Quick Start Guide for Sharing Jupyter Notebooks

Skills required: assumes familiarity with Jupyter notebooks, using the command line and working with package managers for Python and R. Knowledge of Containerization[1] and the Git version control system are helpful but not required.

To share a Jupyter notebook, it takes more than just sharing the Jupyter notebook .ipynb file. Rather than sharing just Jupyter notebooks, reusability is made easier through distributing Jupyter notebooks in Binders.

1. Developing Jupyter Notebooks to be Shared

The Anaconda distribution is recommended for developing Jupyter notebooks to be shared. Anaconda is a popular Open Source distribution for Python and R that is used by researchers across many scientific domains. Anaconda improves reproducibility by including tools to export the working environment, which allows other researchers and services, like Binder, to easily replicate the Author's working environment. Anaconda can be installed on Windows, macOS and Linux workstations without needing privileged access. Anaconda can be downloaded for free at https://www.anaconda.com/download .

The Binder service is a popular free service for sharing Jupyter notebooks that allows anyone to run Jupyter notebooks from their web browser without having to install any software. The Binder service works by creating a Docker container to run any Jupyter notebooks that are available to publicly download, like on GitHub. Researchers who want to update these Jupyter notebooks can either update them directly on the Binder service or they can run the Jupyter notebook with Anaconda on their own workstation by cloning the GitHub repository.

2. Preparing Jupyter Notebooks

Jupyter notebooks should be clear, concise and have results that are reproducible. Below are guidelines for creating well written Jupyter notebooks:

1. Document the workflow and code.
    a. Create **Markdown[2] cells** to document whole cells or groups of cells.
    b. Use inline code comments to document complex algorithms and methods.
2. Do not include shell commands in cells unless the Jupyter notebook will be distributed in a (Docker) container.
3. Keep the cell output if other researchers will use the included data and code to replicate the output.
4. Clear the cell output if other researchers will use their own data or modify the code to create their own original output.
5. Any information that isn't pertinent to the running of the Jupyter notebook should be included in a README file and not in the Jupyter notebook. The exception to this is Author and Grant information.

[1] https://en.wikipedia.org/wiki/OS-level_virtualization
[2] https://en.wikipedia.org/wiki/Markdown

3. Portability

3.1 Create a New Environment

When developing a new Jupyter notebook project, use a new sterile environment that isn't polluted with dependencies from prior projects. Reusing the same environment can cause package version conflicts between projects, causing previously working Jupyter notebooks to fail.

A new environment can be created using the following commands:

Python: `conda create -n <enviroment_name> python=3 jupyter -y`

R: `conda create -n <enviroment_name> r-base r-essentials jupyter -y`

3.2 Managing Dependencies

The most important part of maintaining portability is managing dependencies. There are two parts to managing dependencies, identifying installed packages and identifying custom external libraries and datasets.

3.3 Package Dependencies

To ensure reproducibility include version numbers[3] with all included packages. If version numbers are not specified then the latest version of the package will be installed. Algorithms can change and APIs can be deprecated when developers update their packages. Jupyter notebooks can fail to run or return different results when package versions are different from when they were last tested.

Use the conda package manager[4] to install packages in order to simplify package management when creating portable Jupyter notebooks. The conda package manager has a large selection of packages for Python and R, however conda doesn't contain the complete selection of packages that are available. The conda-forge software channel has additional packages not available in the main software channel. To search for packages in the conda-forge channel run:

```
conda search -c conda-forge <package_name>
```

Most R conda packages names start with "r-". Packages from conda-forge can be installed by running:

```
conda install -c conda-forge <package_name>
```

If the package or version isn't found then use the PIP or CRAN package repositories, try to avoid installing packages from their source code.

[3] https://mybinder.readthedocs.io/en/latest/tutorials/reproducibility.html
[4] https://docs.conda.io/projects/conda/en/latest/user-guide/concepts/installing-with-conda.html

Anaconda can export the list of packages installed in your environment with version numbers in a format that can be shared with other researchers or used by Binder. The environment export command is:

```
conda env export -n <enviroment_name> --file environment.yml
```

The Anaconda environment export command will only capture packages installed using the conda and PIP package managers. CRAN R packages installed using the install.packages() command, should have all of the install.packages() commands written in a file named install.R[5] and a postBuild[6] file should be included for compatibility with Binder.

Include detailed instructions in the README file included with the Jupyter notebook for installing packages installed from source code. Jupyter notebooks that require packages installed from source code will need additional configuration files[7] in order to run on the Binder service. A postBuild[8] shell script is the recommended way to automate the installation of any packages from source and is compatible with the Binder service.

3.4 Custom External Library and Dataset Dependencies

Custom external libraries and datasets should be included in the root directory or a subdirectory. All references in the Jupyter notebook to the custom external libraries and datasets should be done using paths relative to the root directory.

Be mindful of large datasets. If the dataset is over 500 MB, include the smallest representative subset of the data with the Jupyter notebook. Links to the full dataset can be added to a README file included with the Jupyter notebook.

4. Sharing Jupyter Notebooks

4.1 Gather All of the Files

All of the Jupyter notebooks, environment files, custom external libraries, datasets and a README file should be placed in a single project directory before sharing the Jupyter notebook. Create a `binder/` or `.binder/` directory if you want to keep the environment files separate from the other project files.

The README file will be the first file displayed if shared to GitHub so the README file should contain the title and a description of the project along with any additional information that isn't appropriate to have in the Jupyter notebook itself. The README file is formatted using Markdown or reStructuredText[9] and named README.md (Markdown) or README.rst (reStructuredText).

See sections 3.3 and 3.4 for how to save the environment files, custom external libraries and datasets.

---

[5] https://github.com/binder-examples/r/blob/master/install.R
[6] https://github.com/kevincoakley/rehydratoR-jupyter-notebook/blob/main/postBuild
[7] https://mybinder.readthedocs.io/en/latest/using/config_files.html
[8] https://mybinder.readthedocs.io/en/latest/using/config_files.html#postbuild-run-code-after-installing-the-environment
[9] https://en.wikipedia.org/wiki/ReStructuredText

4.2 Test Jupyter Notebook Project with repo2docker (optional)

repo2docker is the tool that the Binder service uses to create the Docker containers that run the Jupyter notebooks. Testing your Jupyter notebook project with repo2docker will verify portability, make publishing with Binder easier and will create a Docker image that can be shared to a Docker repository. Testing with repo2docker is optional, but after the initial time required to install Docker Engine and download the Docker image, it can dramatically reduce the time to test your Jupyter notebook project with the Binder service.

1. Install Docker Engine, it can be downloaded for free for Windows[10], macOS[11] and Linux[12].
2. Open a terminal or command prompt window.
3. Install repo2docker:
   ```
   pip install jupyter-repo2docker
   ```
4. Go to the Jupyter notebook project directory.
5. Create and launch the Jupyter notebook project:
   ```
   jupyter-repo2docker .
   ```
6. Copy and paste the URL printed by Jupyter when it is done loading.

4.3 Share Jupyter Notebook with GitHub

GitHub is a service for public software development and version control based around the git[13] version control system. GitHub is the most popular service for sharing Jupyter notebooks despite GitHub's focus on software developers. GitHub's web interface makes it easy for novice developers to share their software without having to learn the git version control system. Use the following steps to share your Jupyter notebooks on GitHub.

1. Create a GitHub account by going to https://github.com/join.
2. Create new public GitHub repository[14].
3. Drag and drop all of the Jupyter notebook project to upload[15] to GitHub. Ignore the recommendation to create a new branch and commit directly to the master branch.
4. View the publicly available Jupyter notebook project at http://github.com/username/repo-name/.

4.4 Create Jupyter Notebook Binder

http://mybinder.org is the most popular free Binder service. Binders can be created by entering the GitHub repository URL into the Repository URL textbox and pressing the Launch button. It can take a few minutes for the Jupyter notebook environment to start since a new environment is built every time the Jupyter notebook project is launched, the build progress can be viewed in the Build Logs window.

---

[10] https://store.docker.com/editions/community/docker-ce-desktop-windows
[11] https://store.docker.com/editions/community/docker-ce-desktop-mac
[12] https://docs.docker.com/engine/install/#server
[13] https://en.wikipedia.org/wiki/Git
[14] https://docs.github.com/en/github/getting-started-with-github/create-a-repo
[15] https://docs.github.com/en/github/managing-files-in-a-repository/adding-a-file-to-a-repository

mybinder.org will create a URL to be shared that will automatically Launch Jupyter notebook project when you enter the GitHub repository URL. mybinder.org will also generate the MarkDown and reStructuredText code for a Binder Badge, copy and paste that code to the top of your README file in order to make it easy to launch the Jupyter notebook from GitHub. It is recommended to share the link to the GitHub URL so other researchers can view and download the Jupyter notebook project from GitHub or launch the Jupyter notebook project from the Binder Badge in the README file.