



owo what's this???

Team 24 - Deliverable 3

Table of Contents

Extensive Features	4
Overview	4
Selected Features	7
Issue 28925	7
Description	7
Analysis	8
Implementation and Design	8
Issue 29268	10
Description	10
Analysis	11
Acceptance Tests	11
Implementation & Design	13
Pandas Architecture	14
Overall Structure	14
Design Patterns	15



Extensive Features

Overview

Taking a deeper dive into the more complex issues of Pandas, we primarily focused on the issues with labels related to the API such as **API Design**, **API Consistency**, and/or **Performance**. As a team, we assembled again a document in the fashion of waterfall planning to scout out and pick the issues that we believe will challenge us, and benefit Pandas the most. Just like Deliverable 2, here is our selected list of issues that caught our attention,

1. <https://github.com/pandas-dev/pandas/issues/29309>

<API Design> <DataFrame>

This issue is related to creating a shallow copy of a DataFrame, which after some investigation, turns out to be unavailable in Pandas. Implementation doesn't seem too difficult, however more discussion by the Pandas team is required to finalize their decision, and the requirements of the feature.

2. <https://github.com/pandas-dev/pandas/issues/29268>

<API Design> <Named Aggregation>

This issue is related to the inability to aggregate data based on multiple columns to perform actions such as intermittent calculations while aggregating. This issue does however seem rather simple relative to some of the other issues in this list.

3. <https://github.com/pandas-dev/pandas/issues/28329>

<Performance> <Dataframe> <Benchmarks>

This issue is related to the slow performance of the correlation method, **DataFrame.corr()**, using the kendall algorithm. Currently the method isn't in



Cython, but uses **kendalltau** and **scipy** in a loop. Therefore the user proposes the method to be Cythonized.

4. <https://github.com/pandas-dev/pandas/issues/32593>

<Bug> <Algorithm> <DataFrame>

The issue is a bug in **DataFrame.rank()**, where when one of the columns is of type float, the rank of each column will not be equal - hence returns the wrong answer. Seems decent issue with an isolated error, some investigation is needed.

Note: This issue was taken by another D01 student before deliverable 3, hence is unavailable for us to do.

5. <https://github.com/pandas-dev/pandas/issues/25887>

<Performance> <DataFrame>

The issue is related to the performance of creating empty DataFrames taking longer than it should be. Hence since batches can be empty, speeding up this declaration would greatly speed up batches. This issue however is seen as a premature optimization and seems to be related more to the overhead of the DataFrame. Lots of investigation would be required as the response for this is, "if u think can be fixed w/on adding complexity then pls submit a PR" (jreback).

6. <https://github.com/pandas-dev/pandas/issues/26747>

<API Design> <Clean> <Visualisation>

This issue is related to creating an API to allow for other plotting libraries to be implemented in a simpler way via a common API. Needs to be looked at as a lot of work has been done on it, such as exploring the base created by the issuer.



7. <https://github.com/pandas-dev/pandas/issues/28925>

<API Design> <Series>

This issue is related to making Series chainable with the cut method. This change would require Series to use their own cut method instead of using the base function. This issue seems very doable however.

Note: This issue was taken by another D01 student during deliverable 3, hence is unavailable for us to do.

8. <https://github.com/pandas-dev/pandas/issues/26897>

<API Design> <Formatting>

The issue is related to adding string formatting options for DateTime. The issuer suggests creating a new function called **strfdelta** to DateTime which would be the timedelta_range equivalent of date_range's strftime. This issue seems relatively easy to do and specifications have been provided by the issuer.

A selected subset of these issues have been emailed to the TA for judgement, to ensure our issues are appropriate and aligned with the expectations of the course. We later refined our selection to the two issues we would tackle for the deliverable, and moved those new issues to our [Github board](#).

With the preliminary research completed, we would now explore the two issues which we have decided to take on for this deliverable.



Selected Features

Issue 28925


Original Issue: <https://github.com/pandas-dev/pandas/issues/28925>

Our Issue: https://github.com/CSCD01/team_24-project/issues/7

Team members: Yishuai Wang

[Issue 28925] Add cut as a Series method #7

Open SpectrumWings opened this issue 3 minutes ago · 0 comments



SpectrumWings

commented 3 minutes ago

Member

Description of Issue

User suggests that series to have the cut method instead of having it as a base function. Seems very doable.

Progress

☐ Define Estimate

☐ Propose Solution (and await feedback from team)

☐ Development

☐ Testing

☐ Final Review

Assignees

No one—assign yourself

Labels

Deliverable 3

Planning

Writing

Projects

None yet

Milestone

Description

This issue was created last October of 2019, and proposed an additional method to the series class instead of calling it as a base function in Pandas, passing in the data structure into it. This enhancement to the features is tagged with API Design, Series, and cut on the issue board. The output of the cut on the series and cut as a function would be the same.

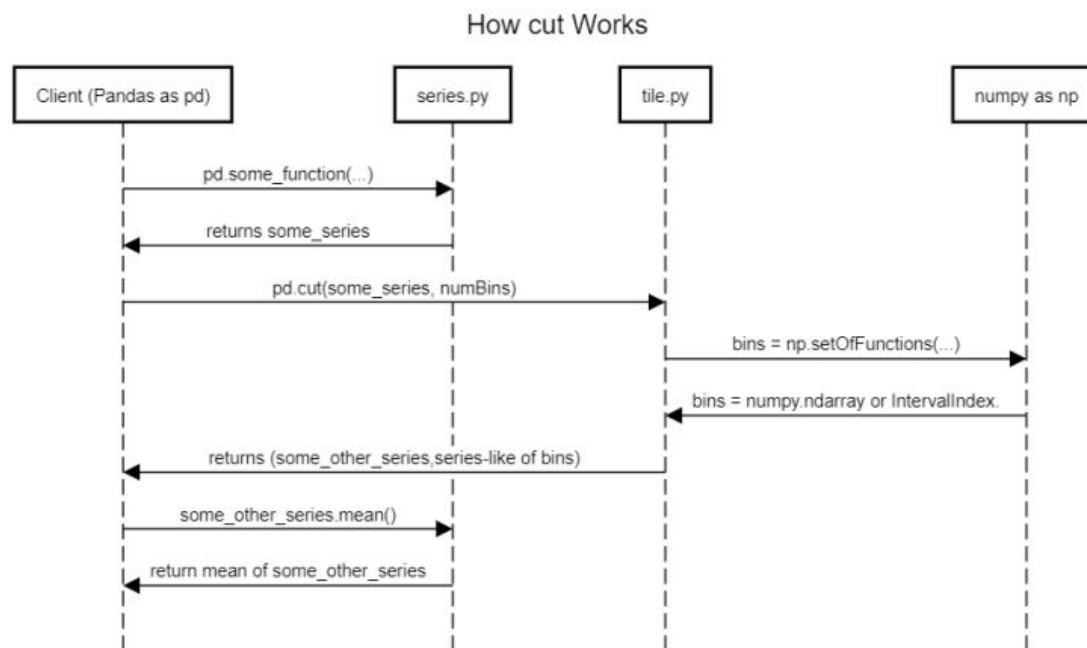
The cut function splits data into multiple bins and represents it as an array. Each bin is a range of values given a set of numbers, where each number will be put into a range. These ranges of values are generated as specified by the user via parameters. These parameters represent different criterias that the user can specify, some of these parameters are,



1. **bins [int]:** The size of each bin within a range provided
2. **bins [Sequence of scalars]:** The edges of the bins to allow various widths for each bin, allowing non uniform ranges
3. **duplicates:** Validation and cleaning for ensuring bin edges are unique

Analysis

We analyzed the trace of how cut is called, and what modules cut relies on to perform its function. From this analysis, the flow diagram of how the cut flows is as shown below,



As we can see, the cut function, with the series being cut and parameters, are passed to the cut function in [tile.py](#). The function then uses numpy to construct arrays of the requested size, processed by `tile.py`, then returned.

Implementation and Design

To tackle this issue, we looked at the implementation of cut within the `tile.py`. We are moving this function to the series class located [here](#). In the series class, we

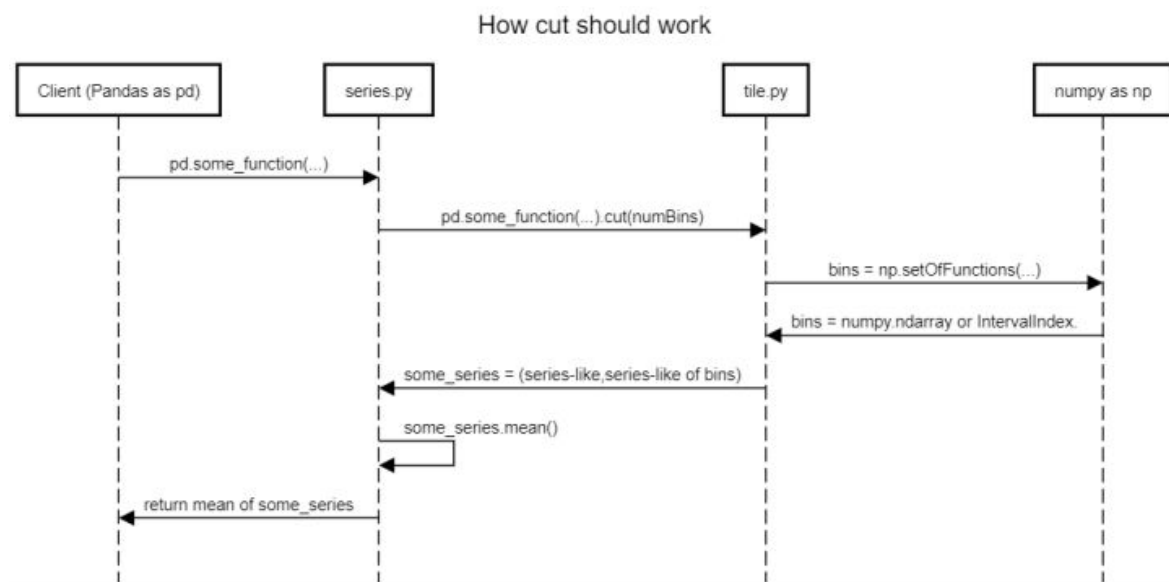


Series
+ data: array
+ index: int
+ dtype: string
+ name: string
+ copy: boolean
- init(): void
+ initDict(): data, index
- set_axis(): void
+ ravel(): numpy
+ view(): series
+ repeat(): array of int
+ reset_index(): dataframe
...

can add an additional method into this class. The method would be an import of the cut function, which maps the data from series and parameters to the cut function. This would ensure that we can perform customizations while adhering to consistency of the cut function itself.

The other parameters that need to be passed into cut are provided by the caller, and furthermore, the calls to tile and numpy are independant and calls are made regularly as needed.

With the changes that we plan to make, the overall flow of the call for cut changes. As shown in the flow diagram below, we will be making much less calls to other classes, and overall simplifying the process highlighted below.



Issue 29268

Original Issue: <https://github.com/pandas-dev/pandas/issues/29268>

Our Issue: https://github.com/CSCD01/team_24-project/issues/6

Team members: William Granados, Jaden Wang

[Issue 29268] Named aggregations with multiple columns #6

 Open SpectrumWings opened this issue 5 days ago · 0 comments



SpectrumWings commented 5 days ago · edited ▾

Member



Description of Issue

User explains if you try to aggregate data based on multiple columns, you cannot do intermittent calculations while aggregating. Slightly simpler than other issues.

Progress

- ☐ Define Estimate
- ☐ Propose Solution (and await feedback from team)
- ☐ Development
- ☐ Testing
- ☐ Final Review

Assignees

No one—ass

Labels

Deliverable

Planning

Projects



Delive
To Do

Description

This issue was created in October 2019, and a similar issue created a few months prior was also redirected to this one - which hints at a promising and impactful feature as it affects multiple audiences. The feature is an enhancement to the capability of the groupby functionality - allowing for aggregations of multiple columns, to one aggregated column.

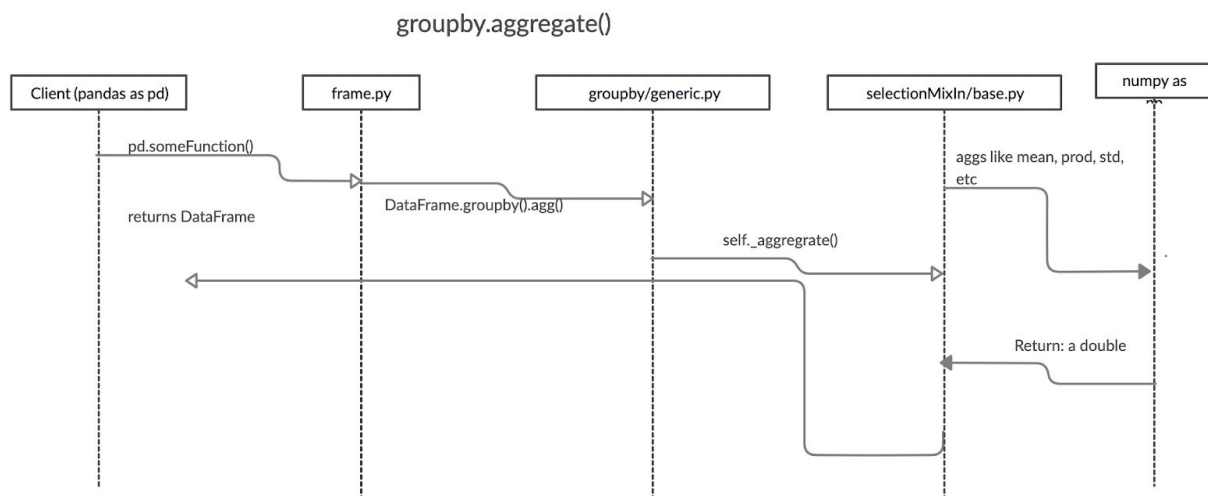
Aggregations are normally done per column in conjunction to groupby, which collapses multiple rows into a single row. In the DataFrame, which we can treat as series or mathematical vectors, where we can change the value of the single row by obtaining the - min/max, sum, mean, etc - from the grouped rows. These aggregations can also be lambda functions, allowing for more complicated and



customized manipulation to the data - such as calculating differences between max and min.

Analysis

We analyzed the trace of how aggregation is called, and what modules aggregation relies on to perform its function. From this analysis, the flow diagram of how the aggregate flows is shown below,



From our analysis, we can see that the bulk of the functionality by the aggregate function is done in the [SelectionMixin](#) class, where aggregations are handled for the DataFrame done through Python builtins and/or numpy.

Acceptance Tests

The following is an example usage of currently available aggregations, with the additional requested feature in the form of the diff_a_b aggregation.



```
# Sample data and it's result
df = pd.DataFrame(np.random.rand(4,4), columns=list('abcd'))
df['group'] = [0, 0, 1, 1]

      a         b         c         d  group
0  0.751462  0.572576  0.192957  0.921723    0
1  0.070777  0.801548  0.601678  0.344633    0
2  0.112964  0.361984  0.416241  0.785764    1
3  0.380045  0.486494  0.000594  0.608759    1

# Aggregations on single columns
df.groupby('group').agg(
    a_sum=('a', 'sum'),
    a_mean=('a', 'mean'),
    b_mean=('b', 'mean'),
    c_sum=('c', 'sum'),
    d_range=('d', lambda x: x.max() - x.min())
)
# Result
      a_sum      a_mean      b_mean      c_sum      d_range
group
0      0.947337  0.473668  0.871939  0.838150  0.320543
1      0.604149  0.302074  0.656902  0.542985  0.057681

# Proposed functionality
df.groupby('group').agg(
    diff_a_b=(['a', 'b'], lambda x: x['a'].max() - x['b'].max())
)
```

We can see that for the `a_sum` aggregation on group 0 in the output it is simply `0.69 + 0.45 ≈ 1.15`, and similarly for the `a_sum` group 1 the sum is `0.62 + 0.11 ≈ 0.74` - with the other function behaving similarly and as expected.



SelectionMixin
<ul style="list-style-type: none"> - cython_table: dict: func -> str { builtins.sum: "sum", np.sum: "sum", ..} - builtin_table: dict: func -> func {builtin.sum: np.sum, builtin.max: np.max, ...}
<ul style="list-style-type: none"> - try_aggregate_string_function: func - _aggregate: (float, Bool) - _aggregate_multiple_funcs: list of (float, bool) - get_cython_function: Optional[str] - get_builtin_func: func

The requested functionality will be similar to `d_range`, where a lambda function is provided, however a list of columns would be provided. As a result, the lambda function parameter will be a dictionary of the specified columns, and can be manipulated as required.

The acceptance testing should cover the ability to have our manual example of `diff_a_b` done programmatically.

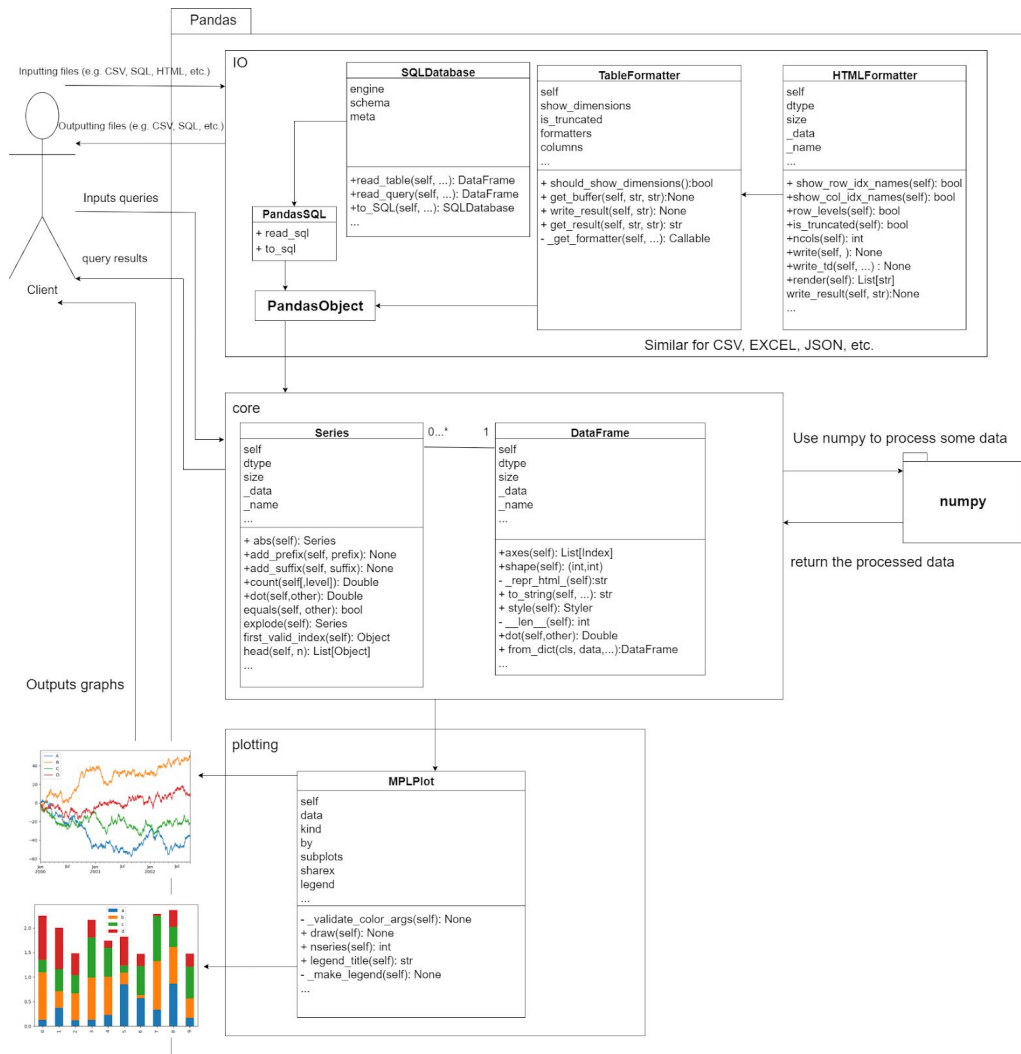
Implementation & Design

The implementation of this new feature would involve the modification of the kwargs of `agg` to allow for a list of columns and a single lambda function - where the lambda is a function of one or more columns from the DataFrame. The aggregate function can be found [here](#), where modifications to multiple funcs and a single func is required to ensure consistency. These changes would all happen within [SelectionMixin.aggregate](#) and [SelectionMixin.aggregate_multiple_func](#), to cover both cases of when a list of functions or a single function is provided to the method.



Pandas Architecture

Overall Structure



Pandas can be represented by four main components - io, core, plotting, and numpy. The data flows through the four main components in the following order,

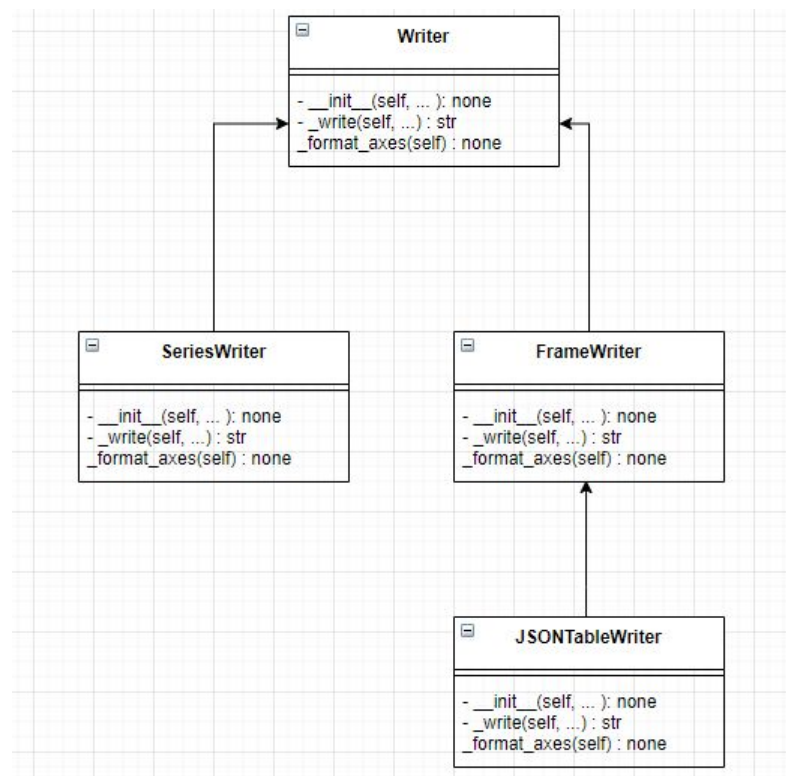
- **pandas/io:** Handles the inputs and outputs of files such as csv, SAS, JSON, etc. This component is responsible for reading several types of files,



identifying and formatting them into a pandas object to be further used by pandas/core.

- **pandas/core:** Handles the representation of data used by pandas, such as Series and DataFrame. These representations act as an API and wrapper to connect the other three main components together.
- **numpy:** Handles most of the processing and manipulation of data and works heavily with core - as can be seen from it's wide usage throughout several classes in core.
- **pandas/plotting:** Handles the data visualization in the form of charts and tables. Once these objects are fully processed by numpy, they can be used with Matplotlib, an integrated component of pandas/plotting, for creating the data visualizations.

Design Patterns



Within **pandas/io**, more specifically [/json/_json.py](#), there is a usage of the factory design pattern. The Writer class acts as the interface, with its children acting as the different products of the factory. The **_write** function is the focus of the design pattern, which is the only method which returns not none. The init function is also important to this design pattern, as each object uses a different initialization process.

