CSCD01 Group Project

Deliverable 3

Zhiqi Chen
Anika Sultana
Christina Ma
Gabriela Esquivel Gaghi
Hyun Woo (Eddie) Shin
Arthur Lu
Man Hei Ho

Table of Contents

Three Chosen Issues

Issue 1: [ENH] add math mode to formatter escape="latex" #50040

Member Tasks

- Zhiqi Chen: Implemented formatter escape ‘latex-math’ and wrote documentation.
- Christina Ma: Implemented tests for formatter escape ‘latex-math’, added additional README documentation on how to test pandas.

Implementation Documentation

Formatter with option `escape='latex'` has a bug where it does not properly preserve content in latex math mode (e.g. `$x=3$`) that does not need to be escaped and also content which needs to be escaped; instead, either everything is escaped or nothing is. The solution of this issue is then to add `escape="latex-math"` as an option. It will escape normal content and exclude the `$` and any content between 2 `$` (i.e. LaTeX inline math mode) from escaping if this string is supplied. Meaning that if the user chooses the option `escape='latex-math'` then any Latex content that has inline math should be properly escaped.

Modifications were made to `pandas/io/formats/style_render.py` for implementing this solution. The following were added: Complete documentation of the new option and example of its use were added. Implemented call of the function `_escape_latex_math(s)` as an additional escape option and its error message. Implemented the `_escape_latex_math(s)` which returns a properly escaped string with any latex content with inline-math and documentation of this function.

Modifications were made to `pandas/tests/io/formats/style/test_format.py` for the testing of the new `escape='latex-math'`. Implemented unit tests and that properly verifies the validity of `escape='latex-math'`. There are 7 test cases in total located in the newly created function, `test_format_escape_latex_math()`. Each test has a comment indicating the type of case that is being tested for. The ”`pandas-copy/acceptance_tests/issue1.py`” file provides 3 example acceptance tests that follow the steps outlined below under Acceptance Test. This file displays how the user would use the new formatter option, and there is a test check to check for correctness.

All in all, the changes were made to `style_render.py` which added the escape option ‘latex-math’ and documentation, and implemented testing in `test_format.py`. No other significant changes were made to the codebase/design of pandas.

Acceptance Test

Test Objective: To ensure that the formatter converts the text into LaTex while also preserving any text in math mode.

Test Scenario: A user wants to use “latex-math” to format their text

Test Steps:

1. The user imports pandas into their file
2. The user creates a pandas DataFrame with their desired data
3. The user uses these methods `` `style.format(escape='latex-math').to_latex()` `` to format their data into LaTex.
4. The user retrieves the output from Step 3 and pastes it into their LaTex document.

Test Result: If the user can complete the steps without encountering any errors and the conversion to LaTex matches exactly how the data should be displayed, then the “latex-math” option has passed the user acceptance test.

There are 3 example acceptance tests in `` `pandas-copy/acceptance_tests/issue1.py` ``.

File Changes Made

`pandas-copy/pandas/core/config_init.py`

- Line 876 - To include “latex-math” as a new escape option

`pandas-copy/pandas/io/formats/style_render.py`

- Lines 991-992, 1214-1215, 1318-1326 - To include “latex-math” as a new escape option in documentation
- Lines 1761-1762 - Add “latex-math” as escape option
- Line 1765 - Modify error message to include “latex-math” option
- Lines 2366 - 2394 - Implementation of “latex-math”

`pandas-copy/pandas/tests/io/formats/style/test_format.py`

- Lines 194 - 249 - Added the function, `test_format_escape_latex_math()`, which has 7 unit tests
- Line 419 - Modified `test_str_escape_error()` to include “latex-math” in error message

`pandas-copy/acceptance_tests/issue1.py`

- File contains 3 example acceptance tests

Issue 2: [BUG] preserve the dtype on Series.combine_first #51764

Member Tasks

- Gabriela Esquivel Gaghi: Implemented the fix for preserving dtype on Series method 'combine_first()' and wrote documentation for the implementation.
- Arthur Lu: implemented unit tests and acceptance tests for dtype and value after using Series method combine_first().

Implementation Documentation

Using the `combine_first()` function on a `Series` results in automatic conversion of the original data type (converting a series of `int64` into `float64`). The expected behavior is that the datatype should be preserved, which works correctly when using `combine_first()` on a `DataFrame`. Therefore, we would have to make changes in `pandas/core/series.py` similar to the implementation in `pandas/core/frame.py`, as well as add test cases that check the correctness of the data type and the values of the `Series`.

```
import pandas as pd

pd.Series([4, 5]).combine_first(pd.Series([6, 7, 8]))
0    4.0
1    5.0
2    8.0
dtype: float64

pd.Series([4, 5]).to_frame().combine_first(pd.Series([6, 7, 8]).to_frame())[0]
0    4
1    5
2    8
Name: 0, dtype: int64
```

The above example shows how combining two integer `Series` resulted in a `Series` of data type `float64`, while doing the same operation after converting the `Series` to `DataFrame` preserved the data type as `int64` as wanted.

The issue came from reindexing `self` with the union of both series' indices, which filled any missing indices in `self` with an appropriate `NaN` value, turning series with dtype `int` to dtype `float`. This would happen whenever `other` contained indices that were not in `self`. Later, when the previous implementation filled those `NaN` values in `self` with the corresponding values from `other`, the resulting series would have dtype float even if all `NaN` values were replaced.

The first solution tried was to simply downcast the series to its original dtype when possible. However, this did not work for large integers, resulting in loss of precision from the dtype conversions. Therefore, to implement the fix, the algorithm was changed to avoid creating `NaN` values that were not already `self`. The new implementation does this by instead taking only the values of `other` for the indices where `self` is missing or `NaN` (since those are the only values we want from `other`), and appending them to the non-missing values of `self`. Only then, it reindexes the result using the union of the original `self` index with the `other` index, to ensure that the index of the series being returned matches the expected index for the combined series. This way, we avoid inserting temporary `NaN` values into `self`, and preserve its dtype. Lastly, the method also needed to call `__finalize__` to make sure that no metadata of the `self` series is lost (such as the series `name` attribute in the datetime timezone tests).

Acceptance Test

Test Objective: to ensure that the user's data is not converted to another type unexpectedly when working with `Series.combine_first()`.

Test Scenario: a user is trying to use `combine_first()` on two series of `int64`s.

Test Steps:

1. The user imports pandas into their file
2. The user creates two pandas `Series` containing integer values
3. The user uses `combine_first()` to combine the two `Series`

Test Result: the user will have passed the acceptance test if the resulting `Series` is of the correct integer datatype, and the integer values have also not been affected by type conversion (i.e; if the integer couldn't be precisely represented as a `float64`).

Example acceptance tests are located in `pandas-copy/acceptance_tests/issue2.py`.

File Changes Made

`pandas/core/series.py`

- Lines 3292 – 3305: Implemented fix.

`pandas/tests/series/methods/test_combine_first.py`

- Lines 116–141: several test cases were added to check the datatype of `Series` objects after `combine_first()` operations.
- The contents of the resulting `Series` object were checked to make sure that (potential) conversion between `float64` and `int64` did not change the resulting values.

`pandas-copy/acceptance_tests/issue2.py`

- File was added to include several acceptance test cases.

Issue 3: [BUG] Unable to compare unordered categories #51543

Member Tasks

- Anika Sultana: Created test cases for the solution to ensure the solution is working with all possible cases and wrote the documentation for the report.
- Hyun Woo (Eddie) Shin: Created test cases for the solution to ensure the solution is working with all possible cases and wrote the documentation for the report.
- Man Hei Ho: Identified the actual issue (the original description on the issue page is not very accurate), found the source of the problem and implemented the fix.

Implementation Documentation

The issue is that when comparing two `Categorical` with unordered `categories` of **mixed types**, an unexpected `TypeError`, `Categoricals can only be compared if 'categories' are the same.`, occurs.

The following is the behavior when comparing two `Categorical` with unordered `categories` of the **same type**, which is the expected behavior.

```
In [1]: cat_1 = pd.Categorical([1], categories=[1,2,3], ordered=False)
In [2]: cat_2 = pd.Categorical([1], categories=[1,3,2], ordered=False)
In [3]: cat_1 == cat_2
Out [4]: array([ True])
```

The following is the behavior when comparing two `Categorical` with unordered `categories` of **mixed type**, which raises an unexpected `TypeError`.

```
In [1]: cat_1 = pd.Categorical([1], categories=[1,2,3,'b'], ordered=False)
In [2]: cat_2 = pd.Categorical([1], categories=[1,3,2,'b'], ordered=False)
In [3]: cat_1 == cat_2
Out [4]:
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/home/pandas/pandas/core/ops/common.py", line 81, in new_method
    return method(self, other)
  File "/home/pandas/pandas/core/arrays/categorical.py", line 144, in func
    raise TypeError(msg)
TypeError: Categoricals can only be compared if 'categories' are the same.
```

We have located the source to be the `_categories_match_up_to_permutation()` function of `Categorical`, which is used to ensure the `categories` of the two comparing `Categorical` are the same before proceeding with the actual comparison.

The `_categories_match_up_to_permutation` function should returns True only if:

1. `ordered` of the two `Categorical` are the same (ie. both True or both False)
2. if `ordered` is True for both, `categories` of the two `Categorical` are the same and in the same order
3. if `ordered` is False for both, `categories` of the two `Categorical` contain the same element (ie. don't consider ordering)

The `_categories_match_up_to_permutation` function compares the hash of the two corresponding `CategoricalDtypes`. The hash function of `CategoricalDtypes` uses the `_hash_categories` function which did not handle the hashing of unordered `categories` of mixed type properly, resulting in the unexpected TypeError.

The main problem is that the `_hash_categories` function converts `categories` into a tuple in order to hash the `categories` of mixed type. However, tuple ensures ordering, as such, the hash of the two unordered `categories` are different, even though ordering should not be considered for two unordered `categories`.

The solution we implemented converts `categories` into frozenset instead of tuple to hash unordered `categories` of mixed types. Frozenset is hashable and does not consider ordering which satisfies our requirements.

Acceptance Test

Test Objective: to ensure that the user doesn't encounter an unexpected TypeError when comparing two `Categorical` with unordered `categories` of mixed types.

Test Scenario: a user is trying to compare two `Categorical` with unordered categories of mixed types.

Test Steps:

1. The user imports pandas into the file
2. The user creates two `Categorical`; `cat_1` with `categories=[1,2,3,'b']` and `ordered=False` and `cat_2` with `categories=[1,3,2,'b']` and `ordered=False`.
3. The user performs the equality test on `cat_1` and `cat_2`, i.e. `cat_1 == cat_2`.

Test Result: If the user doesn't encounter a `TypeError` when comparing two `Categorical`, it's ensured that our solution of using frozenset to generate hash has worked for comparison and has fixed the issue properly.

More acceptance tests can be found in `pandas-copy/acceptance_tests/issue3.py`.

File Changes Made

`pandas-copy/pandas/core/dtypes/dtypes.py`

- Line 463 - Implemented fix.

`pandas-copy/pandas/tests/arrays/categorical/test_dtypes.py`

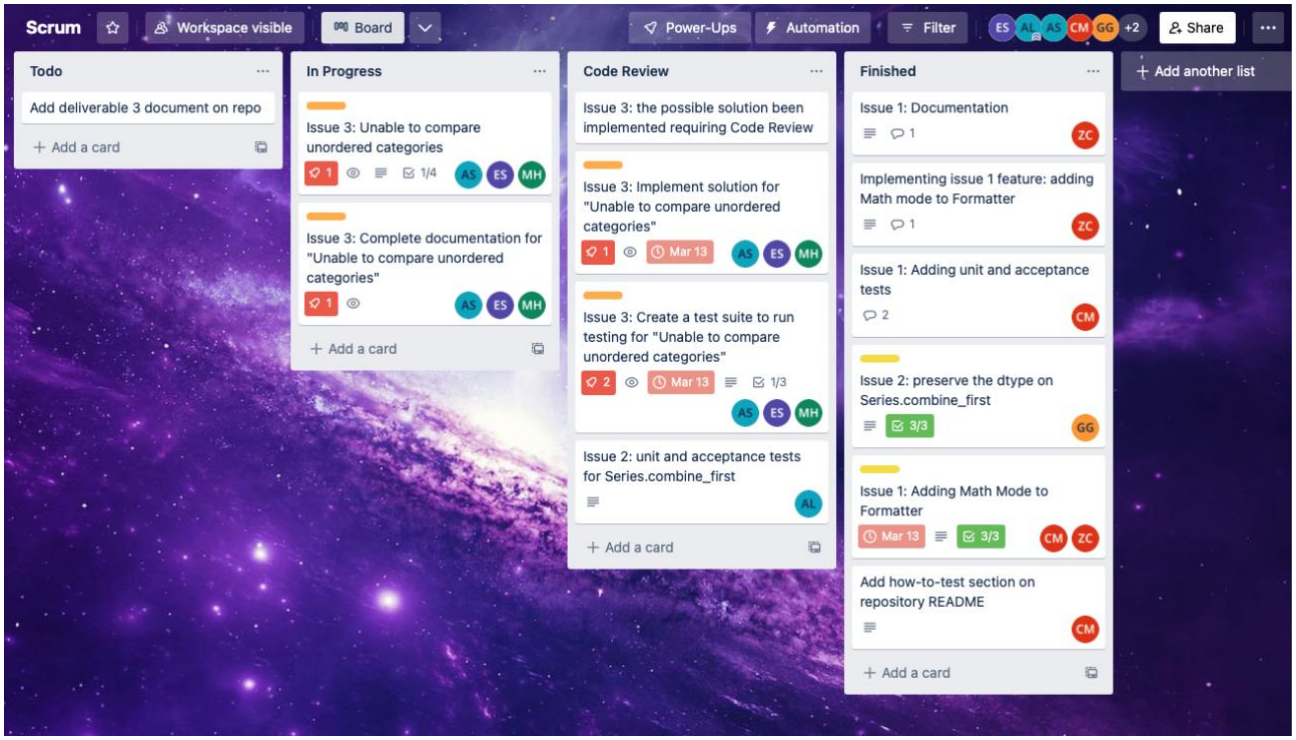
- Line 50-170: added test cases for unit testing which cover possible input combinations of categories based on data types.

`pandas-copy/acceptance_tests/issue2.py`

- File was added to include several acceptance test cases.

Software Development Process: Kanban

Trello Board Link



Trello was used to visually represent the team's progress on their respective issues and subtasks. Each ticket on the Trello board contained a link to the issue on the Pandas repository as well as a short description of what needed to be implemented to resolve the ticket.
You can find the detailed overview of our SCRUM Board here. (You may need to sign up Trello and join the group to view the activities.)

Team Communication

We met every Wednesday on Discord to update the progress on our assigned tasks, plan out next steps, and discuss any challenges that need to be addressed during the meetings. We also had some additional meetings beyond our scheduled Wednesday meetings for weeks that required additional team support due to challenges faced.

All other forms of communications were through our Discord chat server. All members periodically checked the chat and replied to any questions in a timely manner. The chat server was effective in asking broad questions and collaborating to resolve challenges faced. This kept our meetings short as most challenges were resolved before we had our meeting.

Meeting Notes

Meeting #	Date & Time	Duration	Summary
1	Wed, Feb 22 7:30 pm	42 mins	● Went through deliverable 3 document ● Created Trello board to track tasks ● Selected 3 issues for Deliverable 3
2	Sun, Feb 26 7:00 pm	30 mins	● Initialized the Github repo with part of the Pandas codebase ● Had 3 rounds of planning poker to estimate each issue's points. ● Made sure all tasks were assigned between team members
3	Wed, Mar 1 7:30 pm	15 mins	● Updated our status for each issue ● Each team verified that their assigned issue is doable
4	Wed, Mar 8 7:30 pm	17 mins	● Updated our status for each issue. ● Started additional documentation for Deliverable 3
5	Sat, Mar 11 9:30 pm	40 mins	● Updated the progress for each issue. ● Gather ideas about how we lay out our work on our report. ● Discussed some technical challenges working with Git and figured out solutions through team discussion.
6	Mon, Mar 13 6:30 pm	20 mins	● Updated our status on each issue. ● All issues are mostly complete with some tests still in progress. ● Discussed additional write up sections that still need to be completed for the deliverable document. ● Will hold the last meeting on the due date to merge all branches and review the deliverable document before submission.
7	Wed, Mar 15 7:30 pm	30 mins	● Ensured all issues are completed ● Reviewed and submitted document