owo what’s this???

Team 24 - Deliverable 2

Table of Contents

Issues and Tickets

Overview

Pandas is abundant with issues from feature requests and bugs to choose from. However, after further analyzing the issue board, we found that many of the issues overlap, go silent, or have been completed and merged without any mention. As a result, the board is bloated with stale and abandoned issues, which are fairly difficult to find and revisit as the issuer giver and contributors have long forgotten the issue. In our designing and planning, these are the issues that we've found as potential candidates in a document:

1. https://github.com/pandas-dev/pandas/issues/23988 **<Bug>**
   Some guy seems to have this mostly done, but its been dead for a while. Maybe we can scope this one up

2. https://github.com/pandas-dev/pandas/issues/19827 **<Bug> <IO>**
   Some people tackled it but it seems like it's dead. Issue also seems inconsistent between different users

3. https://github.com/pandas-dev/pandas/issues/18198 **<Bug> <IO> <SAS>**
   Seems like a very good issue to tackle. SAS file being read with nothing inside gives an object of None. Doesn't seem like anyone is working on it. However, I'm not sure if this issue is already overlapped by other issues and releases.

4. https://github.com/pandas-dev/pandas/issues/31819 **<IO> <Bug>**
   Seems not resolved. Issues with read_csv for file objects. One person does seem to be tackling it however.

5. https://github.com/pandas-dev/pandas/issues/24518 **<Depreciate>**
   Doesn't seem done. Remove functions. You may have to go through the depreciation cycle that they've documented

6. https://github.com/pandas-dev/pandas/issues/23164 **<Bug> <Array>**
   Function not functioning as stated by documentation. Doesn't seem like anyone is doing it yet since its from Oct 2018

7. https://github.com/pandas-dev/pandas/issues/32392 **<Bug> <CSV>**
   Very recent ticket (within a day). Bug reading csv file from google cloud storage

8. https://github.com/pandas-dev/pandas/issues/28572 **<GroupBy> <Series>**
   Groupby when the corresponding value is zero does not show up

9. https://github.com/pandas-dev/pandas/issues/18414 **<Bug> <Sparse>**
   A KeyError is raised when attempting to index a column

10. https://github.com/pandas-dev/pandas/issues/9314 **<Feature> <Enhancement>**
    Old ticket. Enhance date_range's freq parameter to take constants from dateutil

11. https://github.com/pandas-dev/pandas/issues/32127 **<Bug> <Arrays>**
    Qcut precision parameter inconsistency. Behaviour of Qcut seems to result in a different number of floating decimals of the result.

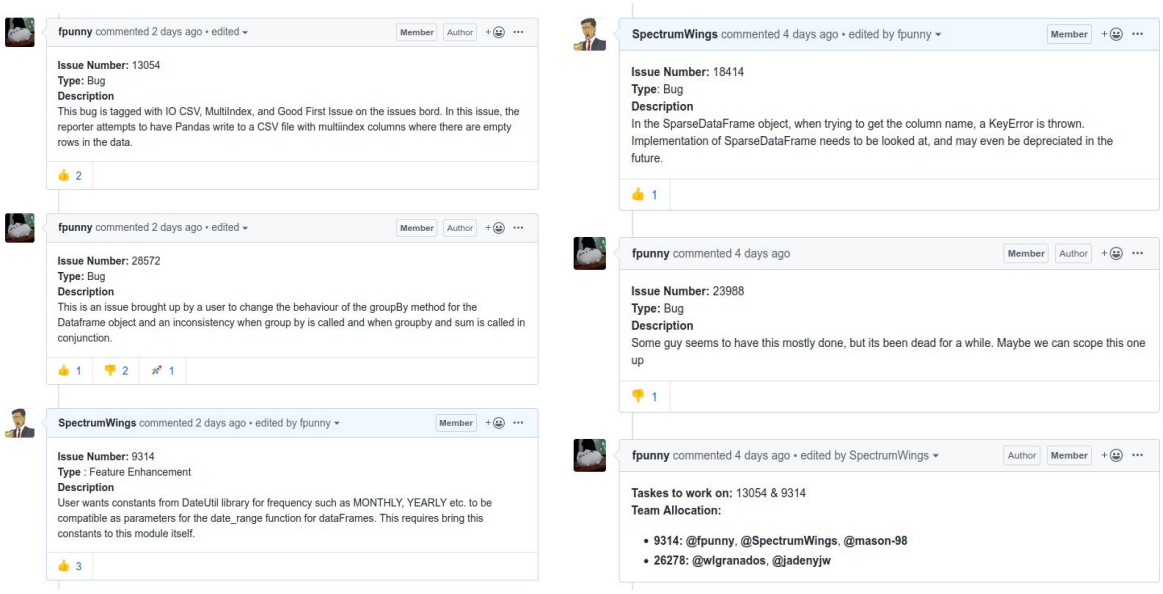
12. https://github.com/pandas-dev/pandas/issues/26278 **<Feature> <LATEX>**
    Users want to automatically format columns in LATEX with a single command instead of multiple lambda functions.

The Process

Using Github projects and issues, we created a board, which encapsulates the iteration as an epic, to document our discussion and progress via issues - which compared to tickets, allow for commenting and adding descriptions to help create a hierarchy for our progress. Following our plan of **waterfall**, we've proceeded with analysis and design of the issues that we are planning to tackle for this deliverable phase, finding issues which are not overshadowed by other issues and are

appropriate to tackle during this phase, before assigning members to the issue to analyze the issue further in depth.

Each member contributed to finding these issues by providing the link, with a couple tags to give a general idea of the issue, and a brief description of what is wanted from the issue. The list above represents an excess of tickets we plan to consider for this and future deliverables to help expedite the analysis and design stage. After some research, we narrowed down the list to 5 issues in the Github project, and picked the two final issues to tackle for this deliverable.



After our requirements gathering stage, we proceeded to allocate the members of the team into the finalized two issues - where each team would review the issue further, create an estimate, and proceed with implementing, testing, and documenting the process on their accord. When a task is completed, then the issue is closed, and the ticket within the project board is moved accordingly.

Issue Descriptions

From the list above, we further narrowed down the list into a smaller set of 5 issues, which we would go into further detail. From this list of 5, we selected two to

implement and discuss even further in later sections of this deliverable. The 5 issues are we would explore are as follows,

1. https://github.com/pandas-dev/pandas/issues/26278
   <LATEX> <Feature>

   The feature was requested by a user who wanted to format the columns of data, which would be written to a LATEX file. The current solution requires the user to define a lambda function for each column to apply the format as shown below,

```
# Create dataframe
data = [[1,2,3], [4,5,6], [7,8,9.001]]
df = pd.DataFrame(
    data,
    columns=['a', 'b', 'c'],
    index=["foo", "bar" , "foobar"]
)

# Create lambdas for formatting each column
cm0 = lambda x:'%1i'   %x
cm1 = lambda x:'%1.1f' %x
cm3 = lambda x:'%1.3f' %x
format = [cm0, cm1, cm3]

# Apply format
latex_tab = df.to_latex(formatters=format)
```

   The user proposed an alternate method to help streamline the formatting process, into simply one line, using a formatters_col parameter as shown below. The issue itself is said to be “reasonable and probably not a lot of effort”, and is labelled as nice to have, but has been abandoned and untouched since May 2019.

```
latex_tab = df.to_latex(
    data,
    formatters_col=['%1i','%1.1f','%1.3f']
)
```

After analyzing the issue, we agreed that this issue wasn't too demanding and is suitable for a first issue for us to tackle. This issue affects the to_latex() method of the dataFrame module and we estimate it would take around 1 to 2 days for us to complete it.

2. **https://github.com/pandas-dev/pandas/issues/9314**
   <Feature> <Enhancement>

   The feature was requested by a user who found an inconsistency in the freq param for date_range. One of Pandas' dependencies, dateutil, has an equivalent feature, rrule, which takes in constants of: YEARLY, MONTHLY, WEEKLY, DAILY, HOURLY, MINTELY, SECONDLY. However, freq accepts currently only strings, which is a discrepancy to the dependency. Therefore the proposed solution was to include these constants for freq, alongside string arguments.

   In this thread, developers discussed since these constants are integers and not strings, the function would need to accept integers as well. A proposed solution would be to introduce enums to help mitigate this issue. The issue itself is labelled to be nice to have, however has been abandoned and untouched since June 2018.

   After analyzing the issue, we agreed that most of the work comes from analysing how the parameter frequency is processed in /core/indexes/datetimes.py, and how these constants from DateUtil will be accessed. We estimate that this task will take 3 days as discovering a good way to implement this feature will take some time.

3. https://github.com/pandas-dev/pandas/issues/28572

<GroupBy> <Series>

The issue was found by a user who noted missing values when grouping by multiple fields and sum is called in conjunction. When multiple fields are grouped and the sum is extracted, the fields with no data are not present in the result. However, if grouped only by one field, the result would contain the field with a value of 0.

The example used was simply a 3 column - Dates, Sales, Product - table where the date is every day in a year, product is either A, B, or C, and value is a random number between 0-100. All the data for March was deleted for this example, however any month should produce the same result.

When a multi group is performed, grouping by Products and Date by month, followed by a sum of sales, the date of march is not present in the result. However, if the data is only grouped by Product, then march appears as expected with a value of 0.

```
monthly = pd.Grouper(key='Dates', freq=M)

# The result of multi grouping
sum_sales = df.groupby(['Product', monthly])['Sales'].sum()

Product  Dates
A        2001-01-31    658
         2001-02-28    460
         2001-04-30    541
         2001-05-31    701
         2001-06-30    517
         2001-07-31    596
         2001-08-31    802
         2001-09-30    654
         2001-10-31    561
         2001-11-30    473
```

```
         2001-12-31    605


# The result of grouping only by one field
sum_sales = df.groupby(monthly)['Sales'].sum()

Dates
2001-01-31    1616
2001-02-28    1256
2001-03-31       0
2001-04-30    1555
2001-05-31    1384
2001-06-30    1451
2001-07-31    1677
2001-08-31    1472
2001-09-30    1535
2001-10-31    1316
2001-11-30    1573
2001-12-31    1403
```

After analyzing the issue, we believe that this issue would be fairly complicated as more components of the code are used in combination to produce this result. Hence, we have to explore the extent of the inconsistency (what other inputs would cause this), as well as familiarize ourselves with the code and behaviour of groupby and Dataframe. Hence, we estimate around 4 days are required to complete this issue.

4. **https://github.com/pandas-dev/pandas/issues/18414**

   <Bug> <Sparse>

   The issue was found by a user who noted the functionality of retrieving the type of the first column of a DataFrame representing a sparse matrix. In the issue, the user tried to declare a DataFrame, convert it into a parse, and call to_coo, which converts the DataFrame into a coordinate representation of

a sparse matrix. This normally would return the sparse matrix but instead raises a KeyError, which is unexpected and unintended.

```
t_df = idx = pd.Int64Index([2,3,4])
t_df = pd.DataFrame(data=0, columns=idx, index=idx)
t_df.apply(pd.SparseArray).sparse.to_coo() # Throws KeyError
from find_common_types
```

The community depreciated and eventually retired SparseDataFrame. However, the issue still exists in another module, dataFrame[sparse], and the issue has been updated to it as such. This issue was picked up last February and a 5 character "fix" has been attempted. However, it has failed to pass the test suite and doesn't seem like it would be completed and/or revisited.

After analyzing the issue, we believe it will take approximately 2 days to complete from analyzing the DataFrame, and exploring the is_coo method for issues with accessing dictionaries and/or arrays relating the first column of the matrix - which would justify the KeyError from the first column type.

5. https://github.com/pandas-dev/pandas/issues/32127

   <Bug> <Arrays>

The issue was found by a user using a DataFrame method qcut, which is used for applying continuous values onto discrete values. When this method is used with precision, which changes how many significant digits the continuous value has, the continuous values go beyond the specified precision - as shown in the example below.

```
# Generate one random value for discrete data set
np.random.seed(42)
df = pd.DataFrame({"a": np.random.rand(20)})

# Apply map continuous data into one discrete value
for prec in range(1,10):
    print(f"Prec: {prec}, first interval: {pd.qcut(df['a'],
```

```
q=10, precision=prec).cat.categories[0]}")

# Result
Prec: 1, first interval: (0.011000000000000001, 0.15]
Prec: 2, first interval: (0.011000000000000001, 0.15]
Prec: 3, first interval: (0.0196, 0.146]
Prec: 4, first interval: (0.02048, 0.1462]
Prec: 5, first interval: (0.020574000000000002, 0.1462]
Prec: 6, first interval: (0.020583499999999998, 0.146203]
Prec: 7, first interval: (0.02058439, 0.1462034]
Prec: 8, first interval: (0.020584483999999997, 0.14620343]
Prec: 9, first interval: (0.0205844933, 0.14620343]
```

As we can see from precision - 1, 2, 5, 6, 8 - that the continuous value is completely out of the allocated precision value. According to the issuer, the first two values may be justified since it might not be possible “to create enough intervals with low precision”. However, a warning/error should have been raised in response.

After analyzing the issue, we believe that it would take around 4-5 days to complete this task as no one from the community has isolated the exact issue for this (perhaps a roundoff error? Pancer would not be impressed). We think that perhaps the issue may originate from DataFrame and/or dependency which manages the rounding and/or calculation of this.

Issues Completed

Rationale for issues

Before selecting these issues for this deliverable, we had considered a few points. The issues we selected had to be implementable with our process within two weeks. This cuts down on the amount of time we have to implement the code since requirements gathering, designing, and testing would take significantly more time due to our unfamiliarity with the codebase, codebase processes, and our own waterfall process. Therefore, the issues should be relatively easy to help focus our time on the process to help expedite the process in preparation for more challenging issues in later deliverables.

Issue 26278

Original Issue: https://github.com/pandas-dev/pandas/issues/26278
Our Issue: https://github.com/CSCD01/team_24-project/issues/5
Team members: William Granados, Jaden Wang

Identifying the Issue

To recap on our description in the first section, the issue is a request for an enhancement to formatters easier to apply when exporting a Pandas DataFrame to Latex by reducing the amount of repeated work to change the column formatters in Latex. By tracing the project we were able to identify that **to_latex()** in

pandas/core/generic.py is where the function is defined. In this function, we implemented a new **formatter_col** parameter to create a compatible list which would be applied to **formatters**, which is used by the original formatter parameter.

Testing the Code

The files changed were **generic.py** in the **pandas/core** folder, therefore an additional test case is needed for the corresponding test suite - **pandas/tests/io/formats/test_to_latex.py**.

Since this is a new feature, our acceptance test consisted of us running an example usage of the problem, and including this as a unit test in the testing library. We also made sure to run the rest of the tests in the suite to make sure we weren't incurring any regressions in the codebase.

```
(pandas-dev) dankpro-2:pandas william$ python3 -m pytest ./tests/series/test_io.py
============================================================ test session starts ============================================================
platform darwin -- Python 3.7.6, pytest-5.3.5, py-1.8.1, pluggy-0.13.0
rootdir: /Users/william/github/pandas-team24, inifile: setup.cfg
plugins: asyncio-0.10.0, forked-1.1.2, xdist-1.31.0, hypothesis-5.6.0, cov-2.8.1
collected 31 items

tests/series/test_io.py ...............................                                                                                    [100%]

============================================================= 31 passed in 1.45s ============================================================
```

Issue 9314

Original Issue: https://github.com/pandas-dev/pandas/issues/9314
Our Issue: https://github.com/CSCD01/team_24-project/issues/3
Team members: Mason Tang, Yishuai (Alex) Wang, Frederic Pun

Identifying the issue

To recap on our description in the first section, the issue is a request for an enhancement to apply constants for the freq parameter of date_range to make it more consistent with the rrule function dateUtil. By tracing the project we were able to identify that to_offset in **pandas/tseries/frequencies.py** is where the freq is processed and would return a Pandas' offset object. So we created a new case in to_offset for int values so that it would match the constants from DateUtil as follows (0 = YEARLY, 1 = MONTHLY, ... 6 = SECONDLY). Error processing is also included in this change to process invalid integer inputs.

Testing the Code

The file changed in this issue was **test_to_offset.py**, located in the **pandas/test/tseries/frequency** module. We added a test case for each new accepted value in to_offset and also tested for some invalid integers that may also be entered. The second image shows how these enhancements have not broken anything in date_range. (1 skipped due to running a window operating system with python 3)

```
(pandas-dev) C:\Users\Mason\Desktop\D01\pandas-team24>pytest ./pandas/tests/tseries/frequencies/test_to_offset.py
============================================= test session starts =============================================
platform win32 -- Python 3.7.6, pytest-5.3.5, py-1.8.1, pluggy-0.13.0
rootdir: C:\Users\Mason\Desktop\D01\pandas-team24, inifile: setup.cfg
plugins: hypothesis-5.6.0, asyncio-0.10.0, cov-2.8.1, forked-1.1.2, xdist-1.31.0
collected 92 items

pandas\tests\tseries\frequencies\test_to_offset.py ............................................................ [ 66%]
...............................                                                                                  [100%]

============================================== 92 passed in 0.18s ==============================================
```

```
(pandas-dev) C:\Users\Mason\Desktop\D01\pandas-team24>pytest ./pandas/tests/indexes/datetimes/test_date_range.py
============================================= test session starts =============================================
platform win32 -- Python 3.7.6, pytest-5.3.5, py-1.8.1, pluggy-0.13.0
rootdir: C:\Users\Mason\Desktop\D01\pandas-team24, inifile: setup.cfg
plugins: hypothesis-5.6.0, asyncio-0.10.0, cov-2.8.1, forked-1.1.2, xdist-1.31.0
collected 105 items

pandas\tests\indexes\datetimes\test_date_range.py ...s...................................................... [ 59%]
.........................................                                                                        [100%]

========================================== 104 passed, 1 skipped in 3.73s ==========================================
```

Testing our Changes

Environment setup

When working on a project in python, it's important that an environment is setup to help ensure that dependencies between other projects, and the user's computer, are isolated from one another. After setting up the environment, the development version of Pandas can be compiled and loaded into the environment, and can be worked on freely without rebuilding. The process for this can be found in this link, https://pandas.pydata.org/docs/development/contributing.html. However, here is a summary on how to get setup,

1. First the codebase, which in our case is https://github.com/CSCD01/pandas-team24, must be cloned. Our branches are this deliverable can be found as "26278" and "9314".

2. The C compiler is also required as an external dependency since parts of the code run on Cython to help speed up certain operations. Other optional dependencies, such as numpy and jupyter, are nice to have for debugging.

3. Anaconda is required for setting up the environment. When using Anaconda, make sure it's updated with the command, `conda update conda.`

4. Navigate into the cloned repo and create a virtual environment using the command `conda env create -f environment.yml` followed by `conda activate pandas-dev` to enter the environment.

5. Build and install Pandas from the branch

```
python setup.py build_ext --inplace -j 4
python -m pip install -e . --no-build-isolation --no-use-pep517
```

6. To test if Pandas has been successfully built and installed, import pandas in the Python shell and run `print(pandas.__version__) pytest`

Testing the individual issues

Now that your environment has been set up we can now test our issues. We created a suite of tests for each issue that can be run separately to ensure the functionality of the fixes we made. These tests cover the functionality of **date_range()** and **to_latex()** methods of DataFrame. If a more thorough testing is required, such as full regression testing, run `pytest pandas`. (But be warned that this apparently takes around 20 minutes to run)

Issue 9314

```
git checkout -b 9314
pytest pandas/tests/tseries/frequencies/test_to_offset.py
```

Issue 26278

```
git checkout -b 26278
pytest pandas/tests/series/test_io.py
```