owo what's this???

Team 24 - Deliverable 1

Table of Contents

Software Development Process

Overview

Pandas is a considerably large project with thousands of lines of code. We will be contributing to this project as a team of four developers with the goal of submitting 3-5 issues over the month of March. To accomplish this, we need to decide on an adequate software development process model to help organize ourselves for the upcoming weeks. We will go in detail of the pros/cons of each one.

Incremental development

Although this process is banned for selection, we still believe that it will be beneficial to discuss since we can incorporate some aspects of the process. Iterative development is a technique used for relatively large software projects that need to be maintained (i.e. the software is versioned). Generally an agile/scrum process is incorporated so that weekly meetings with a product owner, in this context a project maintainer, can happen. This allows developers to have a good feedback loop on the features they're working on, so their work can better suit the users.

The issue with this approach is that the audience for this project are mainly developers. These developers are required to submit detailed requirements for features that are sufficient to meet the maintainers criteria. Once these requirements have been vetted by a maintainer, they are available for contribution. Hence, there is no incentive for a short feedback loop since it's importable that requirements will change once development for a feature has begun.

Additionally, this development methodology is great for maintaining intermediate versions of the software. However for the scope of this project and our tasks, we won't need to worry about versioning of the software since that will be handled by the maintainers of Pandas.

Waterfall process

The waterfall process is a plan-driven software development technique where all stages of the development lifecycle are sequential. There are a few stages in following the waterfall model - Requirements analysis and definition, system and software design, implementation and unit testing, integration and system testing, operation and maintenance.

In our case, we would first identify the given issues and sub-issues at hand and we would devise a plan to fix it. Then, carry out the plan and implement the solution for the issue. Finally, we would create a set of relevant unit tests to ensure code stability. This is a relatively straightforward software process to follow, as it seems to follow the contribution guidelines for the project that many other developers follow. Hence, it would be easy for us to coordinate with maintainers in this fashion.

One of the biggest issues with this method however, is that potential to change to the requirements could invalidate a lot of work - especially during the earlier stages of development. This however, isn't too much of a non-issue, as discussed above in the iterative section. Most tasks will have very little change required since they most likely would have been vetted, and discussed, by the maintainers.

Another issue that arises is the rigidness of the design, since a large amount of design work is done before the issue is worked on. Thankfully, this is made easier with the help of the maintainers, since they usually require developers to discuss the design of the project before creating the work. As a result, pull requests become easier to review later on, and least likely to fail in any regression tests as well.

Reuse orientated

This type of methodology is great when trying to resolve a user's issues with existing software suites available to us. This is most commonly seen in web programming where common web elements, like components in frameworks and styling, can be reused and combined in different areas of the code.

This will be relevant for the pandas project depending on the task at hand. If the task is mostly related to fixing existing functionality inside the codebase, then there's no need to reuse other software suites. However, if the task is to generate a new feature, we might be required to incorporate other existing software projects, and design a system around this. This has already been the case looking at the system architecture of Pandas, where portions of matplotlib are reused to facilitate simple plotting features.

Our process

The process that we will be ultimately deciding on for this project will mirror the waterfall process with selected elements from reuse and iterative development. We will be following the waterfall model closely because it matches the needs in the contributing guidelines most closely - Requirements analysis and definition, system and software design, implementation and unit testing, integration and system testing, operation and maintenance.

For "requirements analysis and definition", we will identify open issues on the Github repository that are appropriate for members of our team to resolve. We would then consult each other and decide on which tasks we wish to pursue. According to the contributing guideline, we will then assign one or more of our members to that issue, create an estimate on development time, and allot the appropriate amount of time to set a deadline.

For "system and software design", we will evaluate our options with reuse development to see if we can save time by incorporating existing software similar to how they incorporated Matplotlib. Otherwise, we will iteratively work on the

software features each week, attempting to have some baseline features worked out.

For “implementation and unit/integration/system testing”, we will follow the guidelines in the repository to ensure that we’re not failing any regression tests. If required, we would develop new unit tests, or run existing integration tests, depending on the complexity of our change.

Finally for “operation and maintenance”, we will submit a pull request to indicate that the issue has been resolved. This will allow us to receive feedback on the maintainability of our feature.

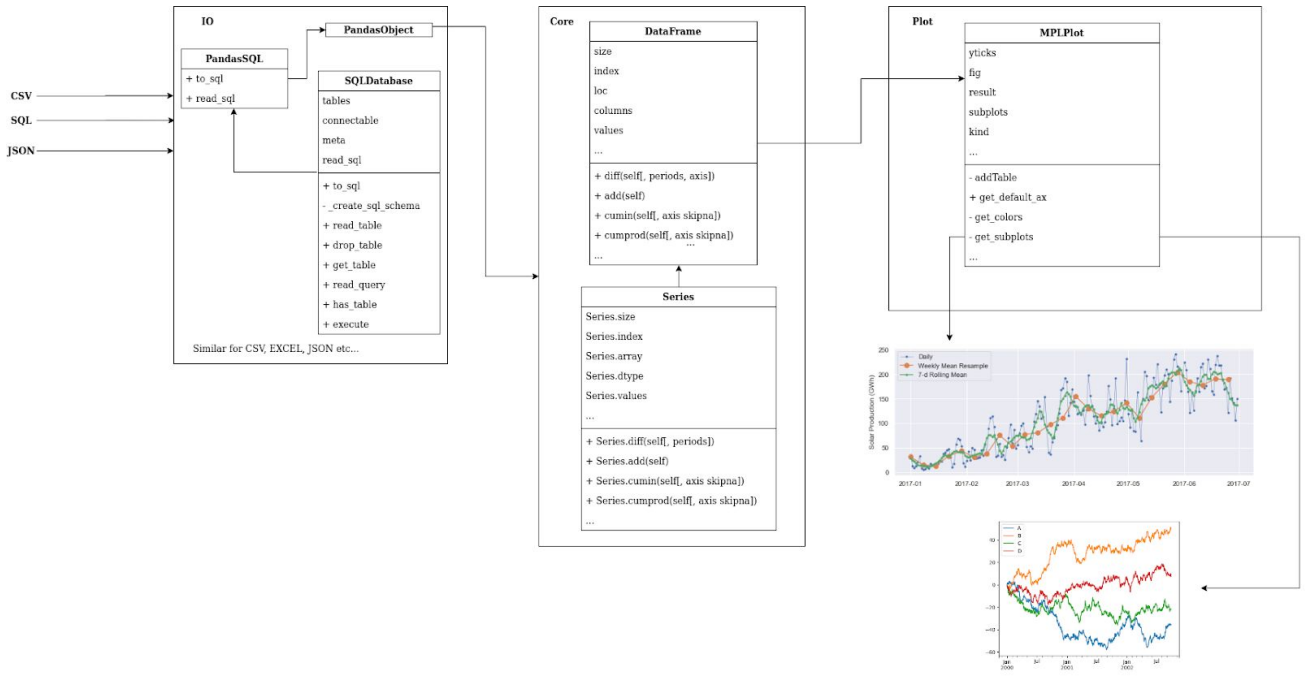
System Architecture of Pandas

Overview

Pandas, has multiple sub directories representing different modules of the library that encapsulates specific functionalities. The main modules of the library are the following,

- **pandas/core:** The main driver for pandas, containing most of the tools and data types that Pandas offers. Mainly used to initialize data frames and arrays, that allows users to store data, manipulate it's form and values, and to visualize them as tables or charts.
- **pandas/tests:** A suite of tests to ensure functionality of changes made to Pandas by the community. This is automatically executed through Azure pipelines, on multiple OS platforms, once a pull request is submitted on GitHub. This makes up a large bulk of classes within the project repository, and must all pass in order for the PR to be merged into master.
- **pandas/io:** Pandas is capable of reading and writing data to a multitude of file types. This component handles inputting files as bytes and identifying the format they are in to correctly read all the headers and to take in all the corresponding information. Same applies for writing to different formats and the serialization of data.
- **pandas/plotting:** Pandas has integrations with Matplotlib which allows for the two libraries to work together seamlessly. This module acts as an adaptor for converting Matplotlib data structures into Pandas data structures, and vise versa. Hence, Pandas is able to represent its data not only in tables, but in graphs and charts as well, using Matplotlib as an external library.

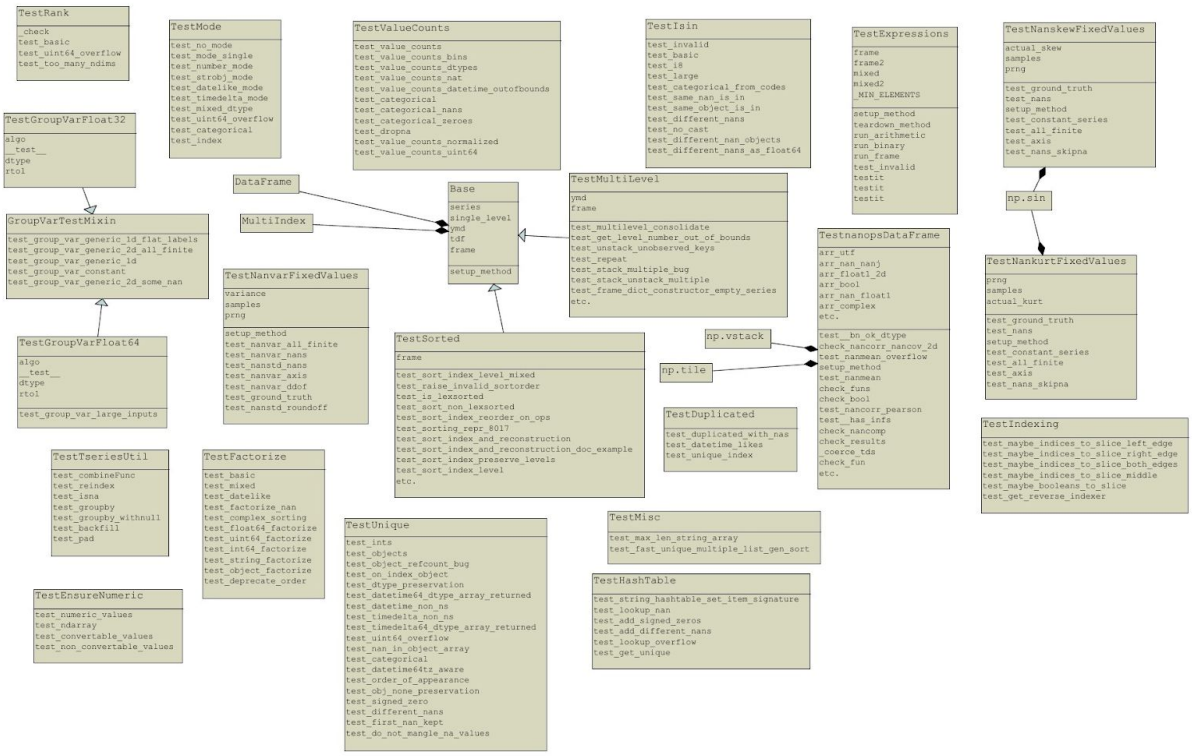
pandas/core



Core contains, and revolves around, the central object of **Frame**. Originating from R, Frame acts as a container for a variety of data types, supporting multiple storage containers such as dictionaries and data types like time series. Within the cores module, there an encapsulated standard functions defined that can be used to manipulate data within Pandas objects.

Pandas has categorized mathematical functionality together in **computation**, which allows Pandas to parse and evaluate mathematical operations such as arccosh and apply it to specific cells in tables. Some other functions allow data manipulation such as **groupby** and **reshape** that modify the format and presentation of tables, each housed within their own sub module.

pandas/tests



[The UML above is a sample of one of the many sub test folders]

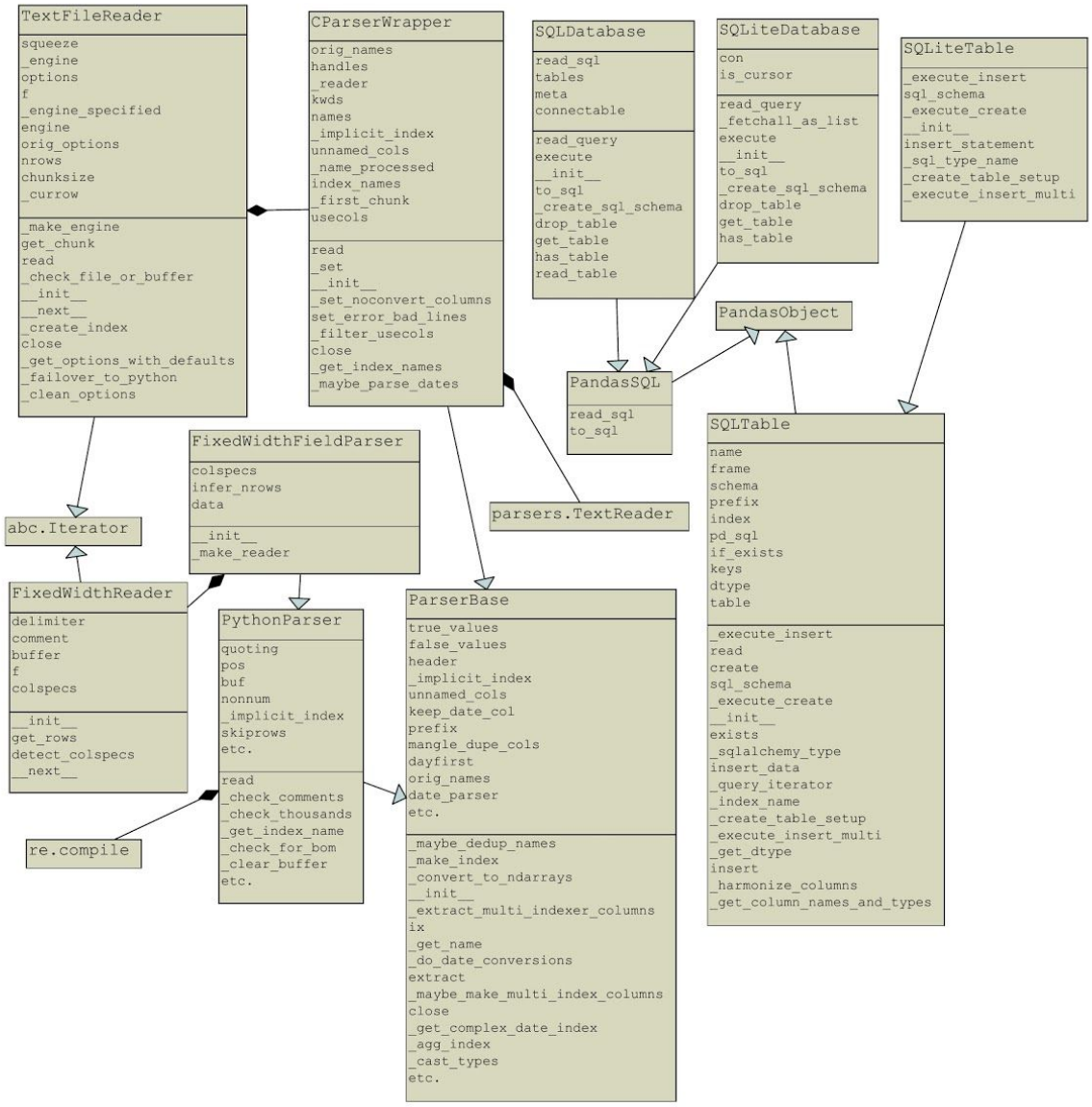
Spanning from every module in the project, **tests** contain the test suite for the correctness and functionality of each and every object, function and IO operation that Pandas provides to the user.

The test cases for objects seeks to verify that featured functions for Pandas are successful on manipulating the corresponding object types - such as **tests/frame** for testing **reshape** and **operators**.

The test cases for IO seeks to verify that the data written out by the module, is properly represented with what is stored internally. Using byte readers, the test cases can read the various types of formats, such as **.xls** or **.ods** files, to extract its data for comparing with internal data.

The testing is performed through multiple custom defined assert functions in the **._testing** module. It is also noted that some testing relies on the **hypothesis** library to find cases that will break functions by poking holes in different areas of the code. Pandas, is created following a test driven development process, and these tests are critical to ensure a functional library.

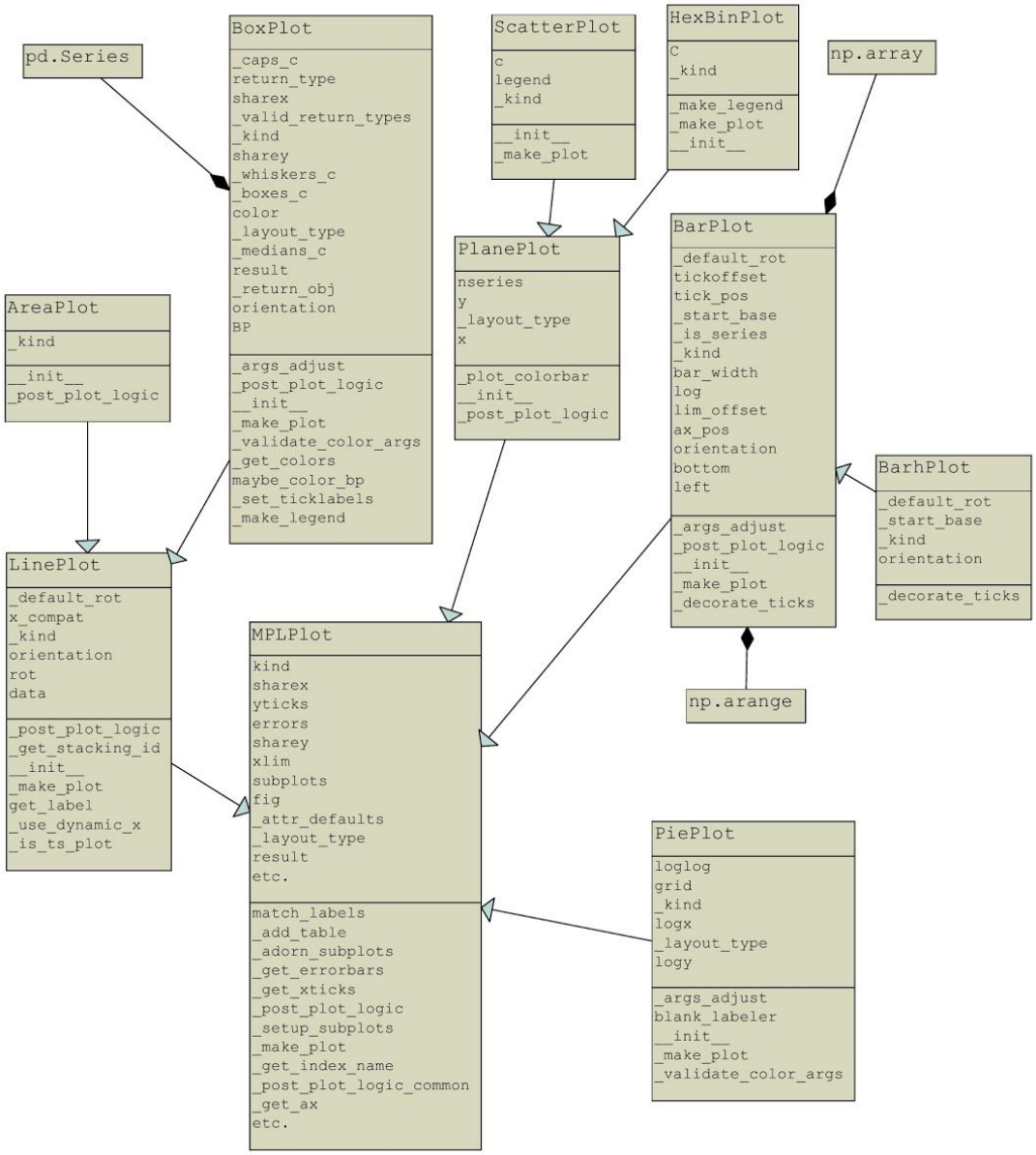
pandas/io



Supporting a large number of file formats, the IO tools allows Pandas to read and write data to different files. These tools demonstrate the adapter design pattern since it defines a collection of interfaces to relay file types to their corresponding tool. There are 18 different supported formats found here - varying between text, binary, and sql based inputs/outputs. Each file adaptor is based off of the general **parser**

module, defining the most basic reader that Pandas uses. It is responsible for reading text, binary, or tables, by parsing the inputted parameters and grabbing chunks of data from the inputted file. It also manages the SQLite Database that is used to store the Panda objects generated.

pandas/plotting



Using plenty of functions from Matplotlib, Pandas is able to create different graphs, such as histograms or boxplots. The driver for these plotting tools are contained in the **plotting/core** module, and uses components from Matplotlib to visualize different sets of data with various colours. There are also more specific

objects that focus on specific functionalities - such as the **timeseries** object, that handles the collection of time data.

Other modules

Pandas contains more auxiliary modules that add to core's basic and base functionalities Some of these modules include, but limited to,

- **Tseries:** Utility functions and objects that help support the functionality of time series plots, and data structures. Some functions include searching out specific dates of the week, or getting holidays of the year.
- **Arrays**: Self defined array objects, that act as adaptors, to hold different types of data structures offered by Pandas. Some of these include DatetimeArray, PeriodArray, and TimedeltaArray.
- **API**: A simple module that exposes parts of the core module to the public. Looking at the root __init__.py file, we can see that most functionalities of the library come from exports from the api directory of their respective module. Some modules that exposes their functionalities from api include reshape, tseries (partially), and core.
- **Compat:** A utility module that provides tools for retrieving environment information, and dependencies checking. Some of the environment information compact extracts is type of OS, version of Python, and if it's 32/64 bit. Using this information, Pandas would provide warnings such as missing dependencies, or functionality deprecations.
- **Errors:** A collection of self defined errors that are to be thrown and caught by Pandas itself. These errors are unique to Pandas. Some errors include DtypeWarning, ParserWarning and AccessorRegistrationWarning.
- **Util:** Auxiliary functions that help provide APIs and decorators for general use across the entire library. Some of these functions include functionalities such as,
    - Decorators for functions to add a prompt to inform users of Pandas about depreciated functions.
    - An API for table plotting in Matplotlib

- Validators such as argument length checking, axis params checking, etc.

Pandas Structure

The overall structure of Pandas contains supporting modules that helps manage and maintain the repository, help contributors and document the progress of the project. Multiple modules that help this process such as **doc**, **flake8**, and **ci**, are located in the root of the repository. A couple scripts are also located here to help set up and build the environment that Pandas will be executed in. It is nice and convenient that most of the supporting tools are located here and helps categorize the tools that help keep Pandas coherent.

Pandas uses Cython to execute these auxiliary tools. It is interesting that they pick Cython as it emulates the speed and efficiency of C while retaining the clear and cohesive syntax and structure of Python. Given a test driven development model, it makes sense that the Pandas community would want a fast way to run their test suite and to ensure the code is up to their standards.

It is also nice that Pandas as included a directory containing a complete documentation of their functionalities, as well as a cheat sheet, tips on working with Pandas and a suite of tests for contributors to use when developing.

From an architectural perspective, Pandas have done a good job at modularizing their key features as seen from above. They also achieved encapsulation, important for a language that doesn't have modifiers, by having api modules that export only functions that they want other modules to have access to.

A possible improvement for this library might be to introduce a dependency inversion design pattern for their plotting module. Looking at how dependent this module is to Matplotlib, where _core returns only Matplotlib objects, we can see that the introduction of other plotting libraries are impossible/difficult - requiring additional retranspiling of the data. Perhaps introducing a similar design like done in io and arrays can help improve the flexibility of this module.