [39]: ii = pd.IntervalIndex.from_tuples([(0, 4), (1, 5), (5, 8)])

In [40]: ii
Out[40]:
IntervalIndex([(0, 4], (1, 5], (5, 8]],
              closed='right',
              dtype='interval[int64]')
```

The `in` operator (`__contains__`) now only returns `True` for exact matches to `Intervals` in the `IntervalIndex`, whereas this would previously return `True` for any `Interval` overlapping an `Interval` in the `IntervalIndex`.

Previous behavior:

```
In [4]: pd.Interval(1, 2, closed='neither') in ii
Out[4]: True

In [5]: pd.Interval(-10, 10, closed='both') in ii
Out[5]: True
```

New behavior:

```
In [41]: pd.Interval(1, 2, closed='neither') in ii
Out[41]: False

In [42]: pd.Interval(-10, 10, closed='both') in ii
Out[42]: False
```

The *`get_loc()`* method now only returns locations for exact matches to `Interval` queries, as opposed to the previous behavior of returning locations for overlapping matches. A `KeyError` will be raised if an exact match is not found.

Previous behavior:

```
In [6]: ii.get_loc(pd.Interval(1, 5))
Out[6]: array([0, 1])

In [7]: ii.get_loc(pd.Interval(2, 6))
Out[7]: array([0, 1, 2])
```

New behavior:

```
In [6]: ii.get_loc(pd.Interval(1, 5))
Out[6]: 1

In [7]: ii.get_loc(pd.Interval(2, 6))
---------------------------------------------------------------------------
KeyError: Interval(2, 6, closed='right')
```

Likewise, *get_indexer()* and get_indexer_non_unique() will also only return locations for exact matches to Interval queries, with −1 denoting that an exact match was not found.

These indexing changes extend to querying a *Series* or *DataFrame* with an IntervalIndex index.

```
In [43]: s = pd.Series(list('abc'), index=ii)

In [44]: s
Out[44]:
(0, 4]    a
(1, 5]    b
(5, 8]    c
Length: 3, dtype: object
```

Selecting from a Series or DataFrame using [] (__getitem__) or loc now only returns exact matches for Interval queries.

Previous behavior:

```
In [8]: s[pd.Interval(1, 5)]
Out[8]:
(0, 4]    a
(1, 5]    b
dtype: object

In [9]: s.loc[pd.Interval(1, 5)]
Out[9]:
(0, 4]    a
(1, 5]    b
dtype: object
```

New behavior:

```
In [45]: s[pd.Interval(1, 5)]
Out[45]: 'b'

In [46]: s.loc[pd.Interval(1, 5)]
Out[46]: 'b'
```

Similarly, a KeyError will be raised for non-exact matches instead of returning overlapping matches.

Previous behavior:

```
In [9]: s[pd.Interval(2, 3)]
Out[9]:
(0, 4]    a
(1, 5]    b
dtype: object

In [10]: s.loc[pd.Interval(2, 3)]
```

(continues on next page)

(continued from previous page)

```
Out[10]:
(0, 4]    a
(1, 5]    b
dtype: object
```

New behavior:

```
In [6]: s[pd.Interval(2, 3)]
---------------------------------------------------------------------------
KeyError: Interval(2, 3, closed='right')

In [7]: s.loc[pd.Interval(2, 3)]
---------------------------------------------------------------------------
KeyError: Interval(2, 3, closed='right')
```

The *overlaps()* method can be used to create a boolean indexer that replicates the previous behavior of returning overlapping matches.

New behavior:

```
In [47]: idxr = s.index.overlaps(pd.Interval(2, 3))

In [48]: idxr
Out[48]: array([ True,  True, False])

In [49]: s[idxr]
Out[49]:
(0, 4]    a
(1, 5]    b
Length: 2, dtype: object

In [50]: s.loc[idxr]
Out[50]:
(0, 4]    a
(1, 5]    b
Length: 2, dtype: object
```

Binary ufuncs on Series now align

Applying a binary ufunc like `numpy.power()` now aligns the inputs when both are *Series* (GH23293).

```
In [51]: s1 = pd.Series([1, 2, 3], index=['a', 'b', 'c'])

In [52]: s2 = pd.Series([3, 4, 5], index=['d', 'c', 'b'])

In [53]: s1
Out[53]:
a    1
b    2
c    3
Length: 3, dtype: int64

In [54]: s2
Out[54]:
d    3
```

(continues on next page)

(continued from previous page)

```
c    4
b    5
Length: 3, dtype: int64
```

Previous behavior

```
In [5]: np.power(s1, s2)
Out[5]:
a      1
b     16
c    243
dtype: int64
```

New behavior

```
In [55]: np.power(s1, s2)
Out[55]:
a     1.0
b    32.0
c    81.0
d     NaN
Length: 4, dtype: float64
```

This matches the behavior of other binary operations in pandas, like *`Series.add()`*. To retain the previous behavior, convert the other `Series` to an array before applying the ufunc.

```
In [56]: np.power(s1, s2.array)
Out[56]:
a      1
b     16
c    243
Length: 3, dtype: int64
```

Categorical.argsort now places missing values at the end

`Categorical.argsort()` now places missing values at the end of the array, making it consistent with NumPy and the rest of pandas (GH21801).

```
In [57]: cat = pd.Categorical(['b', None, 'a'], categories=['a', 'b'], ordered=True)
```

Previous behavior

```
In [2]: cat = pd.Categorical(['b', None, 'a'], categories=['a', 'b'], ordered=True)

In [3]: cat.argsort()
Out[3]: array([1, 2, 0])

In [4]: cat[cat.argsort()]
Out[4]:
[NaN, a, b]
categories (2, object): [a < b]
```

New behavior