



**University of Toronto**

## **CSCD01H3 Engineering Large Software System**

### **Deliverable #3: Pandas Open Source Contribution**

#### **Final Version**

**Winter 2023**

**Instructor: Prof. Rawad Abou Assi**

*Group Name: Timbits*

*Group Members:*

*Xuen Shen*

*Xu Zheng*

*Lingfeng Su*

*Yawen Zhang*

*Megan Mujia Liu*

*Shaopeng Lin*

*Runyu Yue*

Feb.19, 2023

<b>Issues</b>	<b>2</b>
Issue 1: Inconsistent behavior when DataFrame with strings and None is created from lists or dictionary. #32218	2
Member contribution	2
Implementation description and reasoning	2
Issue Description	2
Implementation and reasoning	2
Detailed Changes	3
Acceptance tests/description	4
Discussion timeline	4
Issue 2: JSON serialization with orient split fails roundtrip MultiIndex. #50456	5
Member contribution	5
Implementation description and reasoning	5
Issue Description	5
Implementation	6
Detailed Changes	7
Unit tests and Acceptance tests/description	7
Discussion timeline	7
Issue 3: Series.take needs to validate axis. #51022	8
Member contribution	8
Implementation description and reasoning	9
Issue Description	9
Implementation and reasoning	9
Detailed Changes	9
Unit tests and Acceptance tests/description	10
Discussion timeline	10
<b>Group Development Processes</b>	<b>11</b>
Group-wise Meeting 1 (March 2, 2023 : 2 hours)	11
Group-wise Meeting 2 (March 9, 2023 : 1 hour)	11
Task Time estimation and Burndown Result	11
Burndown Chart	11
Time Estimation	12
Tracking/Assignment artifacts	13
Dependency Graph	13
Task assignments	14

# Issues

## Issue 1: [Inconsistent behavior when DataFrame with strings and None is created from lists or dictionary. #32218](#)

### Member contribution

#### Yawen Zhang:

Design and develop unit and acceptance test cases. See artifacts for detailed task distribution

#### Shaopeng Lin:

Analyze the root of the problem and develop implementation to resolve the issue. See artifacts for detailed task distribution.

### Implementation description and reasoning

#### Issue Description

The issue is an inconsistency between different initialization methods of DataFrame. Pandas takes a nullable approach to its values, but for string or object type it is discovered that the null value **None** is casted to “None” when done in a simple list initialization:

```
1 >>> import pandas as pd
2
3 >>> df = pd.DataFrame(["1", "2", None], columns=["a"], dtype="str")
4 >>> type(df.loc[2].values[0])
5 <class 'str'>
6 >>> df = pd.DataFrame({"a": ["1", "2", None]}, dtype="str")
7 >>> type(df.loc[2].values[0])
8 <class 'NoneType'>
```

#### Implementation and reasoning

The root of the problem stems from that DataFrame initialization is using numpy's `astype()` function to cast everything in a list to the same type.

DataFrame constructor calls *init\_ndarray* defined in *pandas\core\internals\construction.py* to apply type conversion to input datas who are list-like. Due to the nature of numpy, we cannot cast elements one by one and everything will have to coerce to the same type. We investigated how Series casted their input and found that there already exist a work around to this problem by calling *sanitize\_array* defined in *pandas\core\construction.py*.

```

1  >>> import pandas as pd
2
3  >>> sr = pd.Series([1, 2, None], dtype="str")
4  >>> type(sr.loc[2].values[0])
5  <class 'NoneType'>

```

Though **sanitize\_array** only works on 1-D input, we can use convert DataFrame's potentially multidimensional input data to 1-D array using numpy's shape functionalities. After sanitization, we can convert it back to its original shape.

This implementation addresses the issue as, by using the same technique when constructing Series, we will have a consistent construction of values across user APIs. In the detail implementation, we further more ensures that we only cast using this strategy when facing string/object types as numeric types are handled by Pandas already. If we do this casting on numeric types, the resulting type is often object which disallow Pandas to recognize nullable numeric values.

## Detailed Changes

The problem exist in Pandas version 1.1.x, which is checkout in the **pandas-issue-32218** folder.

Modified files: **pandas\core\internals\construction.py**,

**pandas/tests/frame/test\_constructors.py**, **pandas/tests/internals/test\_internals.py**

```

193     if dtype is not None and not is_dtype_equal(values.dtype, dtype):
194
195         # Similar to Series, use sanitize_array to precess array
196         # Need to convert values into a single array before hand
197         try:
198             # Due to nullable number types, we will not modify their behaviour
199             if (is_numeric_dtype(dtype)):
200                 values = values.astype(dtype)
201             else:
202                 values_shape = values.shape
203                 oneD_array: np.ndarray = sanitize_array(values.flatten(), index, dtype, copy,
204                                                         raise_cast_failure=True)
205                 values = np.reshape(oneD_array, values_shape)
206
207         except Exception as orig:
208             # e.g. ValueError when trying to cast object dtype to float64
209             raise ValueError(
210                 f"failed to cast to '{dtype}' (Exception was: {orig})"
211             ) from orig

```

This code segment originally only casts casts "values" using astype() and catches the any casting exceptions. We now only do so when dtype is a numeric type. For any none numeric types, we will store the shape of the input and use the flattened version on **sanitize\_array** as mentioned before. Using the previous shape, we can convert it back to the expected ndarray.

New test cases were added to **pandas/tests/internals/test\_internals.py**, of which you can find in class **TestInitNdArray**. We only need to test our new behaviours as the previous developers

were responsible to write the unit test for the expected behaviour of ***init\_ndarray***, which we should pass.

New test cases were added to ***pandas/tests/frame/test\_constructors.py***, of which you can find in ***TestDataFrameConstructorsOnNoneValue***. They are testing the DataFrame constructor API's behaviour for the different types of initialization that the constructor will call ***init\_ndarray*** on.

## Acceptance tests/description

According to the changes made on function ***init\_ndarray***, acceptance tests are added to test ***DataFrame*** constructor to make sure that constructors that trigger ***init\_ndarray*** works properly.

Acceptance tests and guides are located at:

***pandas/tests/Issue32218\_AcceptanceTests***

The details are in their respective tests.

## Discussion timeline

### 2023.3.6 (1 hour):

#### Summary:

Looked over issues 31104, 32218, 50885 to decide for a issue. 31104 is very nuanced with the type structure of Pandas which can take a long time while 50885 is a web feature that deviates direct contribution to the core functionalities of Panas. We decided to proceed with 32218 as it is a lighter fix than 31104 considering the time constraint and leaves 50885 as the last resort.

Discovered source of the issue, which is ***init\_ndarray*** in ***pandas\core\internals\construction.py***. This function is only called by Dataframe so we can isolate the implementation and test.

There are a few more construction other than the one identified by the issue:

1. Np.masked\_array
2. Np.ndarray, Series, Index
3. Undefined type scalar used by Index to declare a Dataframe (To be discussed as it involves changing the constructor directly)

We initially propose to solve this by casting values in a numpy array one by one.

### 2023.3.8 (45 minutes):

#### Summary:

After the first attempt, we noticed that numpy array cannot be casted separately. By proxy looking at Series, we discovered ***sanitize\_array*** which gives us a new way to resolve this issue.

### 2023.3.9 (30 minutes):

#### Summary:

With the new fix, we resolved the issue but is failing some unit tests in Pandas. We discovered that we need to have the same behaviour as before with regard to numeric value and error handling. We also discovered that we do not need to worry about the third construction type discussed in 2023.3.6, as it is impossible to initialize a list of None as a scalar. It will be treated as a dictionary instead of a list.

## Issue 2: [JSON serialization with orient split fails roundtrip MultiIndex. #50456](#)

### Member contribution

#### Lingfeng Su:

Analyze the root of the problem and develop implementation to resolve the issue.

#### Megan Mujia Liu:

Analyze the root of the problem and fix variable bugs in the implementation.  
Develop Acceptance Tests.

#### Runyu Yue:

Analyze the root of the problem and brainstorm with implementation.  
Develop Unit Tests.

### Implementation description and reasoning

#### Group Results

All work from Group 2 is inside [d01w23-team-timbits/deliverable3/pandas-issue-50456](#), we are working on an older version of pandas since the issue is already closed by others on git.

#### Issue Description

The problem exists due to a conflict on data structure between class MultiIndex and methods **to\_json()**, **read\_json()**.

For any columns in the JSON file given to **read\_json()**, it takes the list and matches the indices to the corresponding index in DataFrame. For example, from Figure1, the first element in the list (“col 1”) will be the column name for the first column.

```
>>> df.to_json(orient='split')
'{"columns":["col 1","col 2"],"index":["row 1","row 2"],"data":[["a","b"],
>>> pd.read_json(_, orient='split')
   col 1 col 2
row 1    a    b
row 2    c    d
```

Figure1

The problem occurs when the user wants to have multiple column names. For example, from Figure 2, the user inputs two names for each column. MultiIndex takes the array and implements it into [("2022", "JAN"), ("2022", "FEB")]. This means that ("2022", "JAN") is the name for the first column, as shown in Figure2. **to\_json()** takes this implementation and puts it into a string, as shown in Figure3, the difference is that the tuple is converted to a list.

```
>>> df = DataFrame([[1, 2], [3, 4]],
...                  columns=pd.MultiIndex.from_arrays([["2022", "2022"], ["JAN", "FEB"]]))
>>> df
      2022
      JAN FEB
0        1  2
1        3  4
```

Figure2

```
In [2]: df.to_json(orient='split')
```

```
Out[2]: '{"columns": [{"2022", "JAN"}, {"2022", "FEB"}], "index": [0, 1], "data": [[1, 2], [3, 4]]}'
```

Figure3

The problem stems from the fact that **read\_json()** and **to\_json()** creates a different data structure compared to how the multi-index data structure is implemented. Behind the scenes, multi-index is actually a list of tuples. For example, our example above results in a multi-index value of [(2022, JAN), (2022, FEB)] when we are provided with columns [["2022", "2022"], ["JAN", "FEB"]]. However, **to\_json()** then performs a change to our data by directly converting it from a list of tuples to a list of lists, resulting in [[2022, JAN], [2022, FEB]], as you can see, **to\_json()** neglected to reverse the transpose performed when we initially created the multi-index value.

Now when we try to perform a **read\_json()** operation, it calls the dataframe constructor again to reconstruct our dataframe from a JSON input. Resulting in a transpose due to a wrong input of [[2022, JAN], [2022, FEB]] when the correct input should be [[2022, 2022], [JAN, FEB]].

## Implementation

To solve the issue, we chose to change the implementation of the function **to\_json()** method in `/pandas/io/json/_json.py`.

In the beginning, when we read through the code and testing on the issue, the **to\_json()** result was confusing to our group. If we don't look into the MultiIndex, it seems like **to\_json()** swaps the input given by the user, and it creates some "transposing" when column names are more than one. After we looked into MultiIndex and the actual columns of the DataFrame, it is the MultiIndex and tuple creating confusion to the **to\_json()**.

So, we decided to create a new DataFrame with only the columns different from the original inside of **to\_json()**. In simpler terms, we transposed the input to **to\_json()** when the scenario of the issue is met. Resulting in **to\_json()** outputting a JSON of the same structure as the user's input. This decision reduces possible confusion and gives a reasonable result to the user.

## Detailed Changes

Function **to\_json()** uses `FrameWriter` when input is type `DataFrame`. The `DataFrame` object is passed into `FrameWriter`, and the `DataFrame` object is then written into a JSON string. We created a copy of the object and rearranged the columns field. In the property function of the class `FrameWriter`, we added an `elif` to differentiate the problematic input. The input object needs to satisfy two conditions: `orient == "split"` and `isinstance(self.obj.columns, MultiIndex)`. Then we copied the original object into **new\_df** with everything the same. Using list comprehension on the original `object.columns`, we constructed a list of column names instead of the original `MultiIndex`. These new columns are stored in the variable **new\_column**. Last but not least, we made this variable the columns of **new\_df**. Initialize this `new_df` to the `FrameWriter`'s `object_to_write`. The JSON file will have our expected result.

Modified files:

`pandas/io/json/_json.py`

Testing files added:

`pandas/tests/io/json/test_json_writer.py`

`pandas/tests/Issue50456_AcceptanceTests`

## Unit tests and Acceptance tests/description

Unit test is written using a separate file in the directory `pandas/tests/io/json/test_json_writer.py`. It tests different scenarios in the behavior of `FrameWriter.obj_to_write` from an empty `DataFrame` to a 4\*4 `DataFrame` by testing whether the generated `obj_to_write` field is the transpose of the original one.

Acceptance tests are written for users in `pandas/test/Issue50456_AcceptanceTests`. Instructions, test files, and the expected output are inside this folder. Users can follow the instructions and test for desired behavior.

## Discussion timeline

### 2023.3.3 (2 hours):

#### Summary:

Focused on the virtual environment setup, and looked into the issue 50456. We used Jupyter to run several tests to understand which specific location caused the problem. By plugging in different types of input, we identified two issues in this 1.5.3 version.

Original issue #50456: Whenever `read_json(json string, orient="split")` is called on a json string with multiple layers of the column name, the result of the column name is turning into a transpose of itself which is not expected. (check `io/json/_json.py/line1307`)

Encountered issue: There was an assertion error in `read_json` that caused (number of rows of column names != number of rows of data) to fail an assertion error when parsing. `DataFrame`



allows users to create variant input, but `to_json` cannot accept any DataFrame object that is not square size.

We collectively read over and discussed the code in `/io/json/_json.py`. And identified the two problems illustrated above, the encountered issue is not found in the GitHub issue tracker.

Therefore, we decided to focus on #50456 for now and look back to the encountered one maybe in Deliverable 4.

### **2023.3.9 (2 hours):**

#### **Summary:**

We read through the `_json.py` file together again and discussed some possible implementations to fix the bug. We encountered four possible implementations: fix the Writer initialization in `to_json`, fix the DataFrame object in Writer class, fix the JSON string output of `to_json`, or fix `read_json`. After looking into `read_json`, it involved many other attributes of the DataFrame such that it used those attributes as conditions. Those fields are not related with the issue input. So, we decided to not touch it to prevent change deeply inside any fields that are not related. Same reason for fixing Writer in `to_json`, the Writer class has many attributes and we don't exactly know how they were initialized. Therefore, we choose to directly fix the DataFrame object inside the `FrameWriter` class.

### **2023.3.10 (40 minutes):**

#### **Summary:**

After all the implementations, we all tested on our own and discussed any confusion in the major implementation. We talked about how unit tests and acceptance tests should be generated and put in which file.

### **2023.3.11 (1 hour):**

#### **Summary:**

After adding all tests, we made sure that all of the tests passed on each of our machines. We removed any print statements and unnecessary comments that were left over from when we were identifying the problem, implementing our fixes, and testing our code. We also created, reviewed, and merged our pull request. Finally, we checked our document by revising our document and fixing grammar mistakes.

## **Issue 3: [Series.take needs to validate axis. #51022](#)**

### Member contribution

#### **Xuen Shen:**

- Analyze the root cause of the problem

- Develop test cases idea for acceptance test and unit test

### Xu Zheng:

- Analyze the root cause of the problem
- Develop test cases idea for acceptance test and unit test
- Develop implementation to resolve the issue
- Develop implementation to unit test and acceptance

## Implementation description and reasoning

### Issue Description

The issue is caused by the absence of the validate parameter, "axis". There is a default value of **0** for axis in Series.take(). And Series.take() ignores this parameter because it is not used by Series. In spite of the fact that the axis is specified, Series.take() always assumes value the axis is **0**.

```
>>> import pandas
>>> pandas.Series([1,2,3]).take([1], axis="foo")
1      2
dtype: int64
>>> █
```

### Implementation and reasoning

We add a validation on axis value, if the value is not **0** or '**index**', we raise value error, since **0** or '**index**' are both considered eligible axis values in Series according to the [Series.take documentation](#).

```
>>> import pandas
>>> pandas.Series([1,2,3]).take([1], axis="foo")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/workspaces/d01w23-team-timbits/deliverable3/pandas-issue-51022/pandas/core/series.py", line 917, in take
    raise ValueError(f"axis={axis} is not a valid parameter")
ValueError: axis=foo is not a valid parameter
>>> █
```

## Detailed Changes

In order to resolve the issue, we need to validate the parameter axis in the function Series.take().

```

913 913
914 914         indices = ensure_platform_int(indices)
915 915
916 +         if axis not in [0, "index"]:
917 +             raise ValueError(f"axis={axis} is not a valid parameter")
918 +
916 919         if (
917 920             indices.ndim == 1
918 921             and using_copy_on_write()

```

Modified files:

***pandas-issue-51022\pandas\core\series.py,***

## Unit tests and Acceptance tests/description

Unit tests is located at

***pandas-issue-51022\pandas\tests\series\indexing\test\_take.py***

Acceptance tests and guide are located at

***pandas-issue-51022\pandas\tests\series\acceptance\_test***

## Discussion timeline

### 2023.3.4 (30 minutes):

#### Summary:

We picked and looked into the issue 51022. We found out ***{0 or 'index', 1 or 'columns', None}*** are the only acceptable value for axis, and axis value is unused and default is ***0*** in Series.take() through the documentation of the function. We propose to solve this by validate axis value is ***0*** or ***'index'*** in the function, since ***1***, ***'columns'*** and ***None*** are nonsense in Series.take().

### 2023.3.9 (30 minutes):

#### Summary:

We discussed about the test cases for Series.take(), we categorize test cases in three scenarios:

valid axis values in Series ***0*** or ***'index'***, invalid axis value in Series, but valid axis value ***1***, ***'columns'*** and ***None***, and invalid axis values.

# Group Development Processes

## Trello Invitation Link:

<https://trello.com/invite/b/Uq5KhMEH/ATTIf6e2d135724a0bccadc2f567583a0340CB02AB10/scrum-board>

## Group-wise Meeting 1 (March 2, 2023 : 2 hours)

### Summary:

Using pandas development documentation to collectively setup the development environment. Everyone understands how to run and build Pandas on their own machines for development. Made the decision to split into three separate self-autonomous groups to solve the three required issues. Setup Trello to do task tracking documentation and committed a built version to Github for groups to interpret. The rest of the meeting is spent to filter “good-first-issues” that are substantial enough for each group’s fix.

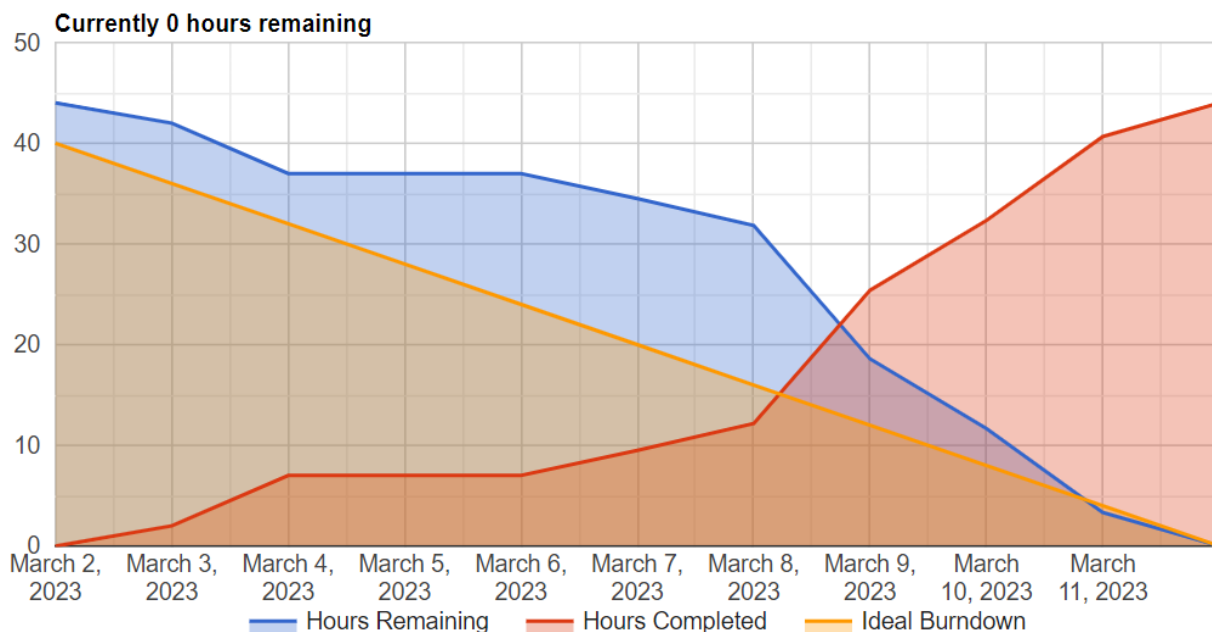
## Group-wise Meeting 2 (March 9, 2023 : 1 hour)

### Summary:

Check-in meeting where the autonomous groups share their progress and synchronizes the task tracking and github processes. Estimation on task time is done. Discussion takes place on how to proceed with testing such as unit tests and we decided to follow the file structure of Pandas in the tests folder. Groups then try to resolve any blockers and questions during development.

## Task Time estimation and Burndown Result

### Burndown Chart



Our estimated total hours is 44 hours and we were able to finish everything in 40 hours 40 minutes. Looking at the artifacts, our estimations are mostly correct with a few overestimated tasks. This signals that we can use the similar tactics to estimate time for each task. Our preparation to analyze tasks took a while which means our development tasks are concentrated in the later stages.

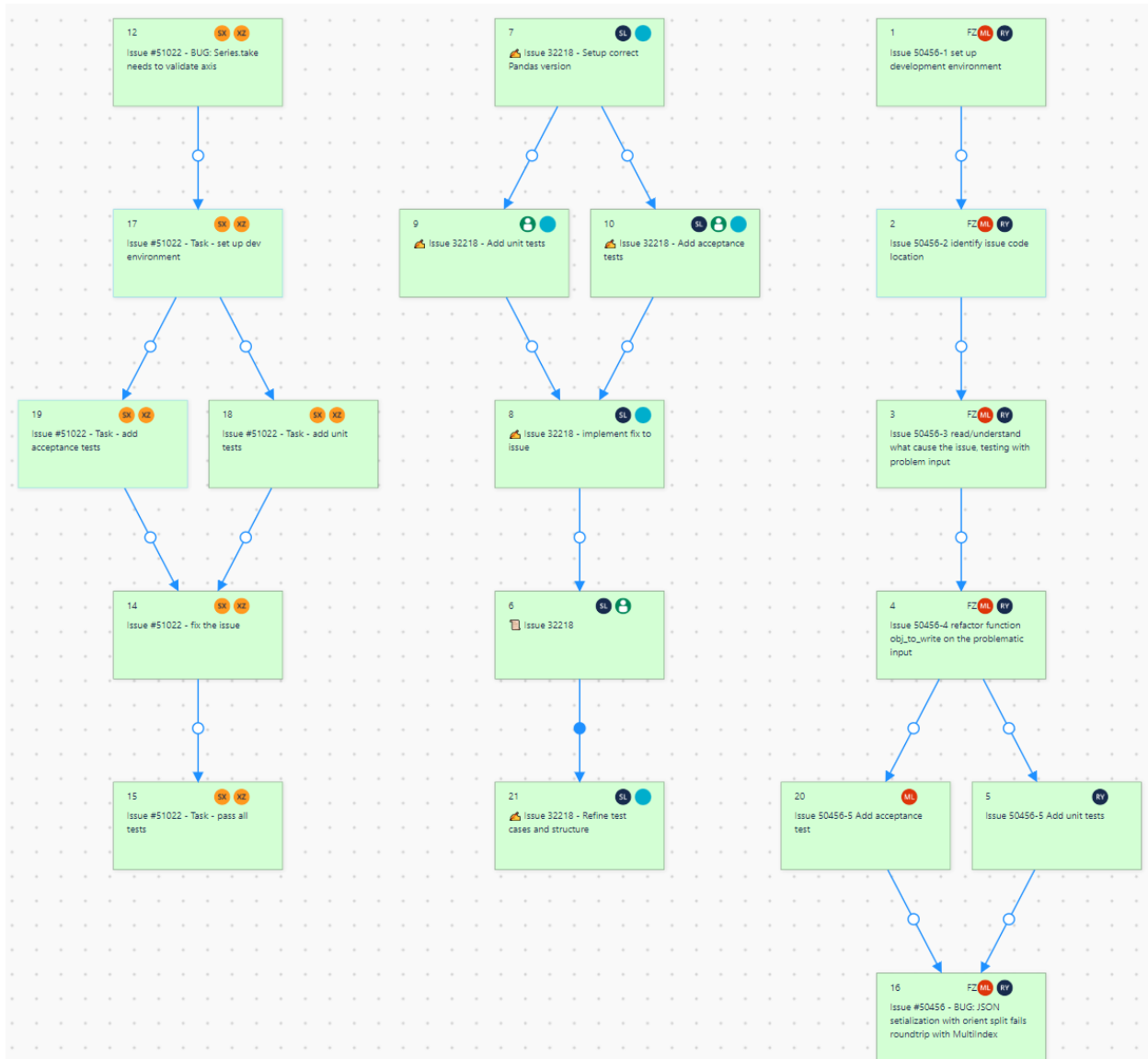
## Time Estimation

Issue 50456-5 Add unit tests	2h	2h
Issue 50456-5 Add acceptance test	1h 40m	2h
Issue 50456-4 refactor function obj_to_write on the problematic input	3h 20m	3h
Issue 50456-3 read/understand what cause the issue, testing with problem input	3h	3h
Issue 50456-2 identify issue code location	7h 30m	8h
Issue 50456-1 set up development environment	4h	4h
Issue #51022 - Task - set up dev environment	1h 31m	2h
Issue #51022 - Task - pass all tests	15m	30m
Issue #51022 - Task - add unit tests	21m	30m
Issue #51022 - Task - add acceptance tests	21m	30m
Issue #51022 - fix the issue	39m	1h
Issue #51022 - BUG: Series.take needs to validate axis	4h	4h
Issue #50456 - BUG: JSON setialization with orient split fails roundtrip with MultiIndex	4h 50m	5h
🔥 Issue 32218 - Setup correct Pandas version	30m	30m
🔥 Issue 32218 - Refine test cases and structure	2h	2h
🔥 Issue 32218 - implement fix to issue	1h 52m	2h
🔥 Issue 32218 - Add unit tests	51m	2h
🔥 Issue 32218 - Add acceptance tests	2h	2h

# Tracking/Assignment artifacts

We develop in the priority of the dependency relation and task assignment below.

## Dependency Graph



## Task assignments

Xuen Shen:

- |   |  |
|---|--|
|  Shen Xuen | Issue #51022 - Task - set up dev environment |
|  Shen Xuen | Issue #51022 - Task - add acceptance tests   |
|  Shen Xuen | Issue #51022 - Task - add unit tests         |
|  Shen Xuen | Issue #51022 - fix the issue                 |

Xu Zheng:

- |  |  |
|--|--|
|  xu zheng   | Issue #51022 - BUG: Series.take needs to validate axis |
|  xu zheng | Issue #51022 - Task - add unit tests                   |
|  xu zheng | Issue #51022 - Task - pass all tests                   |
|  xu zheng | Issue #51022 - Task - add acceptance tests             |
|  xu zheng | Issue #51022 - Task - set up dev environment           |
|  xu zheng | Issue #51022 - Task - set up dev environment           |
|  xu zheng | Issue #51022 - fix the issue                           |

Lingfeng Su:

 flying zambie

Issue 50456-2 identify issue code location

 flying zambie

Issue #50456 - BUG: JSON setialization with orient split fails roundtrip with MultiIndex

 flying zambie

Issue 50456-4 refactor function obj\_to\_write on the problematic input

 flying zambie

Issue 50456-3 read/understand what cause the issue, testing with problem input

 flying zambie

Issue 50456-1 set up development environment

Yawen Zhang:

 嗡嗡嗡

 Issue 32218 - Add unit tests

 嗡嗡嗡

 Issue 32218 - Add acceptance tests

Megan Mujia Liu:

 ML Megan Liu

Issue 50456-5 Add acceptance test

 ML Megan Liu

Issue #50456 - BUG: JSON setialization with orient split fails roundtrip with MultiIndex

 ML Megan Liu

Issue 50456-5 Add acceptance test

 ML Megan Liu

Issue 50456-1 set up development environment

 ML Megan Liu

Issue 50456-4 refactor function obj\_to\_write on the problematic input

 ML Megan Liu

Issue 50456-3 read/understand what cause the issue, testing with problem input

 ML Megan Liu

Issue 50456-3 read/understand what cause the issue, testing with problem input


 ML Megan Liu

Issue 50456-2 identify issue code location



Shaopeng Lin:

 Shaopeng Lin

 [Issue 32218 - Refine test cases and structure](#)

 Shaopeng Lin

 [Issue 32218 - implement fix to issue](#)

 Shaopeng Lin

 [Issue 32218 - implement fix to issue](#)

 Shaopeng Lin

 [Issue 32218 - implement fix to issue](#)

 Shaopeng Lin

 [Issue 32218 - Setup correct Pandas version](#)

Runyu Yue:

 Runyu Yue

[Issue 50456-5 Add unit tests](#)

 Runyu Yue

[Issue 50456-4 refactor function obj\\_to\\_write on the problematic input](#)

 Runyu Yue

[Issue #50456 - BUG: JSON setialization with orient split fails roundtrip with MultiIndex](#)

 Runyu Yue

[Issue 50456-1 set up development environment](#)

 Runyu Yue

[Issue 50456-2 identify issue code location](#)

 Runyu Yue

[Issue 50456-3 read/understand what cause the issue, testing with problem input](#)