Colaboratory + Drive + Github -> the workflow made simpler

Oleg Żero Follow
Oct 29, 2019 · 5 min read ★

Introduction

This post is a continuation of our earlier attempt to make the best of the two worlds, namely Google Colab and Github. In short, we tried to map the usage of these tools in a typical data science workflow. Although we got it to work, the process had its drawbacks:

- It relied on *dynamic imports*, which made our code unnecessarily cumbersome.
- We didn't quite get the Github part to work. The workspace had to be saved offline.

In this post, we will show you a simpler way to organize the workspace without these flaws. All you will need to proceed is a Gmail and Github account. Let's get to work.

What goes where?

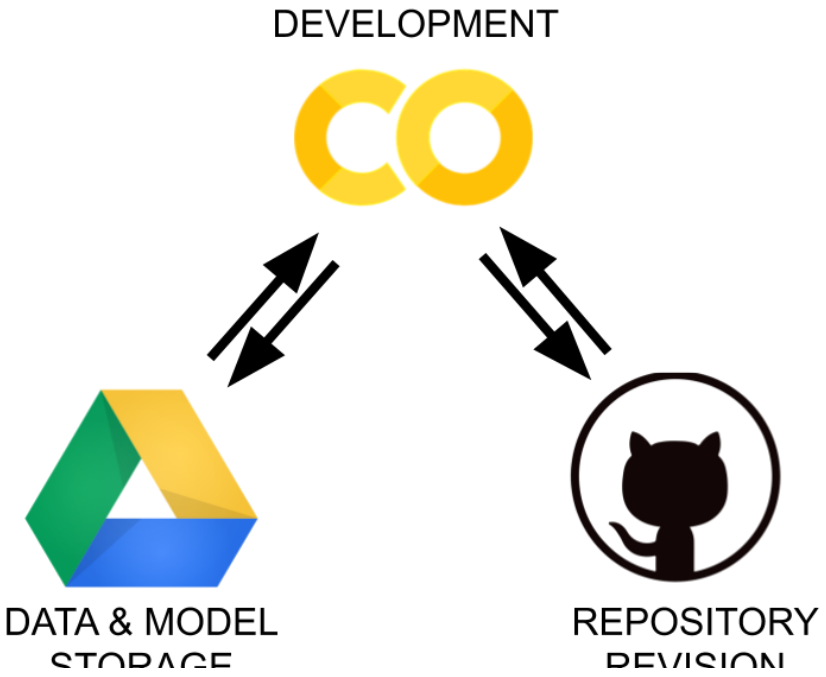

Figure 1. Three parts of our simple "ecosystem".

Typically, we have four basic categories of files in our workspace:

- notebooks *(.ipynb)* — for interactive development work,
- libraries *(.py)* — for code that we use and reuse,
- models — things we try to build,
- data — ingredients we build it from.

Since Colab backend is *not persistent*, we need a permanent storage solution. In addition to that, we also need a version control system so we can keep track of changes. Finally, we would appreciate if we won't have to think of this machinery any more than necessary.

Colab integrates easily with Google Drive, which makes it a natural choice for *storage* space. We will use it for storing our *data* and *models*. At the same time, Github is better suited for code, thus we will use it for *notebooks* and *libraries*. Now, the question arises, how we can interface the two from the position of our notebook, which will make our workflow as painless as possible?

Github

We assume that you already have a Github account and created a repository for your project. Unless your repository is *public*, you will need to *generate a **token*** to interact with it through a command line. Here is a short guide on how to create one.

Google Drive

Next thing is to organize our non-volatile storage space for both models and data. If you have a Gmail account you are halfway there. All you need to do is to create an empty directory in the Drive and that's it.

Colaboratory — operational notebook

To keep things organized, we define one separate notebook that is to be our *operational* tool. We will use its cells exclusively for manipulating of our space, letting the other notebooks take care of more interesting things such as *exploratory data analysis*, *feature engineering* or *training*. All notebooks, including this one, will be revisioned, but with command stored in the *operational* notebook.

The workflow

The workflow is a simple **three-step process**:

1. First, after connecting to the Colab runtime, we need to *mount* Google Drive and *update* our space using Github.

2. We work with the notebooks and the rest of the files (our modules, libraries, etc.). In this context, we simply call it *editing*.

3. We *save* our work, by synchronizing our Drive with Github using the *operational* notebook.

Connecting, mounting and updating

```
from google.colab import drive
from os.path import join

ROOT = '/content/drive'     # default for the drive
PROJ = 'My Drive/...'       # path to your project on Drive

GIT_USERNAME = "OlegZero13" # replace with yours
GIT_TOKEN = "XXX"           # definitely replace with yours
GIT_REPOSITORY = "yyy"      # ...nah


drive.mount(ROOT)           # we mount the drive at /content/drive

PROJECT_PATH = join(ROOT, PROJ)
!mkdir "{PROJECT_PATH}"I    # in case we haven't created it already

GIT_PATH =
"https://{GIT_TOKEN}@github.com/{GIT_USERNAME}/{GIT_REPOSITORY}.git"
!mkdir ./temp
!git clone "{GIT_PATH}"
!mv ./temp/* "{PROJECT_PATH}"
!rm -rf ./temp
!rsync -aP --exclude=data/ "{PROJECT_PATH}"/*  ./
```

The above snippet mounts the Google Drive at `/content/drive` and creates our project's directory. It then pulls all the files from Github and copies them over to that directory. Finally, it collects everything that belongs to the Drive directory and copies it over to our local runtime.

A nice thing about this solution is that it won't crash if executed multiple times. Whenever executed, it will only *update* what is new and that's it. Also, with `rsync` we have the option to *exclude* some of the content, which may take too long to copy (...data?).

Editing, editing, and editing

Development, especially in data science, means trying multiple times before we finally get things right. At this stage, editing to the external files/libraries can be done by:

1. substituting or changing files on Drive and then transferring them to the local runtime of each notebook using `rsync`, or

2. using the so-called IPython magic commands.

Suppose you want to quickly change `somefile.py`, which is one of your library files. You can write the code for that file and tell Colab to save it using `%%writefile` command. Since the file resides locally, you can use simply the `import` statement to load its new content again. The only thing is to remember to execute `%reload_ext somefile` command first, to ensure that Colab knows of the update.

Here is an example:

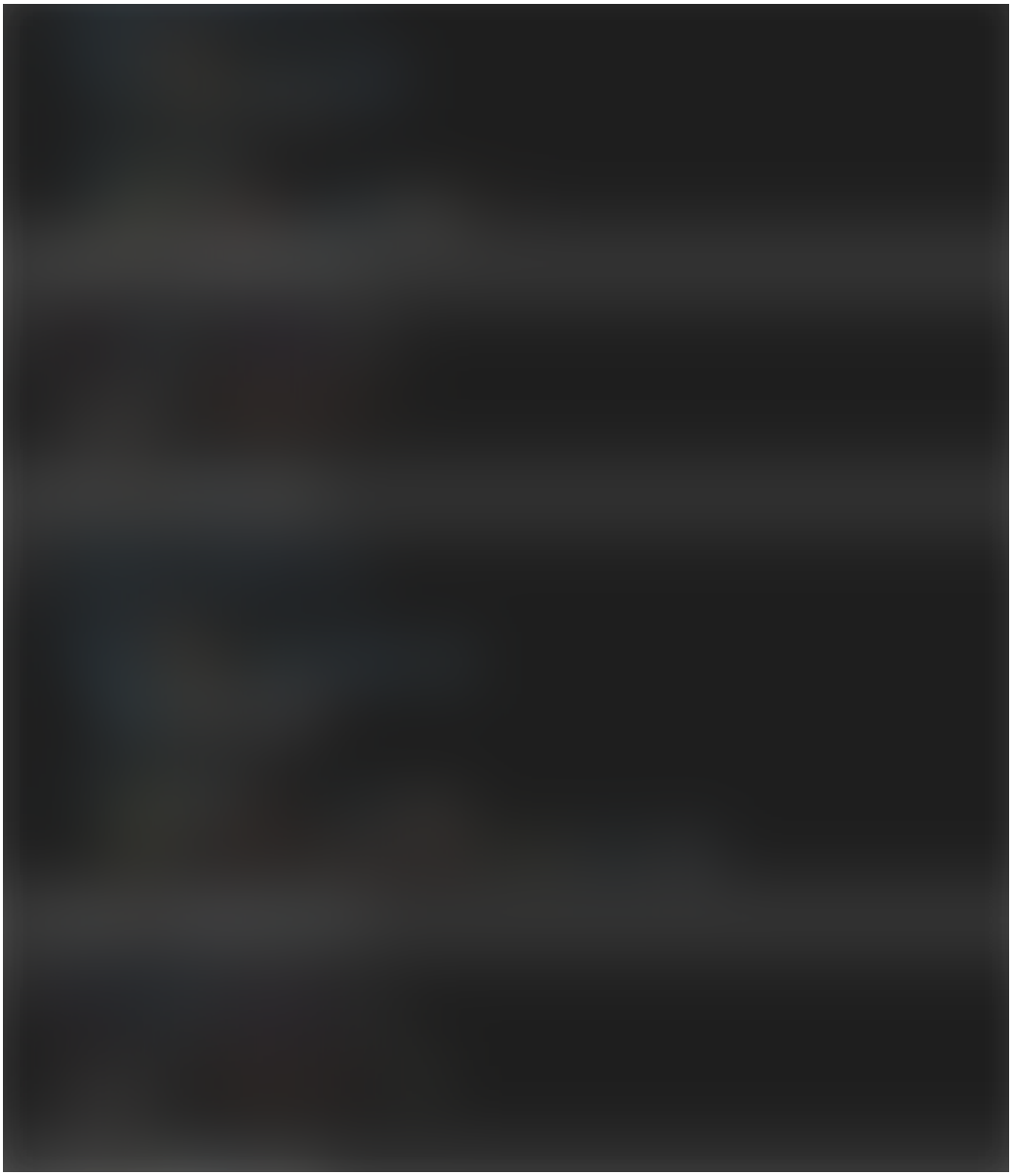

Figure 2. Importing, editing and importing again. All done through the cells.

Saving, calling it a day

Once you wish to make a backup of all of your work, all you need to do is to copy all the files to the storage and push them to Github.

Copying can be done using `!cp -r ./* "{PROJECT_PATH}"` executed in a notebook cell, which will update the Drive storage. Then, pushing to Github requires creating a

temporary working directory and **configuring local git repo** just for the time being. Here are the commands to execute:

```
!mkdir ./temp
!git clone
"https://{GIT_TOKEN}@github.com/{GIT_USERNAME}/{GIT_REPOSITORY}.git"
./temp
!rsync -aP --exclude=data/ "{PROJECT_PATH}"/* ./temp

%cd ./temp
!git add .
!git commit -m '"{GIT_COMMIT_MESSAGE}"'
!git config --global user.email "{GIT_EMAIL}"
!git config --global user.name "{GIT_NAME}"
!git push origin "{GIT_BRANCH_NAME}"
%cd /content
!rm -rf ./temp
```

Obviously, you need to define the strings in `"{...}"` yourself.

Figure 3. Successful upload of the content to Github. Calling it a day.

Conclusion

In this post, we have shown how to efficiently use Google Drive and Github **together** when working with Google Colab. The improved workflow is much simpler than the one presented earlier.

If you would like to share any useful tricks or propose some improvements, please do so in the comments. Your feedback is really helpful.

. . .

Originally published at https://zerowithdot.com.

Github Python Data Science Jupyter Notebook Google Colab