The Lobbyists - Deliverable #2

Pick 5 Bugs or features

Issue 1: Type hint of DataFrame(...dtype...) excludes numpy scalar types such as np.int8
Link: https://github.com/pandas-dev/pandas/issues/31872
DataFrame, a common object in pandas, throws a warning with a type check when using type np.int8 in its values. Currently, DataFrame can only hold the following types: str, np.dtype, and "ExtensionDtype". These are the python string primitive, a data type object from numpy, and an extension data type object from pandas itself. When numpy integers are thrown in for a DataFrame (constructing a dataframe using pandas.DataFrame(data=d, dtype=np.int8), a type check error is thrown because np.int8 is not part of nor an instance of str, np.dtype, and "ExtensionDtype".

To complete, we would likely go with the suggested solution of adding np.generic (a parent of np.int8) to the accepted types for the pandas DataFrame object. This would take nearly 2 hours to find the location of pandas DataFrame and its accepted types and then adding np.generic, and it would take another hour to test and make sure it does not break anything in the process.

Issue 2: BUG: get_loc / get_indexer with NaT and tz-aware DatetimeIndex
Link: https://github.com/pandas-dev/pandas/issues/32572
One type that pandas uses for tracking datetimes better is NaT, which stands for Not a Time. This is similar to NaN, which is common in all languages as Not a Number, which represents missing or illegal number values. In the same fashion, NaT is put in place for missing datetime values. For the bug, when get_loc (a function for converting a datetime to a local time) is called on a NaT value, the result is 0. Furthermore, when it is used to convert to a different time zone, then it throws an error.

To complete, it would take about 6 hours to investigate both parts of this issue, determining why 0 is returned for local time and figuring out what the type issue is when converting to a different time zone. It would then take another 2 hours to fully test our solution and make sure it does not break anything else in the code.

This was one of the issues we chose to tackle at first. After many hours of debugging, we figured out that the problem is in the logic, *get_loc* inside **Index** class is not properly handling some cases that are of “na” type: None, NaT,.. Adding the fix would be easy, we simply checked for those cases using *isna* function. However, it seemed like

Issue 3: dataframe.groupby incorrect with multiindex and None value
Link: https://github.com/pandas-dev/pandas/issues/32492

This bug concerns pandas' groupby function. The group by function is defined as it would be in any other data management system; it takes input column names and then aggregates data across those columns. When you groupby on a DataFrame in pandas and do not specify all the columns, then you will lose some data (as expected). However, when you try to call get_group on a list of items on the group by result that should not exist, it does not return a KeyError as expected.

To complete, it would take about 7 hours to investigate both the groupby and get_group function logic and determine the best way to fix it. It would also take another 2 hours to test our solution against our own requirements and the rest of the code.

Issue 4: Rows order when using slice(None) on MultiIndex Dataframe.loc
Link: https://github.com/pandas-dev/pandas/issues/31330
This issue is based around a cosmetic bug when printing a DataFrame. When printing a DataFrame with several column hierarchies (ex. Cats: Black, Brown, …, Dogs: Black, Brown, …), pandas automatically sorts them using the order that they were inputted when the DataFrame was created. This makes sense, but when the DataFrame is printed with a different hierarchy specified (Brown -> Black instead of Black -> Brown), the result DataFrame does not take the hierarchy specified into account, and just prints what it would originally.

To complete, it would take about 5 hours to investigate the organization of DataFrame data and the print functions available to it and determine the best way to print the cosmetic result. It would take another 1 hour to check for edge cases of our fix and test against the rest of the codebase.

Issue 5: Inconsistency/bug when selecting from a data-frame using an unsorted DatetimeIndex
Link: https://github.com/pandas-dev/pandas/issues/30736
This issue involves inconsistent indexing in pandas. In pandas, you can create an index of a range of values that you want, like between the min and max of your data or between two datetimes. The example on the issue page shows how sometimes, when creating a DataFrame out of a disordered index, inconsistent and buggy results occur. Specifically, if you just want to add an hour value to the end of your range when you are selecting from your data, a KeyError occurs with an uninformative message.

To complete, it would take about 6 hours to investigate the cause of the KeyError bug and determine a clean, scalable way to fix the way indexing works in this case. It would take another 2 hours to test our fix against our requirements and the rest of pandas' test suite.

Issue 6: [BUG] Dataframe.rank() produces wrong results for float columns
Link: https://github.com/pandas-dev/pandas/issues/32593
This bug concerns the Index's rank() function for sorting columns in Dataframes. The specific issue is when infinite values are entered: For some reason, numpys np.inf is not always selected as the largest number in the ranking. This would raise problems in mathematical analyses which one of pandas' main uses, so this issue would be very beneficial to tackle.

To complete, it would take about 6 hours to investigate the reason why rank would mess up with some float columns and specifically np.int, and find a reasonable way to fix the ranking. It would take another 2 hours to test our fix against our requirements and the rest of pandas' test suite.

Our Chosen Issues

We have selected Issue 3 and Issue 6 out of the six explained above.

Issue 3: dataframe.groupby incorrect with multiindex and None value
For a summary of the issue and an estimated time, refer to the previous section at Issue 3.

We chose this issue because it seemed of medium to significant difficulty, and would give us a greater understanding of the codebase (finding out about different types and how pandas deals with it, as well as looking at functions between objects as in groupby). It also looks like an issue that would clear a lot of confusion for users of pandas, which is one of our team goals (increasing the usability of the project).

Potential Risks:
While groupby does seem to have a specific use outside of pandas internal files, it may be used directly from certain functions inside pandas for efficiency. This could raise a problem if these other functions are expecting to get results from group by in its current state, even if this current state is incorrect. To accommodate for this we will be running our solution against the entire test suite after our fix and make sure to go through each potential failure to ensure we are not breaking anything else in our change.

Issue 6: [BUG] Dataframe.rank() produces wrong results for float columns
For a summary of the issue and an estimated time, refer to the previous section at Issue 6.

We chose this issue because it also seemed of medium to significant difficulty, and would give us a better understanding on the deeper logic of pandas. Like we mentioned in the summary, this could go deep into pandas' code as it may have to do with dealing with core comparison logic. It also seems like something that would fix a lot of other issues if done correctly, which would be a great use of our time.

Potential Risks:
Thinking through this issue, this seems like this is likely a type ordering issue which fixing could have other ramifications. Specifically, depending on how the code is organized, there may be a function that we need to change to accommodate for our specific example's fix, and this function could be used for other ordering algorithms that depend on the function in different ways. We will again make sure to run our solution against the entire test suite after our fix and make sure we are not breaking anything else in our change.

Software Process Proof

As we stated in Deliverable 1, we used Kanban as our software process of choice when fixing the selected issues. This involved setting up the Kanban board, opening necessary communication channels, and making sure every task was completed by the due date. Here we will go over how we followed Kanban to complete this deliverable.

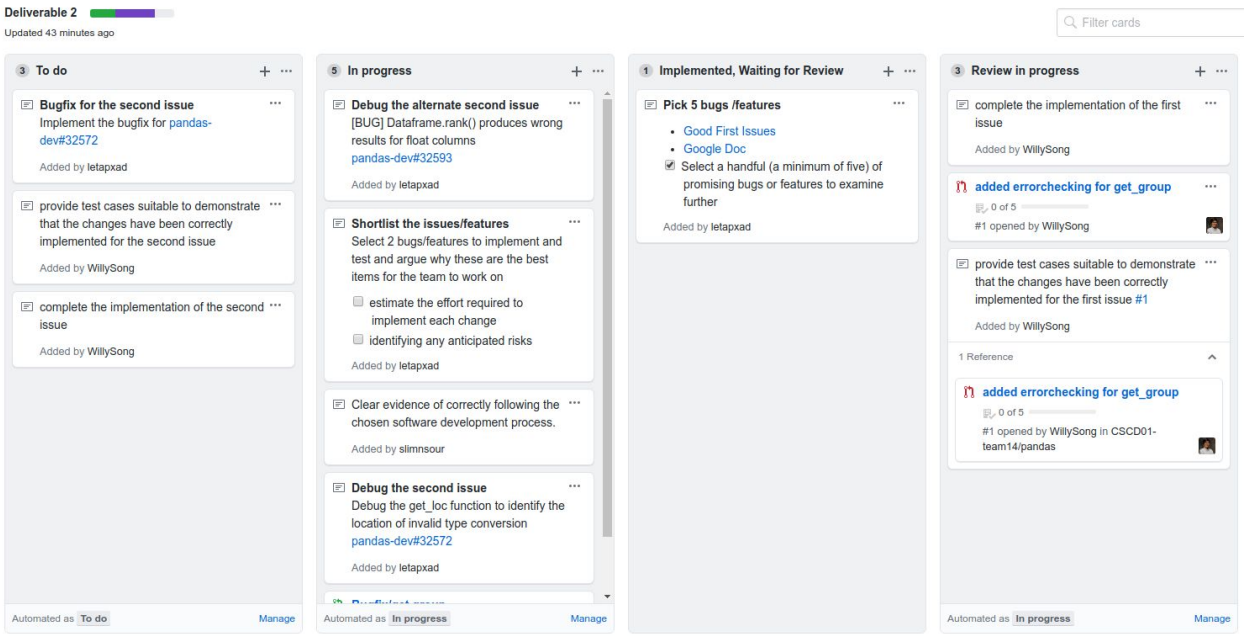
Kanban Board

Our Kanban board for this deliverable can be found here:
https://github.com/CSCD01-team14/pandas/projects/1

As planned, we used Github's Projects board as our Kanban board. We first set up our board with the proper columns to track the lifetime of an issue: To do; In progress, Implemented; Review in Progress, Reviewer approved, waiting for merge; Merging; Done.

We then split up our assignment into tasks that would be put together to complete the work needed for the deliverable (ex. Choosing five issues to develop, and Shortlisting it to 2 issues). For each task, as per Kanban guidelines, we made sure to write a small description of the task, how long it would take, link the necessary resources, and its deadline. Given that we filled out each task with a corresponding description, and knew the estimated time, we were sure that each of the tasks would be completed (and therefore the deliverable) by the due date.

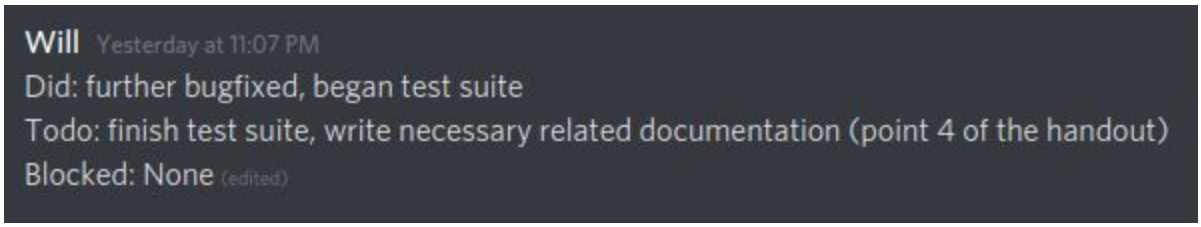
Here is a snapshot of our Kanban board:



Daily Asynchronous Standup Meetings

Another part of the software process that we agreed on is doing virtual standup meetings to update the other group members on progress. We did this in a dedicated channel in our method of communication Discord called “daily-standups”. Every day near the end of the project, we would send messages like the one shown below to update other team members on what we are currently doing for the project.

Here is an example report from a virtual standup meeting:



As you can see, each message would consist of a “Do” part for what was done that day, “Todo” for what they have left to do in the future, and “Blocked” for what they need to do but cannot because of some dependency they are waiting on (another team member’s task, external dependency like a pull request, etc.) This worked great for us because we were able to stay updated with each other while not compromising our busy schedules as students. Furthermore, it allowed for follow up questions and conversation, which is not as easily applicable through just the Kanban board.

Why Kanban Came in Handy

Using Kanban helped us in the project because we were allowed flexibility in our development. For any task in the Kanban board, we were allowed to update it or add new tasks as our understanding of the issues increased over the course of the deliverable. Since Kanban emphasizes continuous updates, we had that flexibility to reroute at any time to ensure the deliverable was completed. For example, when we wanted to investigate other issues to try and implement, we were able to record what we were currently working on and place it in the “To do” column, as well as add new tasks relevant to the new issue and place it in the “In progress” column.

It was also extremely helpful since we were able to accomodate for our busy schedules as students by having a visual guide to completing the work for the deliverable. Every day we could check at any time what tasks needed to be done and how long it would take for them so each team member could fit it into their schedule correctly (among their other assignments). This way we were able to complete all tasks asynchronously, while also staying active on the Discord for synchronous communication.

Technical Commentary

Issue 3: dataframe.groupby incorrect with multiindex and None value
Link: https://github.com/pandas-dev/pandas/issues/32492

How these changes affect the design:
The bugfix implemented for the *get_group* function does not affect the overall structure of the project. This bugfix is adding a verifier to the output of a helper function (*_get_index*) within *get_group*. The helper function sometimes gives an unexpected output but is used in many more functions than *get_group* so we decided to add a verifier within *get_group* to preserve as much of the codebases' integrity as possible. Thus, *"pandas/core/groupby/groupby.py"* was the only file that was changed as it is the file with *get_group*.

The only issue that arises is that many other functions call upon the *get_group* function. When running the extensive *pytest pandas* test suite, we will see a few (12) test cases fail now as they are given a different error. Previously, they were using invalid input and expected a different error to appear through the use of a different function. These test cases will have to be revised and updated, but that is beyond the scope of this bugfix.

The pull request for this issue can be found here:
https://github.com/CSCD01-team14/pandas/pull/2
Test suite can be found here:
https://github.com/CSCD01-team14/pandas/pull/2/files#diff-c53c0d90d98374520ad40b3808903fd1

Issue 6: [BUG] Dataframe.rank() produces wrong results for float columns
Link: https://github.com/pandas-dev/pandas/issues/32593

How these changes affect the design:
The bugfix does not affect the structure of the project. It just slightly changes the behavior of rank of DataFrame when DataFrame consists of floats. The only thing that changed is *rank_2d* function inside *algos.pyx* file in *pandas/_libs* folder.

More details about the bug and *rank_2d* function:
Rank function is supposed to assign ranks for numbers inside the DataFrame (either ascending or descending). Suppose we have a data frame of 1 column: [9, 2, 3]. Since 2 is lowest, it gets rank 1; 3 gets rank 2; and 9 gets rank 3. We're using all valid numbers here, but the column can also contain np.nan (not a number), np.inf (infinity), -np.inf (minus infinity). When the data type of the dataframe is **int**, they should all be the same because infinity is not a valid number for int. However, inf and -inf are valid numbers for float, in particular, inf would be the largest number,

and - inf is the lowest. So you would expect df.rank() to rank them like that, however that is not the case. If you are ranking ascending, then inf is treated as nan, and if you are ranking descending, -inf is treated as nan (because for type(int) it is the same). In particular, they assigned nan_value = np.inf (for ascending case) and nan_value = -np.inf for float case. We changed this to nan_value = np.NaN instead.

The pull request for this issue can be found here:
https://github.com/CSCD01-team14/pandas/pull/3/
Test suite can be found here:
https://github.com/CSCD01-team14/pandas/pull/3/files#diff-ef1ed2b4659b2621060565a4db36ef3c