

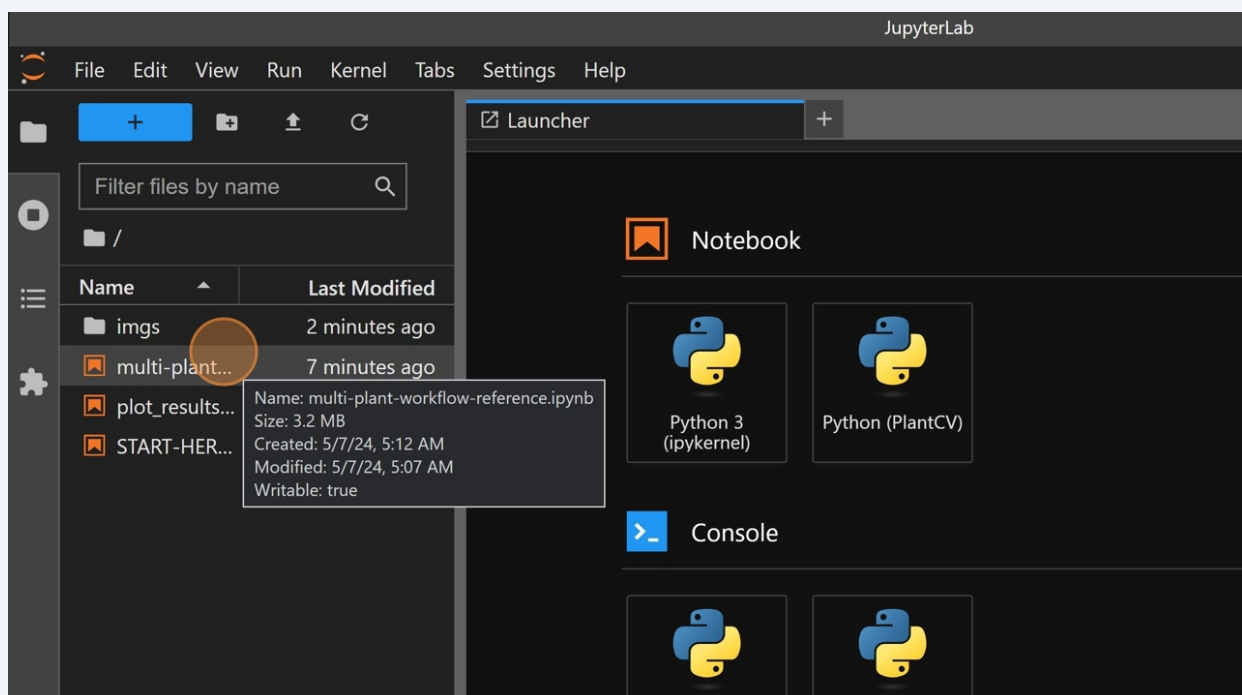
# How to Parallelize Your Workflows Using PlantCV

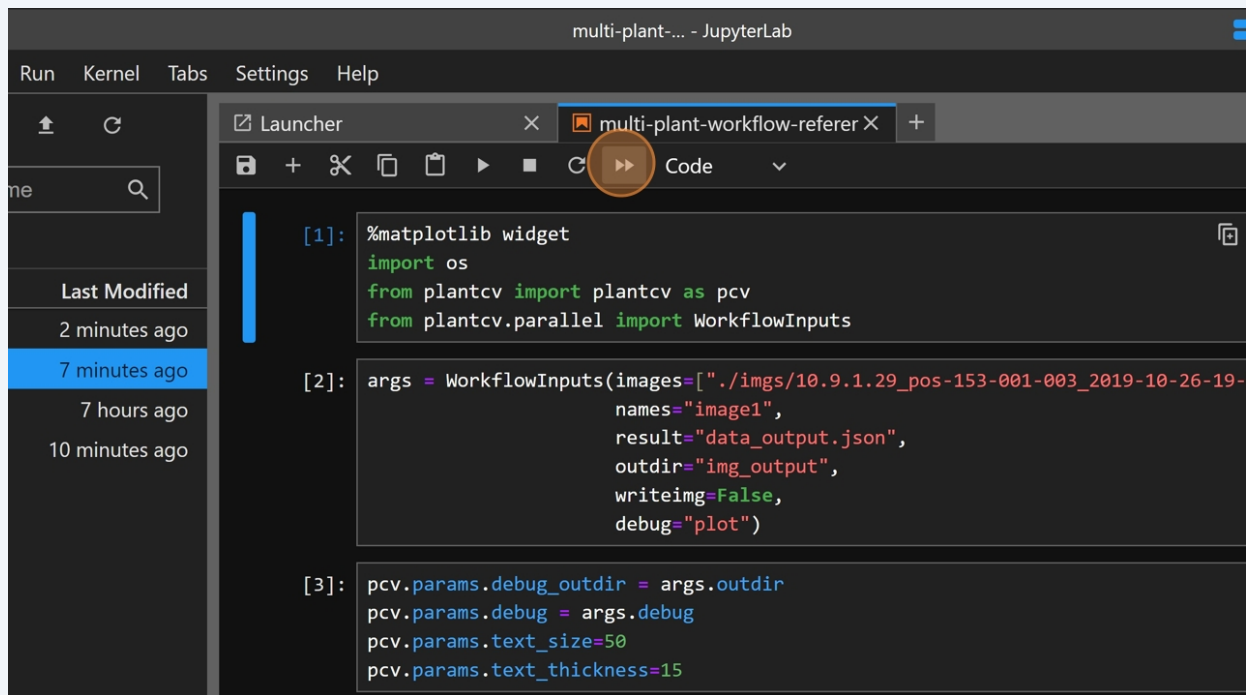
This guide provides step-by-step instructions on how to parallelize workflows using PlantCV. By following these steps, users can optimize their workflow efficiency and increase productivity.

1

Often you will start with a workflow from another tutorial that you will have then edited to work well for your experiment. Here we assume that you start with an .ipynb workflow that runs without errors on several representative test images.

NOTE: Make sure you have ran your workflow and have produced a JSON output file.

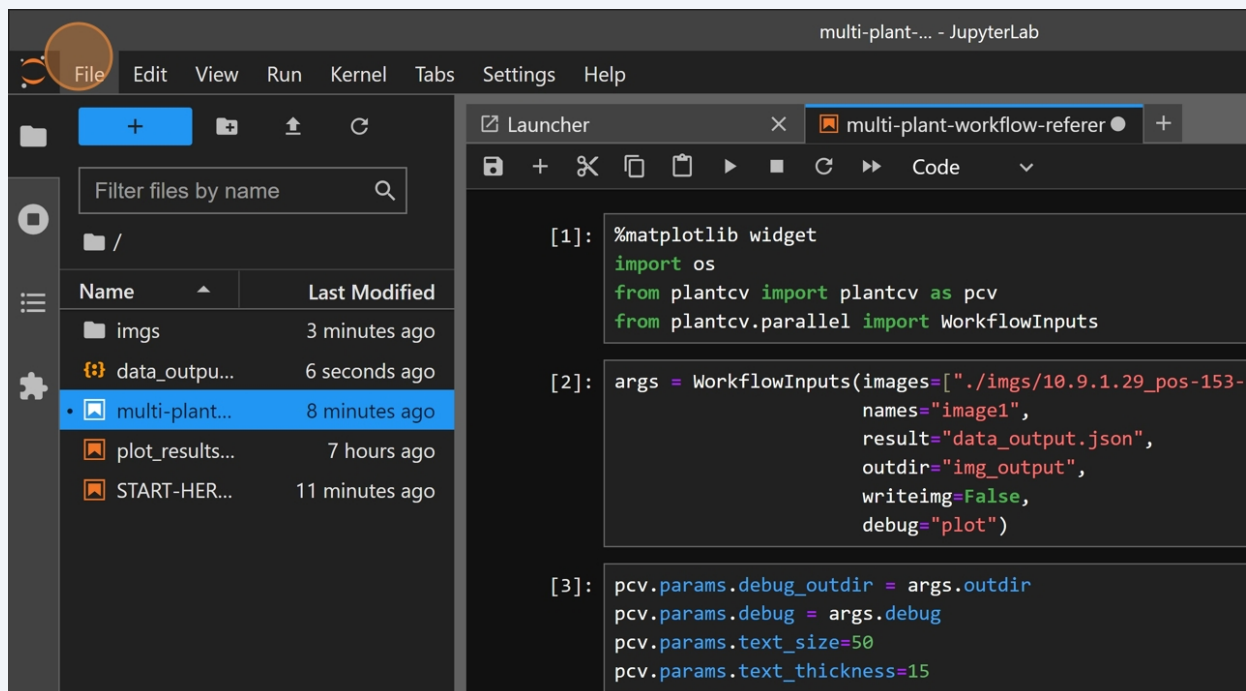


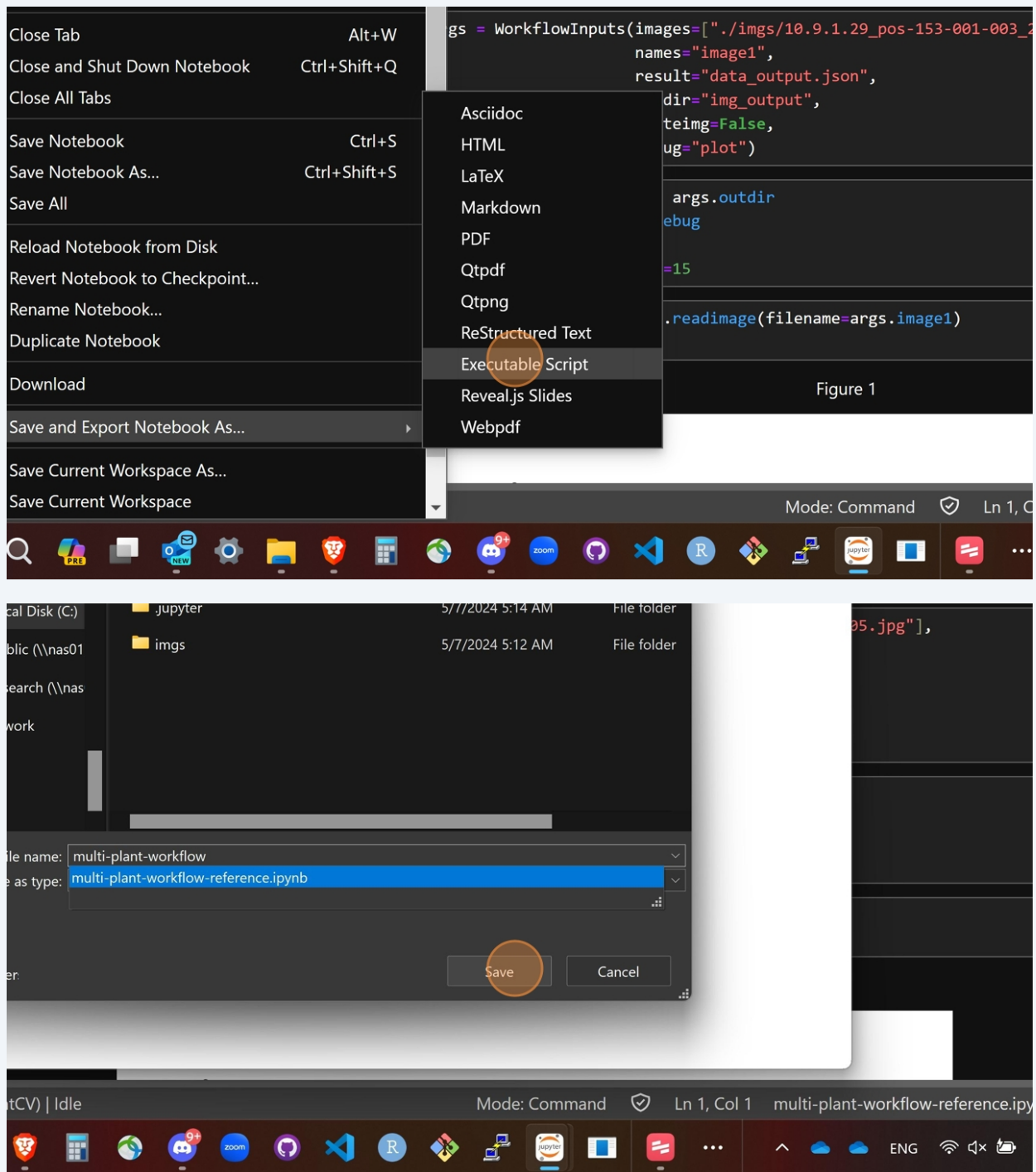


2

Convert your Jupyter Notebook (.ipynb) to an executable script (a Python script, .py) selecting **File > Save and Export Notebook As... > Executable Script**

Save your Python script in the same directory as your Jupyter notebook for easy access. You may need to adjust the title of your script so it is concise (in the example, I am using a reference document and do not want **"reference"** in my final script.



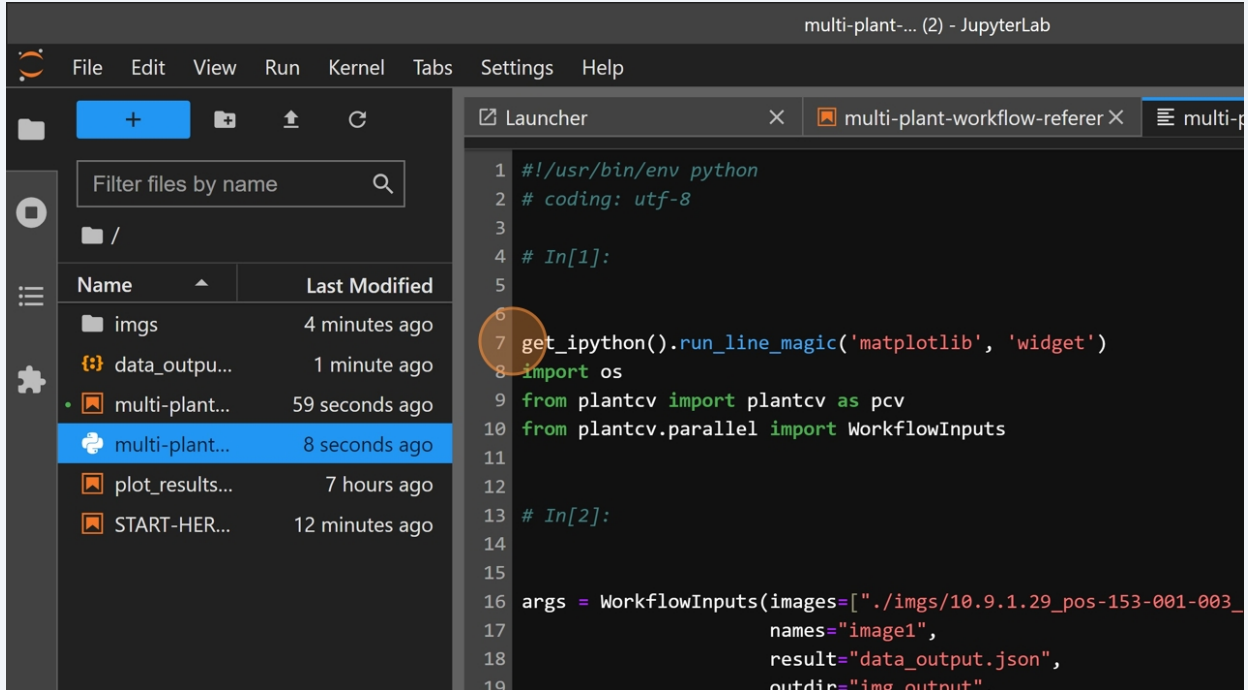


3

Open your new Python script. We will have to make some edits to make it a functioning workflow script.

The first thing we will want to do is "comment out" things we do not want to run in parallel. In the example image this would mean adding an "#" ahead of line 7:  
**get\_ipython().run\_line\_magic('matplotlib', 'widget')**

For your workflow you may also want to comment out commands that only print images or that look for google colab environments.

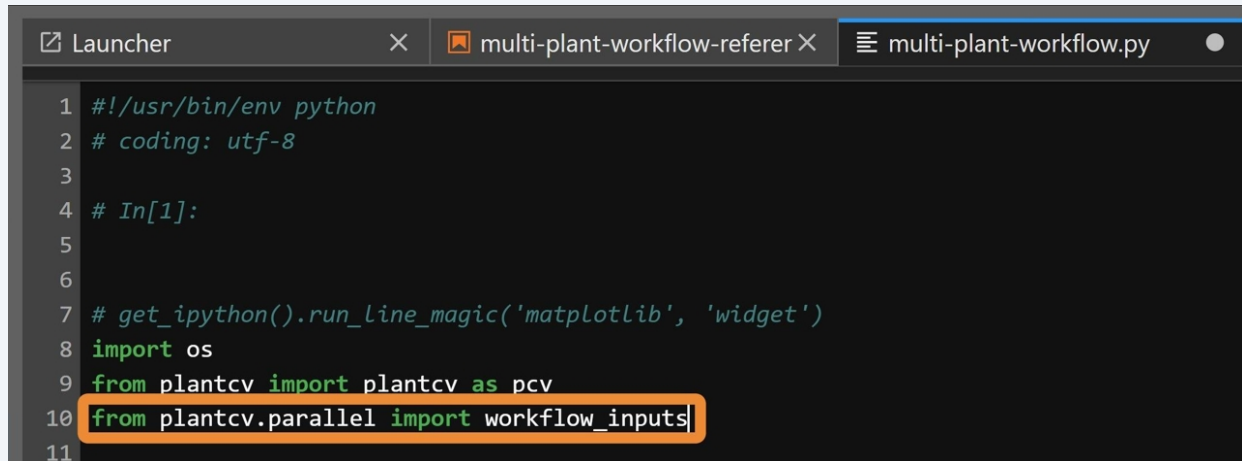


```
1 #!/usr/bin/env python
2 # coding: utf-8
3
4 # In[1]:
5
6
7 get_ipython().run_line_magic('matplotlib', 'widget')
8 import os
9 from plantcv import plantcv as pcv
10 from plantcv.parallel import WorkflowInputs
11
12
13 # In[2]:
14
15
16 args = WorkflowInputs(images=["./imgs/10.9.1.29_pos-153-001-003_
17                             names="image1",
18                             result="data_output.json",
19                             outdir="img_output",
```

- 4 We need to make a change to our import statements to say:

from plantcv.parallel import workflow\_inputs in place of from plantcv.parallel import WorkflowInputs

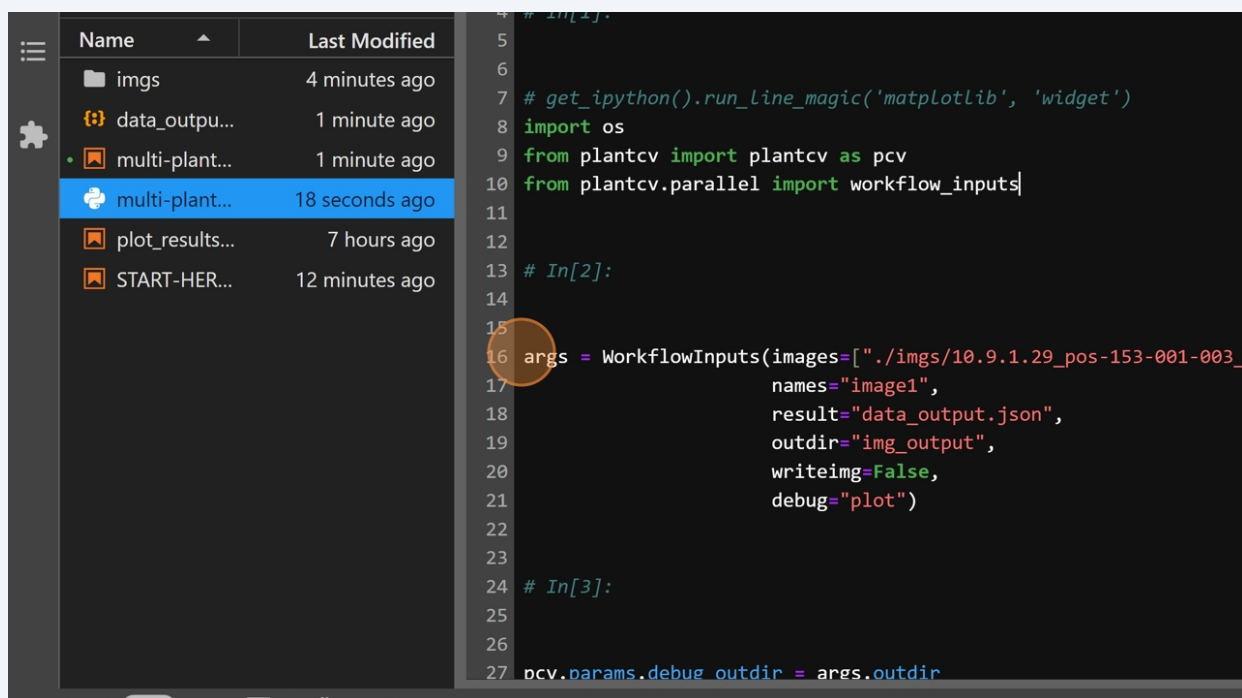
This will tell the parallel process to use workflow\_inputs() to set parameters based on the configuration JSON file instead of using the hard coded arguments from the notebook.



```
1 #!/usr/bin/env python
2 # coding: utf-8
3
4 # In[1]:
5
6
7 # get_ipython().run_line_magic('matplotlib', 'widget')
8 import os
9 from plantcv import plantcv as pcv
10 from plantcv.parallel import workflow_inputs
11
```

- 5 Next, we will change our argument definitions for this workflow by "commenting out" the our **WorkflowInputs** parameters. Remember that you can comment out an entire section by using **CTRL or Cmd + /** in jupyter.

See the image below:



Name	Last Modified
imgs	4 minutes ago
data_outpu...	1 minute ago
multi-plant...	1 minute ago
multi-plant...	18 seconds ago
plot_results...	7 hours ago
START-HER...	12 minutes ago

```
4 # In[1]:
5
6
7 # get_ipython().run_line_magic('matplotlib', 'widget')
8 import os
9 from plantcv import plantcv as pcv
10 from plantcv.parallel import workflow_inputs
11
12
13 # In[2]:
14
15
16 args = WorkflowInputs(images=["./imgs/10.9.1.29_pos-153-001-003_
17                             names="image1",
18                             result="data_output.json",
19                             outdir="img_output",
20                             writeimg=False,
21                             debug="plot")
22
23
24 # In[3]:
25
26
27 pcv.params.debug_outdir = args.outdir
```

Name	Last Modified
imgs	4 minutes ago
data_output...	1 minute ago
multi-plant...	1 minute ago
multi-plant...	18 seconds ago
plot_results...	7 hours ago
START-HER...	12 minutes ago

```
4 # In[1]:
5
6
7 # get_ipython().run_line_magic('matplotlib', 'widget')
8 import os
9 from plantcv import plantcv as pcv
10 from plantcv.parallel import workflow_inputs
11
12
13 # In[2]:
14
15 |
16 # args = WorkflowInputs(images=["./imgs/10.9.1.29_pos-153-001-00
17 #
18 #     names="image1",
19 #     result="data_output.json",
20 #     outdir="img_output",
21 #     writeimg=False,
22 #     debug="plot")
23
24 # In[3]:
25
26
27 pcv.params.debug_outdir = args.outdir
```

- 6 Above the commented out **WorkflowInputs**, we are going to add a new line that will store workflow inputs into **args** to support parallel workflow execution:

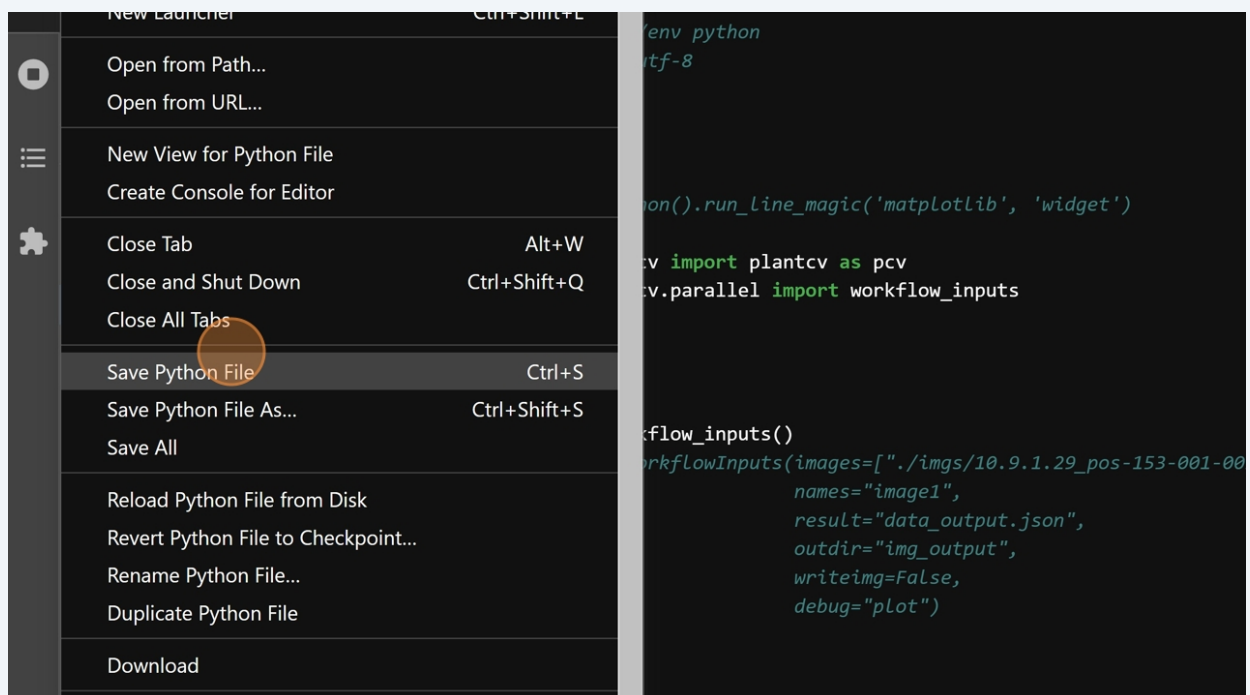
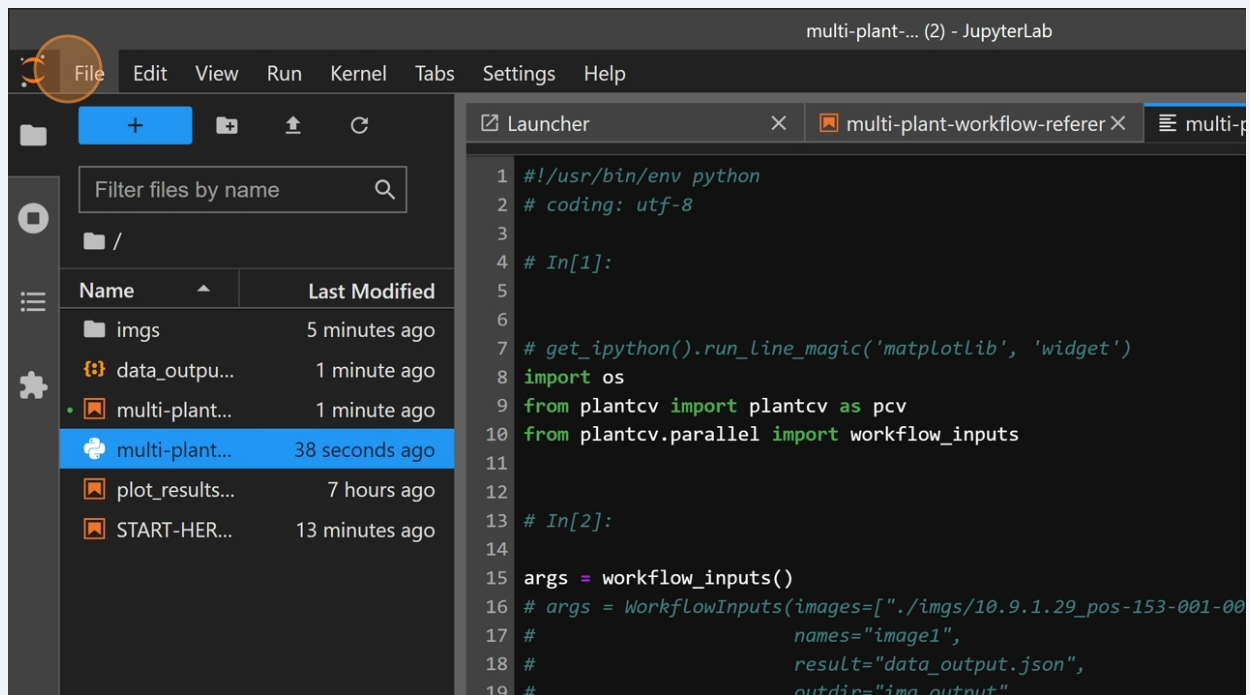
`args = workflow_inputs()`

Last Modified
4 minutes ago
1 minute ago
1 minute ago
28 seconds ago
7 hours ago
12 minutes ago

```
4 # In[1]:
5
6
7 # get_ipython().run_line_magic('matplotlib', 'widget')
8 import os
9 from plantcv import plantcv as pcv
10 from plantcv.parallel import workflow_inputs
11
12
13 # In[2]:
14
15 args = workflow_inputs()
16 # args = WorkflowInputs(images=["./imgs/10.9.1.29_pos-153-001-003_2019-10-26-19-05.
17 #
18 #     names="image1",
19 #     result="data_output.json",
20 #     outdir="img_output",
21 #     writeimg=False,
22 #     debug="plot")
23
24 # In[3]:
25
26
27 pcv.params.debug_outdir = args.outdir
```

- 7 Save the changes you made to your Python script.





In the next step we will have to change the working directory in terminal to the location where our scripts are. There are a couple of ways to obtain the file path for your working directory:

1. In Jupyter if you hover over the folder icon in the file browser it will show the working directory.
2. In Jupyter you can open a terminal and run `pwd` to print the working directory.

**NOTE:** You may want to copy the file path to a document so you can see how much of the file path is copied (the full file path will not be copied).

8

In this guide, the current working directory is local to the root directory, so only a portion of the path was needed to point our conda environment to the current working directory. Remember that to change directories, you need to use the operator `cd` (See image below, outlined in green).

Once you have changed your working directory in conda, type `dir` or `ls` to view the contents of the directory. Make sure that your Python script is in the correct location. (See image below, outlined in orange).

```
(plantcv) C:\Users\pbhatt\Documents>cd .\Workshop\Multi-Plant-Parallelization-Tutorial

(plantcv) C:\Users\pbhatt\Documents\Workshop\Multi-Plant-Parallelization-Tutorial>dir
Volume in drive C has no label.
Volume Serial Number is 769C-CE5C

Directory of C:\Users\pbhatt\Documents\Workshop\Multi-Plant-Parallelization-Tutorial

05/07/2024  05:17 AM    <DIR>          .
05/07/2024  05:12 AM    <DIR>          ..
05/07/2024  05:17 AM    <DIR>          .ipynb_checkpoints
05/07/2024  05:14 AM    <DIR>          .jupyter
05/07/2024  05:16 AM             69,252 data_output.json
05/07/2024  05:12 AM    <DIR>          imgs
05/07/2024  05:16 AM      3,317,542 multi-plant-workflow-reference.ipynb
05/07/2024  05:17 AM      1,822 multi-plant-workflow.py
05/06/2024  10:03 PM      1,834 plot_results.ipynb
05/07/2024  05:04 AM      4,501 START-HERE_multi-plant-workflow.ipynb
               5 File(s)      3,394,951 bytes
               5 Dir(s)  184,838,848,512 bytes free
```

9

Now we need to create our parallel configuration file by typing:

```
plantcv-run-workflow --template config.json
```

Since we have changed our working directory to our current folder, you should see **config.json** appear in the file explorer like shown below.

```
Anaconda Prompt (miniconda3)

(plantcv) C:\Users\pbhatt\Documents\Workshop\Multi-Plant-Parallelization-Tutorial>plantcv-run-workflow --template config.json
```

10

Since we have changed our working directory to our current folder, you should see **config.json** appear in the file explorer like shown below.

We will need to edit **config.json** so that we can configure the parallel analysis. Right-click **config.json** and hover over **Open With** and select **Editor** to make changes to the file.





In Editor, make changes to **config.json**, common things that should be changed are listed below but the full set of options is in the [documentation](#) and is worth reviewing.

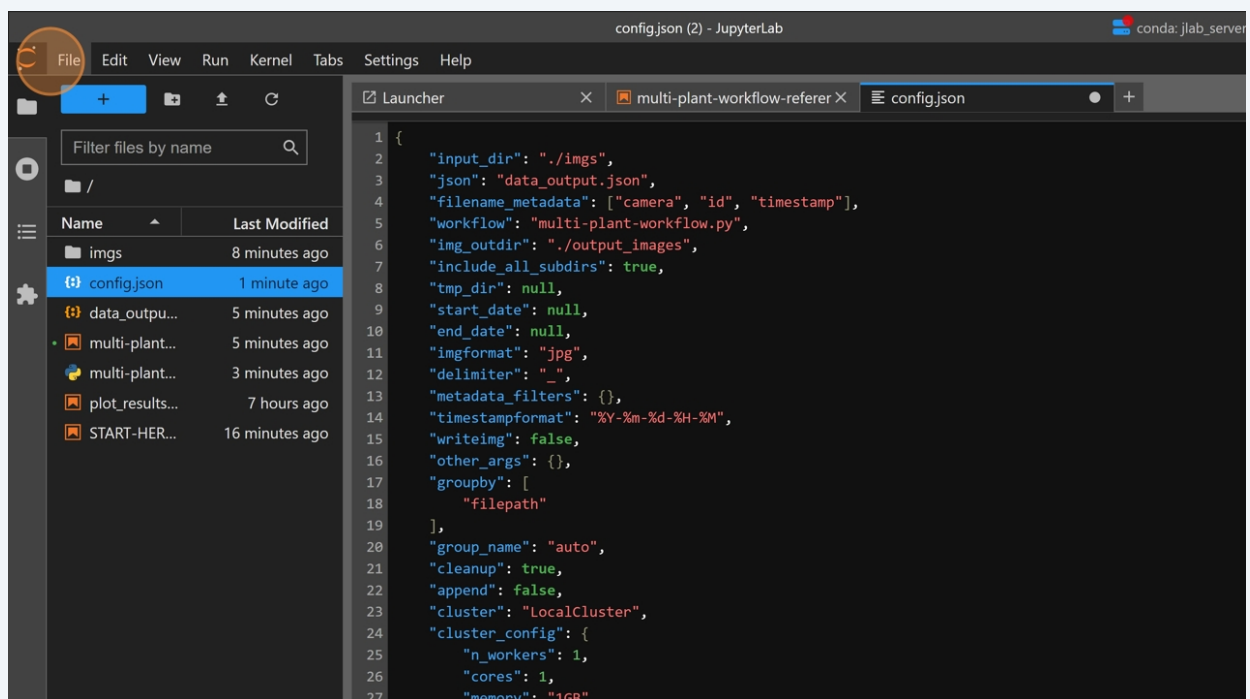
- "input\_dir": "./imgs" *[Put file path/name of input directory for images you want analyzed]*
- "json": "data\_output.json" *[Put the path/name of the data output file (located in the args container under results within your workflow)]*
- "filename\_metadata": ["camera", "id", "timestamp"] *[list of metadata terms to collect. Supported metadata terms include: camera, imgtype, zoom, exposure, gain, frame, lifter, timestamp, id, plantbarcode, treatment, cartag, measurementlabel, and other]*
- "workflow": "multi-plant-workflow.py" *[path/name of user-defined (your) PlantCV workflow Python script]*
- "img\_outdir": "./output\_images" *[path/name of output directory where measured images will be stored. Default is "./output\_images"]*
- "imgformat": "jpg" *[image file format/extension. Default is "png"]*
- "timestampformat": "%Y-%m-%d-%H-%M" *[date format as observed in your naming scheme. For explanation what each of the symbols mean, see the python [time format documentation](#)]*
- "append": false *[(bool, default = False): if **True** will append results to an existing json file. If **False**, will delete previous results stored in the specified JSON file.]*
- "cluster": "LocalCluster" *[There are several cluster types, the default option is "LocalCluster" which will run in parallel on the machine you run the run workflow command from. The complete list of options is: "**LocalCluster**", "**HTCondorCluster**", "**LSFCluster**", "**MoabCluster**", "**OARCluster**", "**PBSCluster**", "**SGECluster**", and "**SLURMCluster**" which can be read about in the [dask docs](#).]*
- cluster\_config:
  - n\_workers: In the example below this is still 1, but you will increase this based on how many cores you have available/want to use. This controls the number of workers to run in parallel. The "cores" argument is how many cores each worker needs, which will almost always stay as 1.

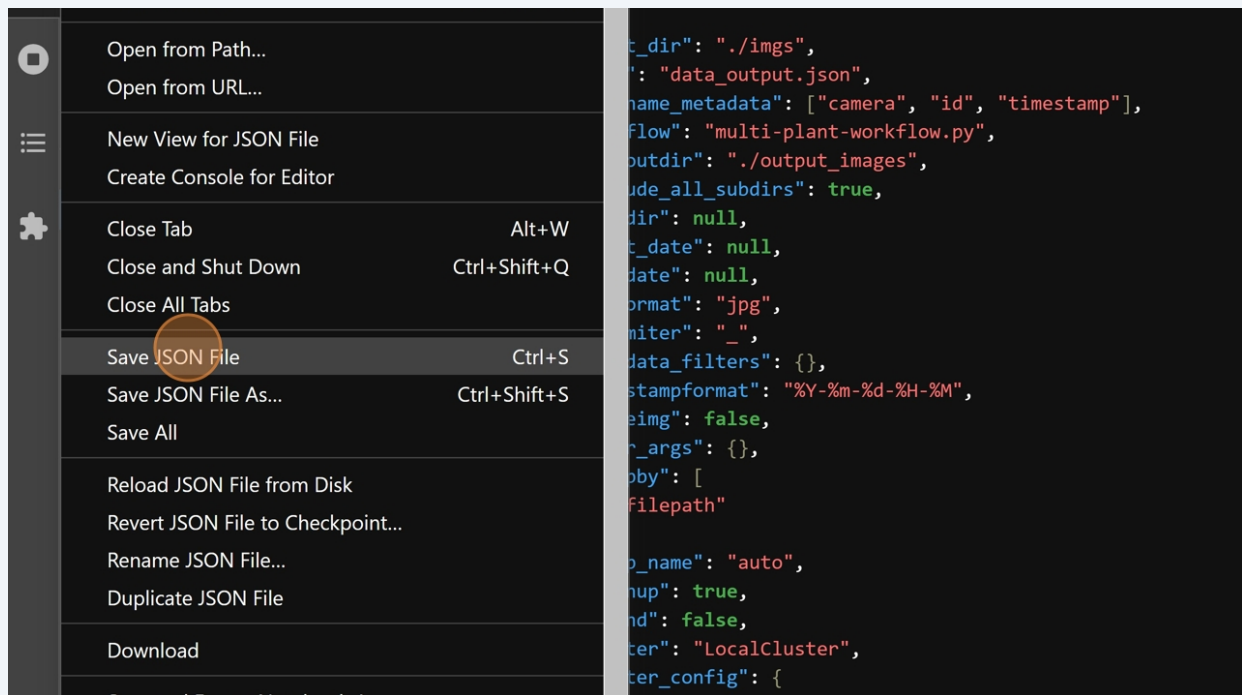
```

1 {
2   "input_dir": "./imgs",
3   "json": "data_output.json",
4   "filename_metadata": ["camera", "id", "timestamp"],
5   "workflow": "multi-plant-workflow.py",
6   "img_outdir": "./output_images",
7   "include_all_subdirs": true,
8   "tmp_dir": null,
9   "start_date": null,
10  "end_date": null,
11  "imgformat": "jpg",
12  "delimiter": "_",
13  "metadata_filters": {},
14  "timestampformat": "%Y-%m-%d-%H-%M",
15  "writeimg": false,
16  "other_args": {},
17  "groupby": [
18    "filepath"
19  ],
20  "group_name": "auto",
21  "cleanup": true,

```

## 12 Save the changes you have made to the **config.json**





13

Now that we have made the necessary changes to our parallel configuration file, it is time for us to run our workflow. To execute your parallel analysis, return to your terminal and type `plantcv-run-workflow --config config.json` into the prompt.

```
(plantcv) C:\Users\pbhatt\Documents\Workshop\Multi-Plant-Parallelization-Tutorial>plantcv-run-workflow --config config.json
```



If you successfully set up your **config.json** then you should see a number of files found and a progress bar on your screen with how long it will take to analyze your dataset. You will also see that your job list will include X workflows.

If you did not set up your **config.json** then you will receive error messages that detail where PlantCV is having issues finding an image directory, your workflow, incorrect date formats, etc.

After the job has completed, PlantCV will automatically convert your JSON file to CSV.

```
(plantcv) C:\Users\pbhatt\Documents\Workshop\Multi-Plant-Parallelization-Tutorial>plantcv-run-workflow --config config.json
Starting run 2024-05-07_05-22-07

Reading image metadata...
Reading image metadata took 0.003996610641479492 seconds.
Building job list...
Task list includes 14 workflows
Building job list took 0.11158013343811035 seconds.
Processing images...
[#####] | 0% Completed | 19.2sWarning: Shrinking radius to make ROIs fit in the image
[#####] | 57% Completed | 1min 30.3sWarning: Shrinking radius to make ROIs fit in the image
Processing images took 154.34734082221985 seconds.Completed | 2min 29.4s[2K
Processing results...
Processing results took 0.3885014057159424 seconds.

(plantcv) C:\Users\pbhatt\Documents\Workshop\Multi-Plant-Parallelization-Tutorial>
```

14 You should see two CSV files appear in your directory:

- **plantcv-results-single-value-traits.csv**
- **plantcv-results-multi-value-traits.csv**

The **single-value-traits.csv** file will be in wide format, with a column per trait, whereas the **multi-value-traits.csv** file will be in long format, with one row per value/label. The hierarchical organization of these files enable more efficient data processing downstream.

