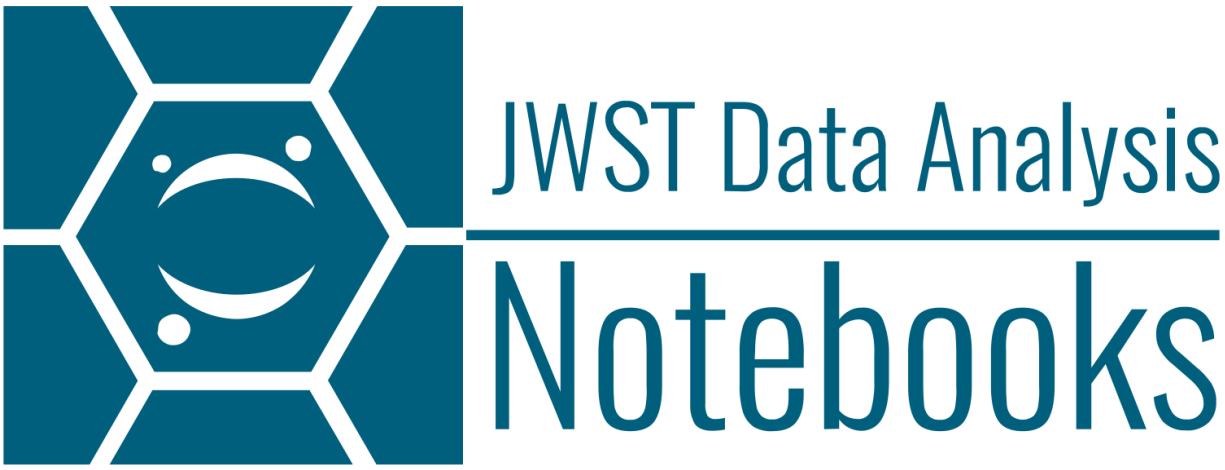
dat$_{p}yinthesky Documentation$

dat$_{p}yinthesky$

Feb 13, 2021

CONTENTS


JWST Data Analysis
Notebooks

CHAPTER

ONE

JDAT AND JDAVIZ

The James Webb Space Telescope Data Analysis Tool (JDAT) team at the Space Telescope Science Institute (STScI) is developing analysis and visualization tools for the upcoming James Webb Space Telescope (JWST). JDAT is involved with the development of astropy (https://www.astropy.org) core and its affiliated analysis tools (such as specutils (https://specutils.readthedocs.io/en/stable/) and photutils (https://photutils.readthedocs.io/en/stable/)). JDAT is also responsible for the development of the JWST Data Analysis Visualization tools (`Jdaviz`). `Jdaviz` is a package of astronomical data analysis and visualization tools based on the Jupyter platform. The `Jdaviz` package includes the following visualization applications:

Application	Description
`Specviz`	Visualization and quick-look analysis for 1D astronomical spectra.
`Cubeviz`	Visualization and analysis tool for data cubes from integral field units (IFUs).
`MOSviz`	Visualization and quick-look analysis tool for multi-object spectroscopy (MOS).

See also:

- For more information on JDAT please see the Data Analysis Tools JDox Page (https://jwst-docs.stsci.edu/jwst-post-pipeline-data-analysis).
- For more information on `Jdaviz`, please visit the Jdaviz documentation (https://jdaviz.readthedocs.io/en/latest/).

CHAPTER

TWO

JDAT NOTEBOOKS

The JDAT team is seeking help from the scientific community to develop example notebooks that outline scientific workflows utilizing the analysis and visualization tools developed by the team. These notebooks will be made available to the public to serve as teaching resources and a form of interactive documentation of the analysis tools.

The submitted notebooks should satisfy the following goals:

1. Reduce and analyze JWST data (we currently use simulated or similar data).
2. Showcase analysis and visualization tools.
3. Drive development by identifying missing functionalities in the tools and libraries.

Note: Completed notebooks are rendered and made available to the public in the jdat_notebooks (https://github.com/spacetelescope/jdat_notebooks) repository. Development of new notebooks is facilitated through the dat_pyinthesky (https://github.com/spacetelescope/dat_pyinthesky) repository.

CHAPTER

THREE

JDAT DEVELOPMENT SPRINTS

The JDAT team at STScI develops the `Jdaviz` applications during two week sprints through out the year. During this time developers are available to assist with the development of notebooks. Notebook leads are strongly encouraged to participate in the JDAT sprints while composing their notebooks because it offers an opportunity to work with the developers and gain exposure to the latest tools. During a Sprint, we expect notebook leads to learn how to contribute their science expertise towards the development of tools via GitHub issues, JIRA tickets, Jupyter Notebook coding, communication with developers, and all the machinery in between. The JDAT team also has staff members dedicated to helping notebook leads with `GitHub` and `Box` related workflows. If you are interested in joining these sprints, please contact one of the maintainers of the dat_pyinthesky (https://github.com/spacetelescope/dat_pyinthesky) repository or file a ticket in the STScI help desk (https://stsci.service-now.com/hst). STScI and affilated staff should refer to the *STScI Notebook Leads* section for instructions.

CHAPTER

FOUR

TABLE OF CONTENTS

4.1 Development Procedure

This document is a description of the JWST Data Analysis Tools (JDAT) approach to "Notebook-Driven Development". The procedures here outline the process for getting a notebook through successive development stages to become something that can be "live" on the spacetelescope notebooks repository.

These notebooks can have many varied science cases, but follow a relatively standard workflow:

1. *Draft Stage*
2. *Baseline Stage*
3. *Notebook-driven development*
4. *Advanced Stage*
5. *Final Notebook - Maintenance Stage*
6. *Revision based on Community feedback*

These stages and the process for moving from one to the other are described below.

The procedure to submit the notebook via a Pull Request is described at the *GitHub section* of this documentation. This is repeated for each of the 5 stages.

Note that there is much more information on writing Jupyter notebooks at the STScI notebook style guide (https://github.com/spacetelescope/style-guides/blob/master/guides/jupyter-notebooks.md), and similar guidance for Python code at the STScI Python style guide (https://github.com/spacetelescope/style-guides/blob/master/guides/python.md). These guidelines are in place to make review steps easier.

Important: Please note that all JDAT related code is written `Python 3`; `Python 2` is not supported.

4.1.1 Draft Stage

The primary purpose of this stage is to record a scientific workflow, but without including actual code. This stage is generally done primarily by a scientist. Reasonably often, notebooks can skip this stage if they are simpler or if the underlying tools are already well-enough developed to be immediately implemented.

To begin a notebook at this stage, the notebook author should start with either the notebook template from the notebook style guide (https://github.com/spacetelescope/style-guides/blob/master/guides/jupyter-notebooks.md) or a blank Jupyter notebook. They then write out their workflow in words. Where possible, they put *example* code of the sort they would *like* to see, even if it is not implemented yet.

For example, an author might write this in such a notebook:

```
In [ ]: spectral_line = find_line(jwst_miri_spectrum)

# `spectral_line` should be a list of line centers and names of lines indexed by␣
↪spaxel,
# found using a derivative-based line-finder.
```

even if the `find_line` function doesn't yet exist anywhere.

The top-level header of the notebook (i.e., the title) should have "Draft: " at the start to make it clear this is a draft notebook. The filename should *not* have `draft` in it, however, as the filename will generally remain the same throughout the later stages.

Once they have the draft ready, the author should create a pull request with the draft notebook's content (see instructions at the end of this document).

4.1.2 Baseline Stage

The primary purpose of this stage is to get a functioning notebook to record a workflow. This stage is also typically done by a scientist (although with developers available to ask questions). It is also frequently the *first* step of development. That is, if the workflow is already reasonable to implement with existing tools, the draft notebook is not necessary.

In this stage the notebook should actually execute from beginning to end, but it is fine to be "rough around the edges". E.g., the notebook might have several cells that say things like:

```
In [ ]: spec = Spectrum(np.linspace(a, b, 1000)*u.angstrom, some_complex_function(...
↪))
```

the scientist might think this is too complicated, and so to communicate their desire for an improved workflow, they create a "Developer Note". A developer note should be a part of the notebook itself and should be a single markdown cell (not code cell - code examples in a dev note can be done as literal markdown blocks - i.e. surrounded by ``` for blocks or ` for inline code). That cell should begin with the text `*Developer Note:*`. E.g., a markdown cell might be added below the above cell in a notebook, which would say:

```
*Developer Note:*
Creating the spectrum above is a bit complicated, and it would improve the workflow␣
↪if there was a single
simple function that just did `spec = simulate_jwst_spectrum(a, b)`.
```

thereby providing guidance for where specific development would simplify the workflow.

If a notebook is freshly created in this form, the author can follow the "Procedure to submit a notebook as a Pull Request" (found at the end of this document), skipping the Draft Stage step.

If the notebook was already created in the Draft Stage step and the "Procedure to submit a notebook as a Pull Request" has already been followed, the author should just create a new branch to modify the existing code and then create a new Pull Request with the changes once they are ready.

In either case, the title (but not filename) of the notebook should begin with "Baseline:" to indicate the notebook is in the Baseline Stage.

Once the Pull Request has been created, the notebook will automatically be built in the repository so that reviewers can view it. Reviewers can then comment on the notebook in Github. At this stage the bar is still relatively low for review - primarily things like ensuring the notebook does run from beginning-to-end and that data files or the like were not accidentally committed to the repository.

Finally, there are three important technicalities for notebooks that become relevant at this stage (and continue for future stages):

1. The output cells of a notebook should *always* be cleared before a git commit is made. Notebook outputs can sometimes be quite large (in the megabytes for plots or the like), and git is intended for source code, not data. Clearing the outputs also ensures the notebook can be run from beginning to end and therefore be reproduced by others.

2. Any data files required for a notebook need to be accessible by others who may be reviewing or testing the notebook. The STScI guidelines on data storage for notebooks (https://github.com/spacetelescope/style-guides/blob/master/guides/where-to-put-your-data.md) should be followed here. The specific addition for the JWST Notebooks is that notebook data should be in the `DMD_Managed_Data/JWST/jwst-data_analysis_tools` Box folder (or subfolders thereof). If you do not have access to this box folder already, ask a Project Scientist and they should be able to get you added. Note that if a baseline notebook is using data that should not yet be public, the easiest choice is probably central store, but in that case it is critical that the notebook state prominently that it must be run inside the STScI network.

3. A notebook should state clearly what version of various dependencies were used to generate the notebook. These versions should be placed in a `requirements` file in the same directory as the notebook itself. An example of this file is in the``example_notebook`` folder. That will ensure reviewers/testers can be sure that if they encounter problems, it is not due to software version mis-matches.

The notebook will undergo a scientific and a technical review, which might also yield additional developer notes. It will then be merged into the repository once the review comments have been addressed. This concludes the Baseline Stage.

4.1.3 Notebook-driven Development

Along and after the Draft and Baseline stages, there is potential for considerable development to be necessary. A baseline notebook may contain a large number of areas where more development is desired in data analysis tools, or it may only require a few minor adjustments (or none at all!). This stage is therefore the most flexible and dependent on developer resources, etc. In general the intent is for developers to be able to re-use bits of code from the notebook as tests for development, while occasionally (if necessary) asking the notebook author for guidance to ensure the implementation actually meets the notebook's needs. There is not a formal process for this step, but it is intended that the JDAT planning process (currently on Jira) keeps track of specific steps needed before a given notebook can proceed on to the next stage.

4.1.4 Advanced Stage

Once a baseline notebook has been completed, the next stage is to build the baseline into a notebook that uses the DAT's or associated community-developed software as consistently as possible. This is typically done via a developer reviewing a baseline notebook and working with the scientist to develop additional DAT code, particularly focused on resolving the "developer notes". It is at the discretion of the notebook author and developer together which of them actually modifies the notebook and sources the Pull Request, but it is likely both will be involved to some degree. An example approach is for the developer to take the baseline notebook, mark it up with comments like (using the example from above):

```
In [ ]: spec = Spectrum(np.linspace(a, b, 1000)*u.angstrom, some_complex_function(...
↪))
```

Creating the spectrum above is a bit complicated, and it would improve the workflow if there was a single simple function that just did `spec = simulate_jwst_spectrum(a, b)`

```
*Development:*
This has now been implemented as JWSTSimulator.make_spectrum(a, b,␣
↪anotherparameterthatturnsouttobeimportant).  Can you try that and ensure it works␣
↪here?
```

and then create a git commit with these comments. The original author would then address the comments in a follow-on commit. There might be multiple pull requests of this sort as the notebook driven development continues. But once all developer notes have been addressed, the developer and author can declare the notebook ready to be called "Advanced".

Once the notebook authors (original author and developer/reviewer) have agreed it is ready, one of them follows the Pull Request workflow as described above, but with the notebook title now changed to be just the title itself (no "Draft:" or Baseline:"). The Pull Request is then reviewed by one of the project scientists, and merged when everyone is satisfied with the notebook.

4.1.5 Final Notebook - Maintenance Stage

The final stage for the notebook is release on the official STScI notebook repository (https://github.com/spacetelescope/notebooks). Specific documentation for this last stage is given in the repository itself. However, that repository and the working repository here have very similar structure, so it is in principle simply a matter of copying the advanced notebook over to a form of the release repository and doing one final Pull Request. Note, however, that other STScI reviewers may comment on this stage. It is also important for the authors to do an additional check over the notebook to ensure that it uses *released* (not developer) versions of requirements where possible. It is also a good opportunity to fill in the scientific context of a given notebook - e.g. add a motivation section, or a final plot at the bottom that shows the final science result. Once this is done, and the Pull Request merged, the notebook can be declared complete.

4.1.6 Revision Based on Community Feedback

Of course, science does not stand still! As time passes some of the completed notebooks may have enhancements or changes necessary. In general these follow the standard Pull Request workflow and can be submitted by anyone once the notebook is public (both in and out of STScI). While the repo maintainers manage this process, the notebook authors may be called in from time to time to provide opinions or perspectives on any proposed changes.

4.2 Submitting Notebooks

This section outlines how to submit notebooks. New notebooks development occurs in the dat_pyinthesky (https://github.com/spacetelescope/dat_pyinthesky) repository. As such, new notebooks should be submitted by making a pull request (PR) in the STSCI dat_pyinthesky repository (this is the *Pre-Baseline Stage*). After the *review process*, the notebooks will be merged (this is the *Baseline Stage*). At this point, it will be public to the community and will be moved to a cleaner github repo jdat_notebooks (https://github.com/spacetelescope/jdat_notebooks) where notebooks are also rendered (https://spacetelescope.github.io/jdat_notebooks/). All further development will happen directly in the dat_pyinthesky (https://github.com/spacetelescope/dat_pyinthesky) repository!

What to Submit:

1. *Data Files* (Upload to *Box*).
2. *Jupyter Notebooks* (Upload to *GitHub*).
3. *Requirements File* (Upload to *GitHub*).

4.2.1 Check List

Before and after submitting your notebook, requirement files and data, please make sure the following items are in order:

```
- [ ] Box:

    - [ ] Data has been uploaded to box.
    - [ ] The data is bing shared via link. Make sure the settings allow for anyone␣
↪with a link to download.

- [ ] GitHub:

    - [ ] Notebook has all cell outputs cleared out.
    - [ ] Notebook is written in `Python 3`
    - [ ] All imports occur at the beginning of the notebook.
    - [ ] All data is loaded into the notebook via a URL from box. (No local files␣
↪being used).
    - [ ] Verify that the python code satisfies PEP 8.
    - [ ] Comments and unused lines of code are removed.
    - [ ] `requirements.txt` file is included.
```

Tip: If you copy and paste this checklist into your PR as a comment or description, it will render as a checklist with radio buttons you can toggle any time.

4.2.2 Box Submissions

All data files should be uploaded to STScI's box folder and made sharable via URL. All notebooks should use human readable URLs in the notebooks when loading/reading data. For instructions on submitting data files, please visit the *Uploading Data to Box* section.

4.2.3 GitHub Submissions

Science Notebooks and *requirements files* files should be submitted by making a pull request against the `main` branch of the STScI dat_pyinthesky (https://github.com/spacetelescope/dat_pyinthesky) repository. For instructions on how to crate a GitHub pull request, please see the *GitHub Guidelines* section of this documentation.

Important: New notebooks should be added to the `dat_pyinthesky/jdat_notebooks` directory.

You must first create a new folder in the `dat_pyinthesky/jdat_notebooks` directory and name the new folder something relevant to the topic of the notebooks being submitted (think of this a short title). For example, `dat_pyinthesky/jdat_notebooks/spectral_fitting`. This "title" will also be used to name the folder on Box containing the *data files* used by the notebook. After creating a new folder and naming it, please place all notebooks and requirement files inside of it.

The folder name ("short title") should be:

- Related to the topic of the notebooks.
- Unique to avoid confusion/conflicts with existing notebooks.
- Reasonable in length (makes navigating in a terminal easy).

- All small letters.
- Using underscores instead of spaces. For example "spectral fitting" -> "spectral_fitting"

4.2.4 GitHub Guidelines

GitHub Setup

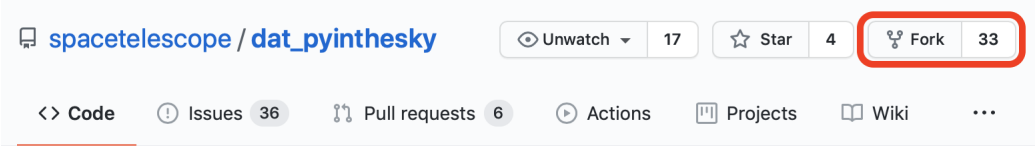
GitHub Accounts

GitHub, is provider of internet hosting for software development and version control using Git. It offers the distributed version control and source code management functionality of Git, plus its own features. If you don't have a GitHub account, please visit GitHub's sign up page (https://github.com/join).

Repository Setup

Step 1: Fork the STScI Repository

The first step is to create a GitHub copy of the STScI dat_pyinthesky (https://github.com/spacetelescope/dat_pyinthesky) repository. The copied repository on your GitHub account is called a "fork". To make a fork, click on the ``**fork**`` button at the upper right corner of the STScI's dat_pyinthesky (https://github.com/spacetelescope/dat_pyinthesky) repository. A dialog should popup up and you can select your GitHub account to create the fork.



Step 2: Clone the Repositories

After you create a fork of the STScI repository, you should now have a copy of the STScI repository under your GitHub account. The next step is to make a copy of both your and STScI's GitHub repositories on your local machine. That way you can make edits on your computer and send the changes to your online repository (fork). The process of making a local copy is called **cloning**. To clone the `dat_pyinthesky` repos, open up a terminal window and `cd` into a directory you would like to save your local copy. Then run the following bash commands:

```
# clone the repository
git clone https://github.com/spacetelescope/dat_pyinthesky

# cd into the local repository (clone)
cd dat_pyinthesky

# rename the online version of the repo
# from "origin" to "stsci" (since you cloned STScI's version)
git remote rename origin stsci
```

Next, you want to link your online GitHub version of the repository (fork) to your local version. Copy the URL to your fork and run the following command in the terminal (make sure you are still in the `dat_pyinthesky` folder in the terminal):

```
# add your GitHub copy
git remote add <your_github_username> <your_fork_url>
```

Warning: Make sure not to confuse your fork's URL with the official STScI URL. Your fork's URL will have the following format:

```
https://github.com/<your_github_username>/dat_pyinthesky
```

Step 3: Check Remote URLs

At this point, you have finished setting up your local clone of the repository. To check if the setup was successful, run the following command to list URLs:

```
# list remote URLs
git remote -v
```

This should return:

```
<your_github_username>  https://github.com/<your_github_username>/dat_pyinthesky.git␣
↪(fetch)
<your_github_username>  https://github.com/<your_github_username>/dat_pyinthesky.git␣
↪(push)
stsci   https://github.com/spacetelescope/dat_pyinthesky.git (fetch)
stsci   https://github.com/spacetelescope/dat_pyinthesky.git (fetch)
```

Step 4: Create a New Branch

Next, create a new branch and name it. You can name your new branch anything relevant to your notebook. Its best to use a short name since you wll need to type it in the terminal often. Do this by running:

```
git checkout -b <branch_name>
```

This will automatically change the current branch to the new branch; it will stay on this branch until you manually change it or create a new one.

Tip: To change to an already existing branch, run `git checkout <branch_name>`

Step 5: Synchronize New Branch to STScI

You should update the branch to make sure its in sync with the STScI `main` branch. To do this, run:

```
# Fetch changes
git fetch stsci main

# Pull (download) changes
git pull stsci main
```

Git and GitHub Workflow

This section outlines the Git and GitHub workflow. Each time you update your notebook, follow the instructions below to "save" your changes. The workflow outlined in the sub-sections below can be summarized with the following set of commands:

```
# cd into directory
cd dat_pyinthesky/jdat_notebooks/<name_of_your_folder>

# Check status
```

(continues on next page)

(continued from previous page)

```
git status

# Add files to commit
# For each file, run:
git add <file_name>

# Check status (files added should be in green)
git status

# Commit with comments
git commit -m "short description of changes"

# When ready to upload, Push to your fork
git push <your_github_username> <branch_name>
```

Step 1: Prepare Files Locally

Next step is to create or update your files to your local repository before uploading them to GitHub. As mentioned above, create a new folder with your project name in the `dat_pyinthesky/jdat_notebooks` directory. Add your *Science Notebooks* and *requirements files* into the new folder. Please make sure to *clear cell ouputs* in your notebooks before continuing.

Note: Remember to clear all cell outputs in your notebooks. See the *Clearing Notebook Outputs* for instructions.

Step 2: Add

Using your terminal, you must first `cd` into the folder you created:

```
cd dat_pyinthesky/jdat_notebooks/<name_of_your_folder>
```

Check Status: First step is to check what changes are available to be added and committed to the repository history. To get a list of changes and their status, run:

```
git status
```

This will return a list of files that have been added to the local repository. Files that are not staged for commit, are highlighted in red. Files staged for commit are listed in green. To select file to be staged for commit, you must first "add" them. Thing of "adding" as selecting which files you want to include in your commit.

Add Files to Upload: To add files to commit, run the following command for each file:

```
git add <file_name>
```

Check Status Again: After adding the files, you should check one more time if the files you intend to commit are staged. To do this, run:

```
git status
```

This time the files you selected should be in green font under `Changes to be committed`.

Step 3: Commit

Now you may commit the files to your local git history. When you commit changes, you should leave short comments describing the changes being introduced in the commit. To add a comment, you can append `-m "comments"` at the end of the commit command. To commit changes with a comment, run the following command:

```
git commit -m "short description of changes"
```

Step 4: Push

You can now push (upload) the changes to your GitHub fork. To do this, run the following command:

```
git push <your_github_username> <branch_name>
```

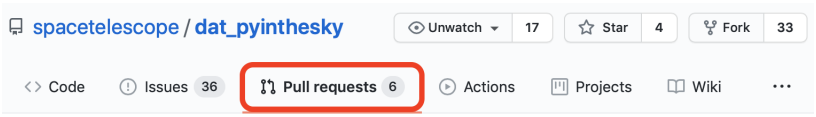
Tip: If you are not sure what branch you are working on, run `git branch`

You will be prompted for your GitHub user name and password. After entering your credentials, your changes will be uploaded to your GitHub fork (online copy).

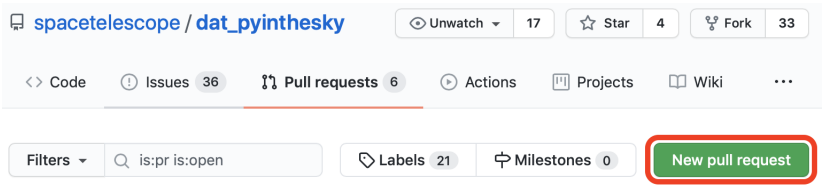
Making a Pull Request (PR)

Step 1: Open PR Tab on GitHub

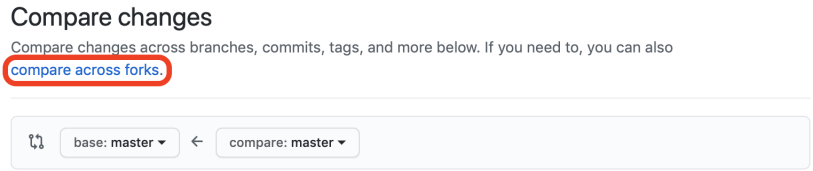
To create a pull request, navigate to the STScI dat_pyinthesky (https://github.com/spacetelescope/dat_pyinthesky) repository on GitHub and click on the `Pull Request`:



Step 2: Click New PR Button

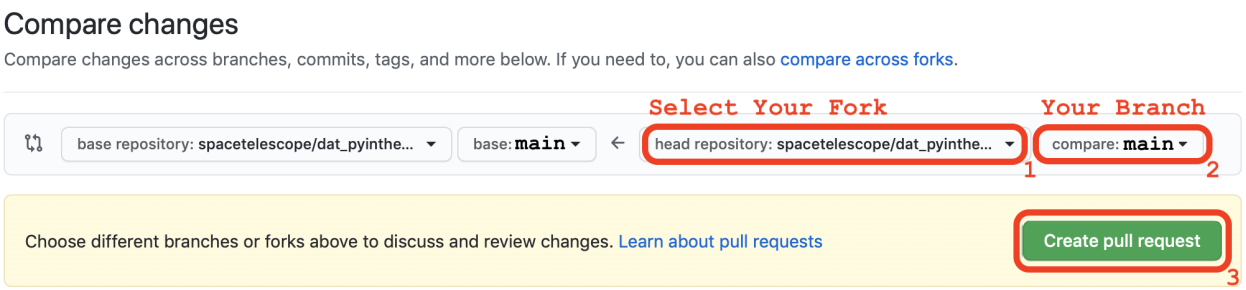


Step 3: Compare Across Forks



Step 4: Select Your Fork and Branch

Use the drop down menu to select your fork and branch. If you can not find your branch, first try to refresh the page. If you still can not find your branch, something probably went wrong in the *Git and GitHub Workflow* section.

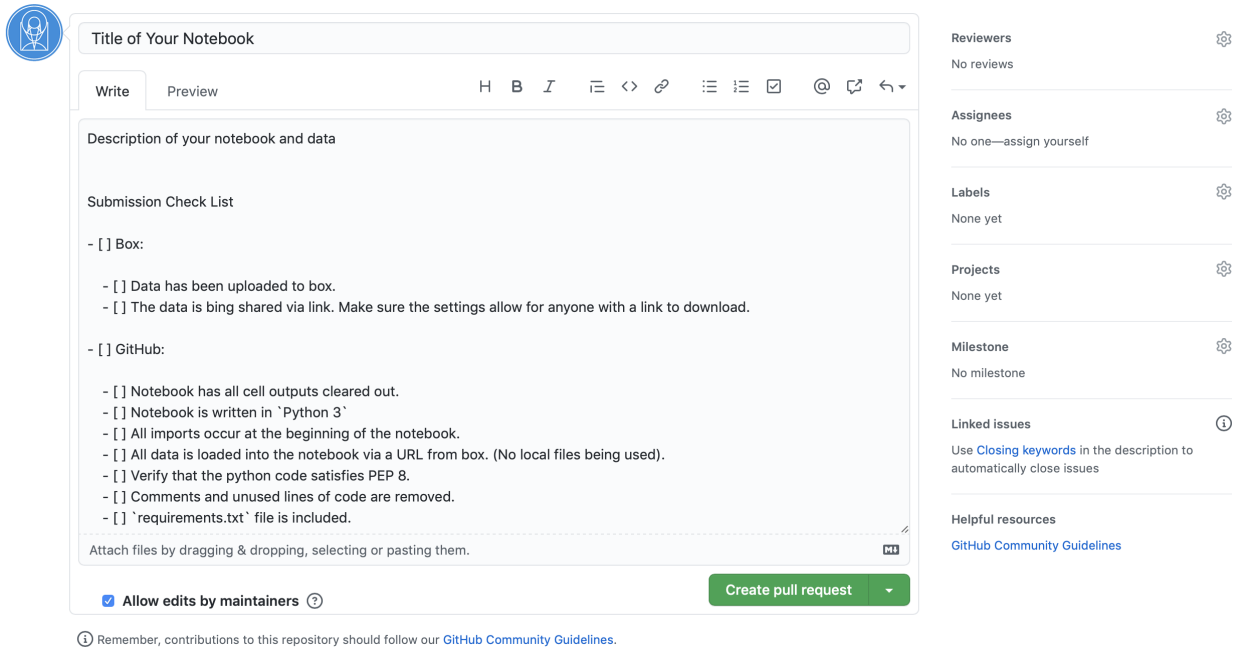


Note: Make sure the `base repository` (left side) is set to the `spacetelescope/dat_pyinthesky` and `main` branch.

Step 5: Write a Description and Create PR

Once the PR form pops up, fill in the title with the name of your notebook or project. In the description box, leave a description of your notebook and the data it uses.

Tip: If you copy and paste the checklist in the *Check List* section, it will render as a checklist with radio buttons you can toggle any time.



Step 6: Updating Your PR

Once the PR is created, you can update it by pushing new changes to **your fork**. This means that you can simply follow the steps described in the *Git and GitHub Workflow* section and GitHub will automatically update your PR to reflect the changes.

See also:

For more information on making pull request visit the GitHub PR documentation (https://docs.github.com/en/github/collaborating-with-issues-and-pull-requests/creating-a-pull-request)

4.2.5 Review Process

After creating a Pull Request (PR), your PR will undergo a science and technical review. The automatic testing infrastructure will also attempt to render your notebook. Reviewers will leave comments in your PR with suggested changes or give their approval. If changes are recommended or requested, you can *update your PR* via the steps described in the *Git and GitHub Workflow* section. Once the all reviewers approve and the automated tests pass, the PR will be merged into the official STScI repo.

4.3 Data Files

Since we will be sharing the notebooks with the astronomical community, the data analysed and visualized in the notebooks need to be easily downloadable. As such, the JDAT team at STScI has set up a Box folder for distributing data products. For security reasons, we only support data files hosted on STScI's Box folder. Please follow the guideline below on uploading and integrating your data files.

See also:

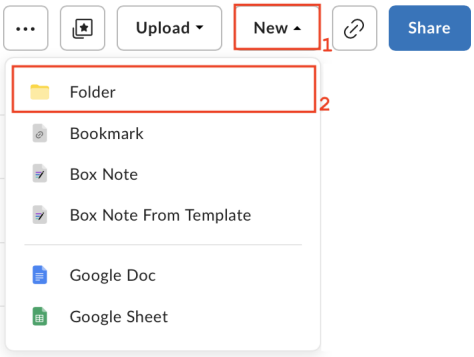
For the most up to date information on data storage at STScI, please visit the STScI guidelines on data storage for notebooks. (https://github.com/spacetelescope/style-guides/blob/master/guides/where-to-put-your-data.md).

4.3.1 Uploading Data to Box

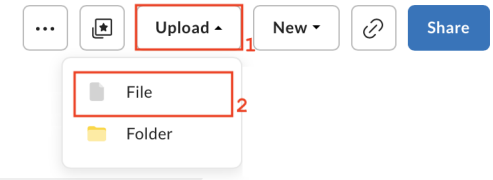
Important: In order to submit your data files, you will need a Box account. Please see the Box support page (https://support.box.com/hc/en-us/articles/360044196373-The-Basics-of-Box) for instructions. Once you have your Box account setup, contact the JDAT team for an email invite to the STScI `jwst-data_analysis_tools` folder. If you are not an STScI staff member, please request access in your pull request.

Tip: If you have multiple files, you should put them in a `zip` file before uploading them to Box.

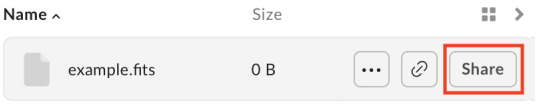
Step 1: Create a New Folder: Once you have access to the `jwst-data_analysis_tools` folder on Box, you may create a new folder with the same name as the folder containing your notebook and `requirements.txt` (see *GitHub Submissions* section for naming conventions). Once the folder is created, you may upload your data files into that folder.



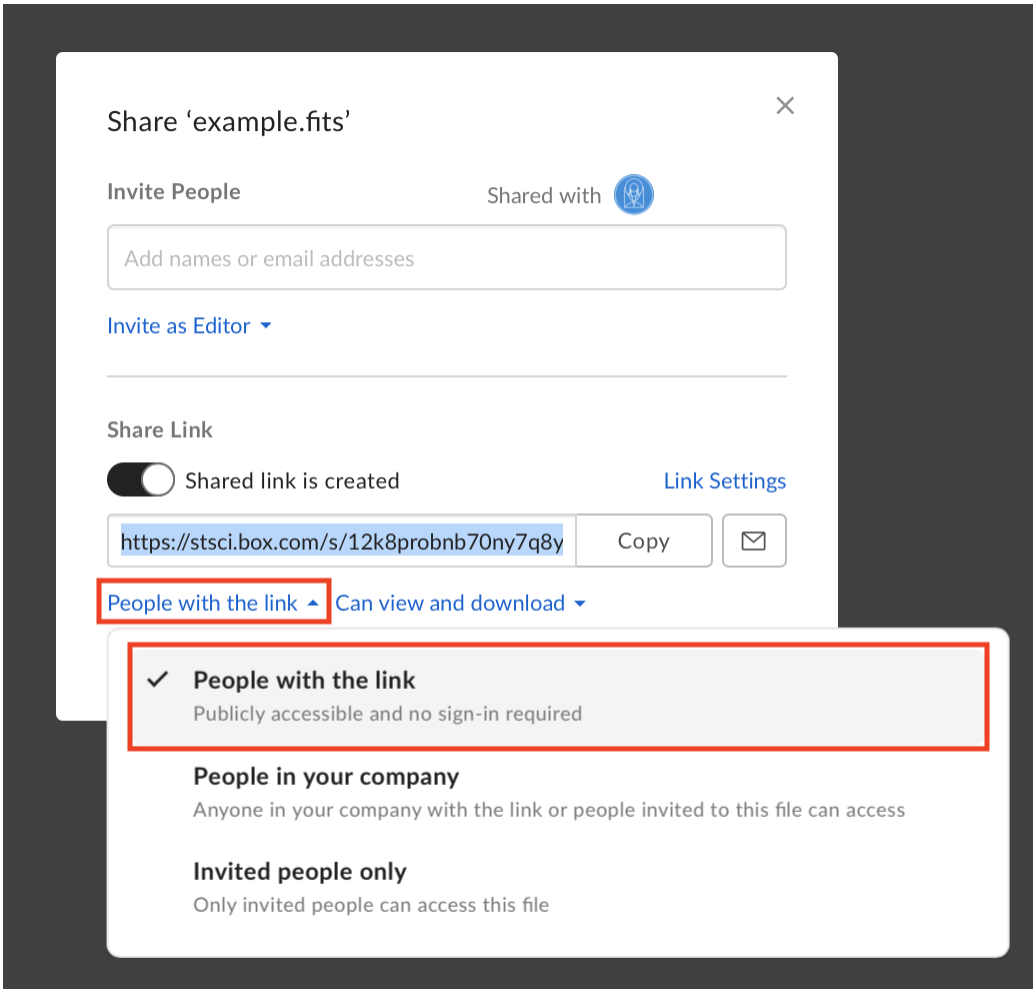
Step 2: Upload Data Files: To upload files, navigate into your the folder you created and select `File` under the `Upload` menu.



Step 3: Sharing Settings: The files you uploaded should be shareable to the public. For each file you uploaded and hover over the file name. Select the `Share` button to open the sharing options.



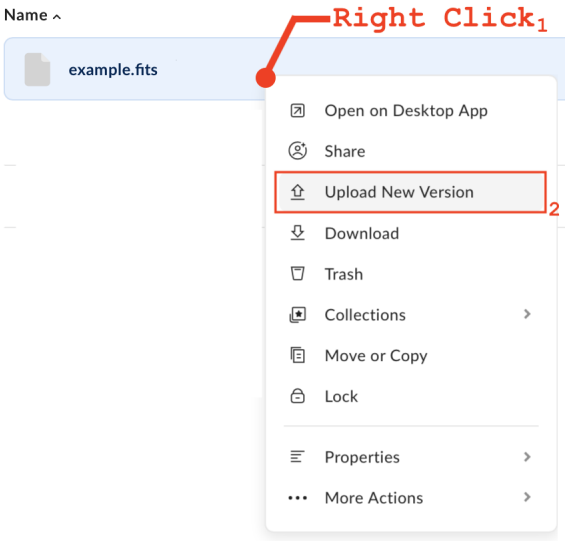
Once the sharing dialog opens up, make sure the `Shared Link` radio button is toggled on and that the `People with the link` option is selected.



Once the file is shared, a blue link icon will appear next to the name of the file.

Note: Make sure the folder is also being shared and that the `People with the link` option is selected. You can check by clicking the blue share button on the upper right corner of the page. You must be viewing the folder you created before clicking the button.

Step 4: Updating Data Files: You can upload new versions of your data on Box by right clicking the name of your file and clicking `Upload New Version`:



4.3.2 Box URLs in Notebooks

Important: Files on the STScI Box are assigned a human readable link and we request that notebook leads use this URL in their Notebooks.

Once your files are uploaded, you can use them in your notebooks via URL link. The human readable URL has the following format:

```
https://data.science.stsci.edu/redirect/JWST/jwst-data_analysis_tools/name_of_your_
↪folder/name_of_file.extension
```

For example, lets say you created a folder called `example_folder` and added a file named `example.fits`, the URL would be:

```
# The path on box:
jwst-data_analysis_tools > example_folder > example.fits

# The URL:
https://data.science.stsci.edu/redirect/JWST/jwst-data_analysis_tools/example_folder/
↪example.fits
```

You should now be able to use this URL just like any path in your notebook. In the example above, we can open the fits file using astropy as follows:

```
from astropy.io import fits
```

(continues on next page)

(continued from previous page)

```
data_url = "https://data.science.stsci.edu/redirect/JWST/jwst-data_analysis_tools/
↪example_folder/example.fits"
hdu_list = fits.open(data_url)
```

Note: If you are not able to open your file using URLs, please let the team know or leave a developer note in your notebook.

If you have to download a file or have a `zip` file, you can use the following code to download the file (and unzip for `zip` files) inside the notebook:

```
import os

# If the example dataset has already been downloaded, comment out these lines:
import zipfile
import urllib.request

boxlink = "https://data.science.stsci.edu/redirect/JWST/jwst-data_analysis_tools/
↪example_folder/example.zip"
boxfile = './example.zip'  # Specify output path and file name of downloaded file

# Download file
urllib.request.urlretrieve(boxlink, boxfile)

# Unzip .zip file
zf = zipfile.ZipFile(boxfile, 'r')
zf.extractall()
```

This example will download and extract data files into the same directory containing the running notebook. Since how you zip your files determines the directory structure of the unzipped data, please use your code to download the files and check to make sure the paths in your notebook match the file structure of your unzipped data.

4.4 Jupyter Notebooks

Please see the *Development Procedure* section for notebook development guidelines.

See also:

For detailed instructions on formatting notebook, please visit the STScI notebook style guide. (https://github.com/spacetelescope/style-guides/blob/master/guides/jupyter-notebooks.md)

4.4.1 Installation

For instructions on how to install Jupyter notebooks, please visit the Jupyter Installation page (https://jupyter.org/install).

We also strongly recomend using Conda (https://docs.conda.io/projects/conda/en/latest/index.html) for managing environments. If you need instructions on how to setup Conda, please see the Conda's Getting Started (https://docs.conda.io/projects/conda/en/latest/user-guide/getting-started.html) documentation.

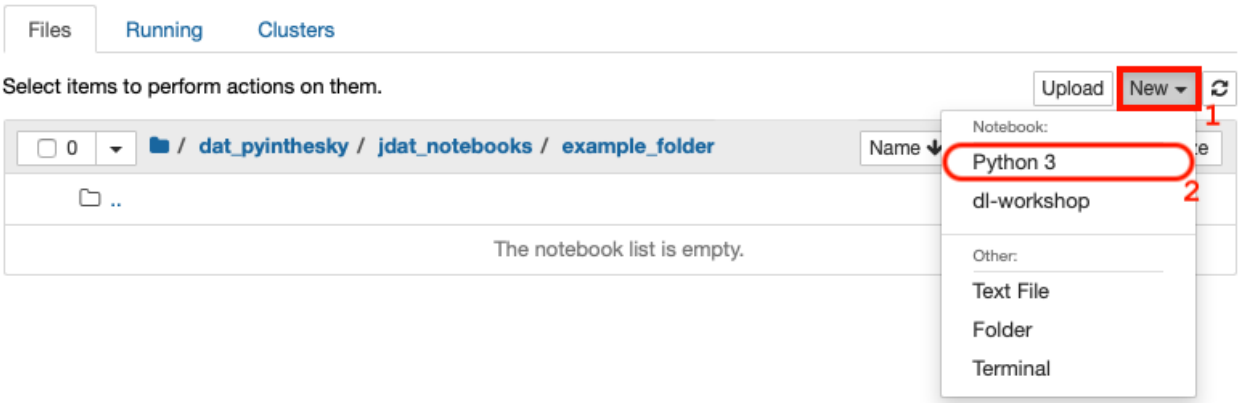
Important: Please note that all JDAT related code is written `Python 3`; `Python 2` is not supported.

4.4.2 Starting a Notebook

In your terminal, `cd` into the directory you would like to work in and run `jupyter notebook`. If a web browser does not pop up, use the URL in the terminal output to open the Jupyter home page.

4.4.3 Creating a New Notebook

To create a new notebook, navigate to the directory you would like to work in and select the `Python 3` option under the `new` menu item:



4.4.4 Markdown

"Markdowns" are used to add supporting literature to notebooks. To convert a cell to markdown, select it with your courser and select markdown in the cell type drop down menu.

Once the cell converts to a markdown cell, you can enter your text and equations. If you have links to an image, you can embed them and they will be rendered when you run the cell. **After you are done editing, make sure to run the cell (Shift + Enter) to render the markdown**

See also:

See Adam Pritchard's markdown cheatsheet (https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet) for tips and instructions.

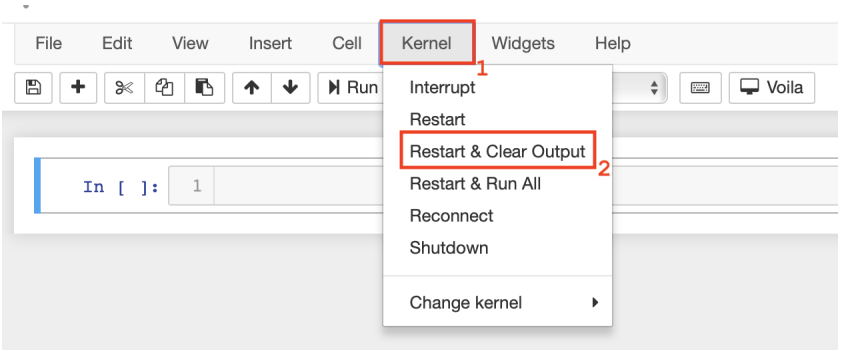
4.4.5 Developer Notes

Developer notes are used to leave notes or feedback for the JDAT developers. A developer note should be a part of the notebook itself and should be a single *markdown* cell. That cell should begin with the text `*Developer Note:*`. For example:

```
*Developer Note:*
Creating the spectrum above is a bit complicated, and it would improve the workflow␣
↪if there was a single
simple function that just did `spec = simulate_jwst_spectrum(a, b)`.
```

4.4.6 Clearing Notebook Outputs

All cell outputs of the notebook should be cleared before opening or updating a pull request. This is because fully rendered notebooks with figures can take up large amounts of storage space. Please make sure your results are reproducible (you do not need the outputs) because clearing the notebook is not revisable. To clear notebook cell outputs, please click `Kernel` in the Jupyter menu and select `Restart & Clear Output`. After the notebook restarts, make sure to save the notebook before closing and submitting it.



4.4.7 Multiple Notebooks

If you have multiple notebooks that need to run in a specific sequence, please name the notebooks by prepending a number before each notebook title according to the sequence (up to 99 notebooks allowed). For example:

```
dat_pyinthesky
└── jdat_notebooks
    └── example_folder
        ├── 01_generate_simulated_data.ipynb
        ├── 02_run_calibration_pipeline.ipynb
        ├── 03_data_analysis.ipynb
        └── requirements.txt
```

4.4.8 Pep-8 Guideline

Please see STScI's Python Guideline (https://github.com/spacetelescope/style-guides/blob/master/guides/python.md) and the official Pep-8 Guideline (https://www.python.org/dev/peps/pep-0008/) for more information.

See also:

- STScI notebook style guide (https://github.com/spacetelescope/style-guides/blob/master/guides/jupyter-notebooks.md)
- STScI Python style guide (https://github.com/spacetelescope/style-guides/blob/master/guides/python.md)

4.5 Requirements File

A pip requirements file is a text file, named `requirements.txt`, that contains a list of Python packages needed to run science notebooks. When users download a science notebook, they need this requirements file to install the necessary Python packages. Therefore, Notebook Leads will need to provide a requirements file in addition to their notebooks.

Listing Packages: Each Python package should be listed in a new line in the requirements file. You should only list packages imported into your notebook (used in your notebook). You can make a list of packages by looking though import statements in your notebook. Packages that are native to Python, such as `os` or `math`, should **not** be listed in the requirements file.

Tip: You can get a list of package versions by running `conda list` or `pip list` in your terminal.

To specify a version of the package, you can use the `==` operator (for example `astropy==4.1`). Please list all requirement versions to match the conda or pip environment you used to develop your notebooks. If you need to add a Python package only available on GitHub, you can list the module as follows:

```
# Package on GitHub:
git+https://github.com/name_of_repo

# If you need to specify a branch:
git+https://github.com/name_of_repo#branch_name
```

Example: Here is an example requirements file (`requirements.txt`):

```
numpy==1.19.1
jupyter==1.0.0
aplpy==2.0.3
astrodendro==0.2.0
astropy==4.0.1.post1
matplotlib==3.3.1
photutils==0.7.2
scipy==1.5.0
specutils==1.0
git+https://github.com/spacetelescope/jwst#0.16.2
```

Tip: To install packages in a `requirements.txt` file, use `pip install -r requirements.txt`

Warning: Before installing a new set of packages from a requirements file, one should consider creating a new Conda environment. If you need to setup Conda, please see the Conda's Getting Started (https://docs.conda.io/projects/conda/en/latest/user-guide/getting-started.html) documentation.

Sometimes, the dev team adds an extra requirement file named `pre-requirements.txt`. This file is used for the testing infrastructure and should be installed before the `requirements.txt`. The notebook lead is not expected to contribute the `pre-requirements.txt` file.

4.6 Useful Links

- Jdaviz documentation (https://jdaviz.readthedocs.io/en/latest/)
- Data Analysis Tools JDox Page (https://jwst-docs.stsci.edu/jwst-post-pipeline-data-analysis)
- Technical details about the GitHub notebook stages and process (https://github.com/spacetelescope/dat_pyinthesky/tree/main/jdat_Procedure-for-JDAT-Notebooks)
- Astropy core libraries (https://docs.astropy.org/en/stable)
- Astropy affiliated packages (https://www.astropy.org/affiliated/index.html)
- Visualization tools (https://github.com/spacetelescope/jdaviz)
- Existing notebooks (https://github.com/spacetelescope/dat_pyinthesky/tree/main/jdat_notebooks)
- Pep 8 coding style guide (https://www.python.org/dev/peps/pep-0008/)
- Notebook Style Guide (https://github.com/spacetelescope/style-guides/blob/master/guides/jupyter-notebooks.md)

4.7 Developers and Staff

4.7.1 STScI Notebook Leads

Important:

- **Sign Up:** Talk to Camilla Pacifici and Ori Fox to find a use case that suits your scientific and technical interests.
- Charge your timecard: **P0004.06.05.09**

Notebook Draft

Start writing your notebook and think through the following. Contact Cami or Ori along the way to ask questions. We don't expect you to perfectly integrate all tools on your own.

- **Manage your time**. Consider you will likely need to spend 50% of your time writing your notebook draft, and 50% in the Sprint and Review stage. A realistic rough estimate is 30 hours total, so 15 hours at each phase.
- Download the notebook template and make sure to document your work as much as possible. It is important to communicate the purpose of your notebook and where you obtained your data, as well as the algorithms you use to do the science.
  - Where did you get your data?
  - What is the scientific purpose of the notebook?
  - Which tools/packages do you use?
- We prefer you use JWST simulated data, but we realize this is not always possible. Please communicate with Cami and Ori in advance about where you will obtain your data.
- Does my desired functionality exist in astropy and, specifically, specutils, photutils, and the visualization tools?
  - If so, am I using it? Am I using it correctly?
  - If not, what needs to be done to enable it? Make a developer note inside your cell.

- Make sure parts of your code are devoted to science validation.
- Break the notebook down into many, focused cells. Follow the notebook style guide.
- Consider making your science as generalized as possible.
- If you get stuck, contact anyone above (step e) or use #jdadf_dev channel on slack.
- Upload your notebook on dat_pyinthesky as soon as you feel comfortable doing so.

Join a Sprint

When you have completed your notebook draft, you are ready to join a developer sprint.

- Contact either Cami or Ori to sign up for a 2-week long sprint when you can devote time to developing your notebook. You will be invited to a number of different meetings throughout a 2 week period.
- Attend an Orientation Meeting the week before your Sprint is set to begin, including a live technical review to understand the process.
- Prepare by bringing a list of developer notes and/or personal notes where you think your notebook could be improved with new functionality built into the tools.
- Attend the kick-off meeting (usually Monday 2pm) where you will meet the team, learn about the Product Owners (POs), and be assigned a scientific reviewer and a technical reviewer (this part can also happen offline).
- Schedule a meeting with your technical reviewer.mFocus on utilizing the correct libraries and functionality. Learn to turn your notes into Jira tickets and GitHub issues, as well as the process for having those tickets implemented and integrated back into your notebooks.
- Attend a couple tag-ups each week, which happen every morning at 10.30am via bluejeans; this will provide you with an opportunity to focus on development of individual cells within a notebook
- Attend the Viz Tool hack hour (typically the second Monday of the sprint at 11am) to focus on notebook workflow using the Viz Tools
- If you get stuck, contact anyone above (step e) or use #jdadf_dev channel on slack.
- Make sure parts of your code are devoted to science validation.
- Attend the Review meeting (usually at the end of the sprint on Monday at 1:30pm) and show your notebook (this is also an opportunity to give feedback on the process).

4.7.2 Technical Review

Note: This page is under construction

Instructions for Technical Reviewers

The reviewer will need to:

- Verify that the language satisfies PEP 8.
- If not, suggest the appropriate changes.
- Check that the necessary data are stored as requested in the guidelines.

- Verify that the currently available core-libraries functionalities and visualization tools are used when needed. If not (i.e., Author has written their own functionalities), suggest new code that calls the already available libraries. Address the 'Developer Notes'.

4.7.3 Science Review

Note: This page is under construction

Instructions for Science Reviewers

The reviewer will be provided the link to the PR. They will need to check that:

- The notebook shows a scientific result.
- The modeling used is sound.
- The plotting is clear.
- The comments are sufficient and clear.