UNIVERSITY OF TORONTO

JUPYTERHUB

[Getting Started Guide]

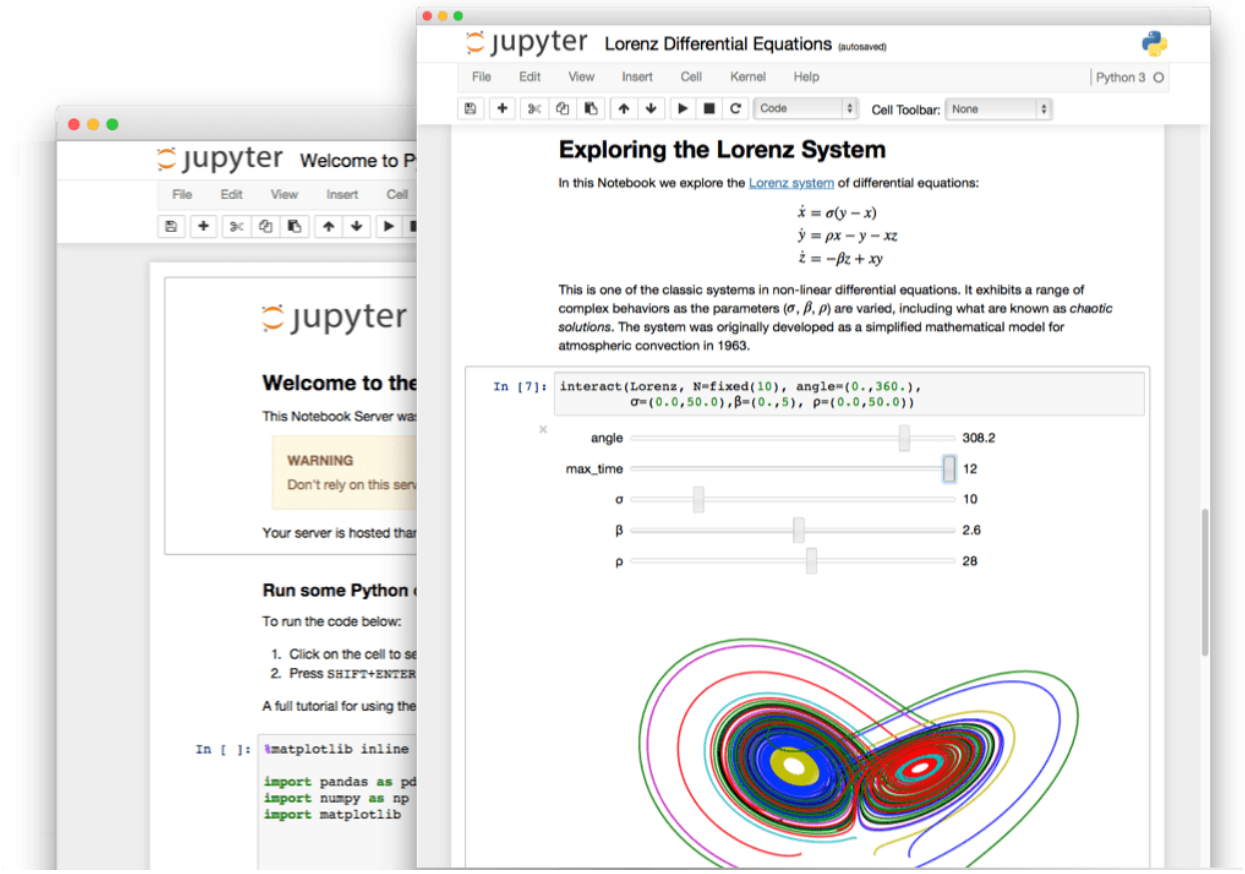
Contents

What is Jupyter Notebook?

The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations, and narrative text. Uses include: data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more.

Jupyter supports more than 40 programming languages, including Python, R, and Julia. The notebook can be shared just like any other document. The code within Jupyter notebook can produce rich, interactive output, including HTML, images, videos, and LaTeX. JupyterLab, Jupyter's Next Gen Notebook Interface, extends notebooks into fully interactive and customizable development environment for scientific computing applications.

The above information is summarized from Project Jupyter's official website. Learn more at *https://jupyter.org/*



How To Use Jupyter Notebook?

Log in to the UofT JupyterHub

To use the Jupyter hub environment, open any browser (Chrome, Edge, Safari, etc.) and search *https://jupyter.utoronto.ca*. The page will look like the image as shown below.

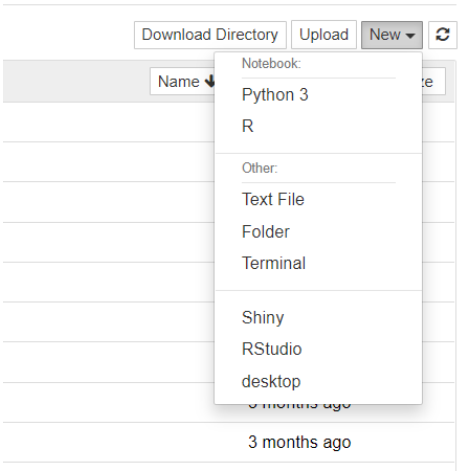
Select the interface you want to use and click on '**Log in to continue**'. This will redirect you to the sign-in page. Please use your **UTORid to sign-in**.

Jupyter file explorer

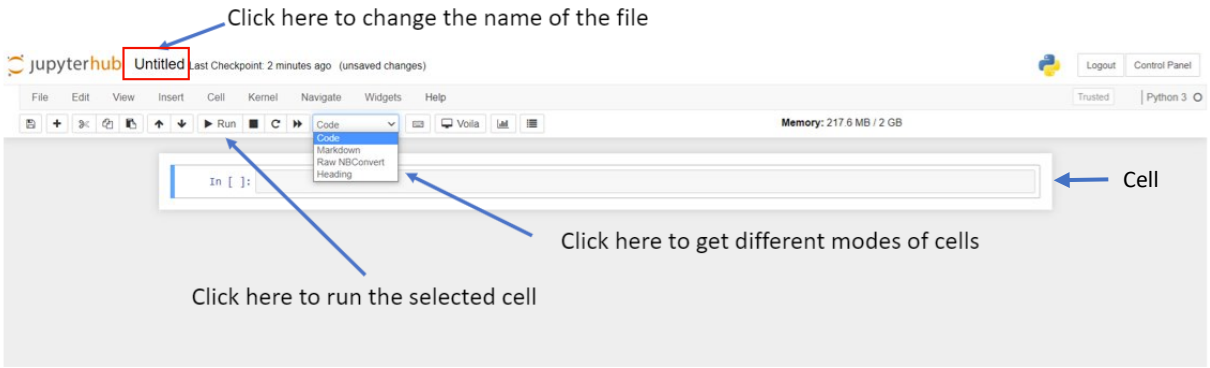
After logging in, a new instance/server will be allocated to you. You can store your data files, code files here. The file structure is similar to what you see on your own system (laptop/pc). A sample structure is shown below. The files created/uploaded to the Hub will remain stored until the end of a current academic term. **Please make sure to back-up files on your local computer if you want them after the end of an academic term.** (Instructions on how to upload and download files are given later in this document).

Create a new Jupyter notebook for python

On the top right corner of the file explorer, you can find a dropdown menu: "**New▼**", which will give you an option to either create a python, R, and text file or a folder, as shown below in the image. Click on '**Python 3**' to create a new python ipynb (iPython notebook) file.



After creating a new python 3 file, an '**Untitled**' file will open, as shown in the image below.

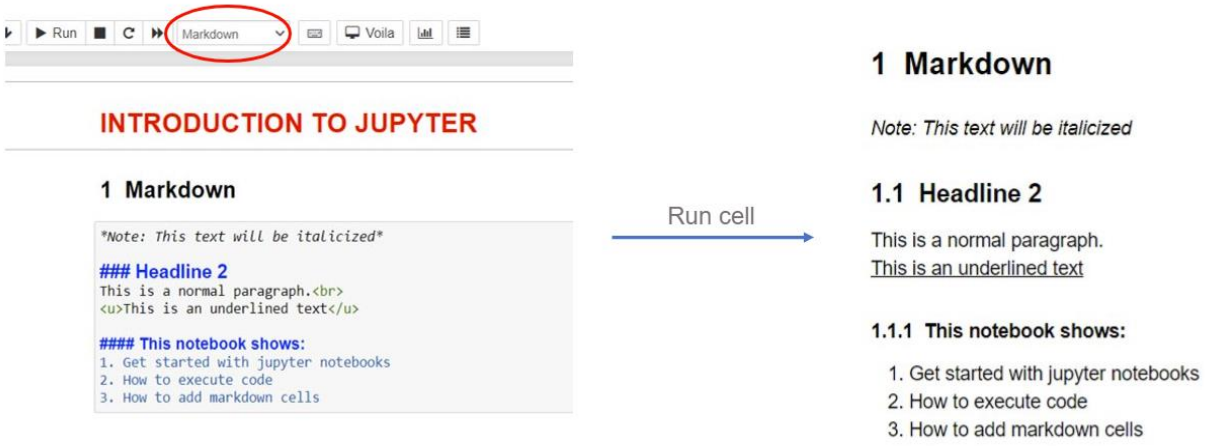


The file can be renamed by clicking on 'Untitled'. Just below the filename are menu bar and tool bar. The body of the page contains rectangle bar like structure called cell. This is where the code/markdown text is written. The toolbar has various buttons to add a new cell, copy a cell, run cell, change the type of cell (code, markdown, Raw NBConvert and heading), and so on.

Markdown cell

The markdown cell is generally used to write explanation or description of the code. These include headings, titles, bullet points, numbering, etc. The cell mode can be changed to Markdown type by selecting Markdown from the dropdown button in the tool bar.

An example of Markdown cell is shown below. When the cell is run, the output is rendered as shown on the right in the image below. To learn more about Markdown, refer *The Ultimate Markdown Guide (for Jupyter Notebook) | by Hannan Satopay | Analytics Vidhya | Medium*



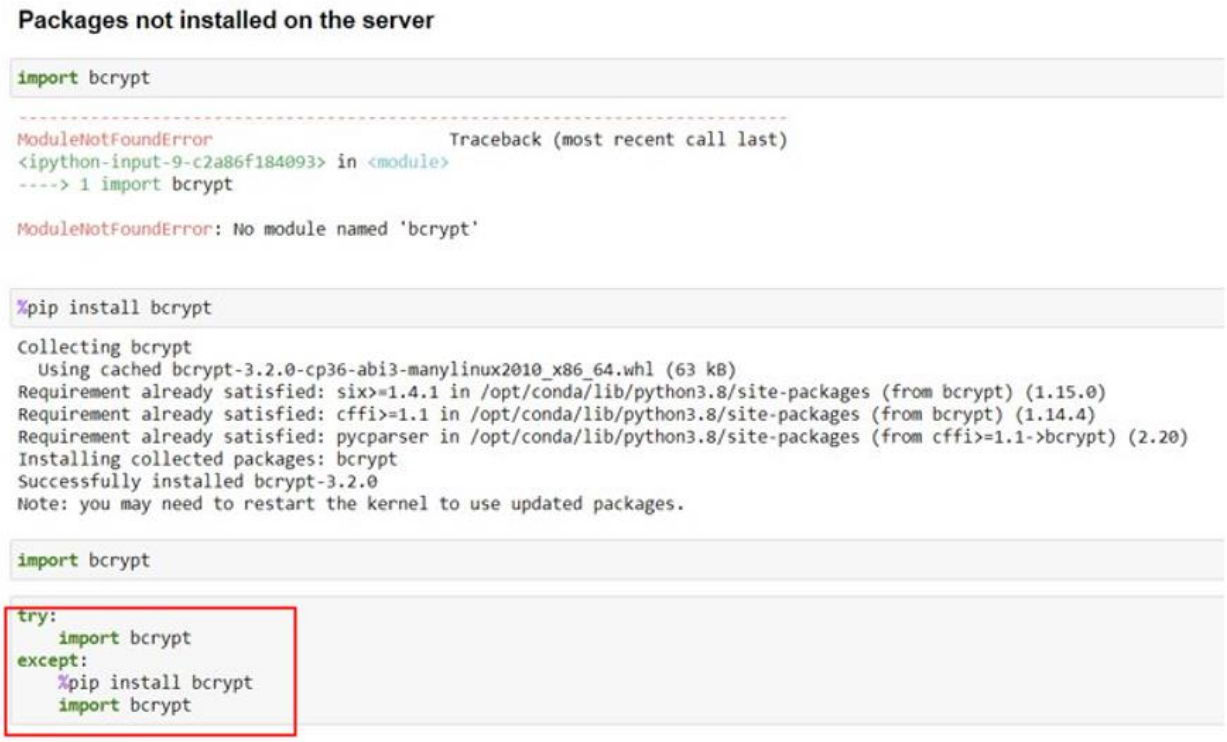
Installing packages

The UofT JupyterHub is setup with Anaconda Python distribution and comes with common packages pre-installed. The packages can be imported into current notebook as usual by typing "**import <package-name>**".

You can install your own packages or packages that do not exist in the UofT hub directly from the code cell by using the Jupyter line magic command, %.

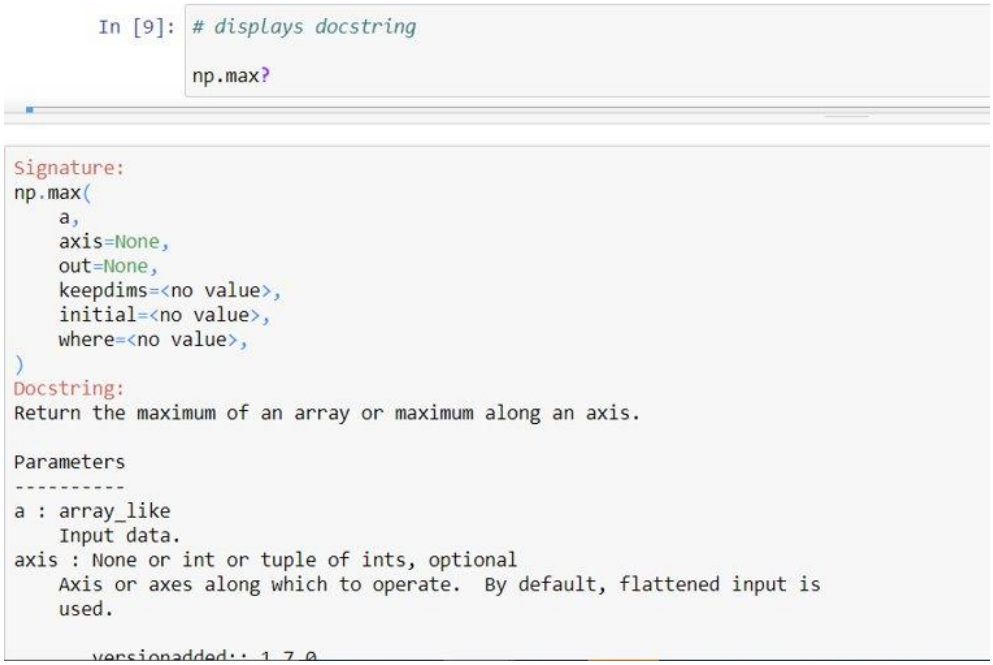
To install a new package, run "**%pip install <package- name>"** in a new cell. By doing this, you do not need to open the terminal to install new packages. This is done from within the notebook itself.

The manually installed packages do not persist in the system after logging out. This means that if you had installed a package from within the notebook, the next time you log in and try to run the code, you will have to re-install the package. To avoid this, it is better to implement the try and except method offered by python. An example of such manner is shown below and indicated by red rectangle.



Docstring (Package, class, and function descriptions)

Another advantage of a Jupyter notebook is that you can easily read the descriptions of packages or python objects such as class, methods, and functions, if the description is provided in the code by developer. Just type '**?**' after the function/package name and run the code cell.



```
In [9]: # displays docstring

        np.max?
```

```
Signature:
np.max(
    a,
    axis=None,
    out=None,
    keepdims=<no value>,
    initial=<no value>,
    where=<no value>,
)
Docstring:
Return the maximum of an array or maximum along an axis.

Parameters
----------
a : array_like
    Input data.
axis : None or int or tuple of ints, optional
    Axis or axes along which to operate.  By default, flattened input is
    used.

       versionadded:: 1.7.0
```

Magic commands

The '**%**'sign used while installing new packages and '**?**' used to view the documentation of certain packages or functions are examples of magic commands. These commands can be used in Jupyter notebooks for enhancing the functionality and understanding of the code. Magics are specific to and provided by the IPython kernel. There are many Magic functions available with python kernel. Two examples are shown below. To learn more about other available commands, please refer the documentation at *Built-in magic commands — IPython 7.19.0 documentation*

Examples of such magic commands are:

1. **%%time** or **%time** : These are used to measure the time taken by a function or a line of code to execute. To use this command in line mode prefix with single % sign. To use it in cell mode, prefix with %%.
2. **%who_ls** : This command shows the sorted list of all the interactive variables/aliases used in the notebook.

```
def fact(n):
    out=1
    for i in range(1, n+1):
        out = out*i
    return out
```

```
%%time
fact(5)
```

```
CPU times: user 3 µs, sys: 2 µs, total: 5 µs
Wall time: 8.58 µs

120
```

```
%who_ls
```

```
['bcrypt', 'fact', 'np', 'pd', 'time']
```

How to share/distribute files?

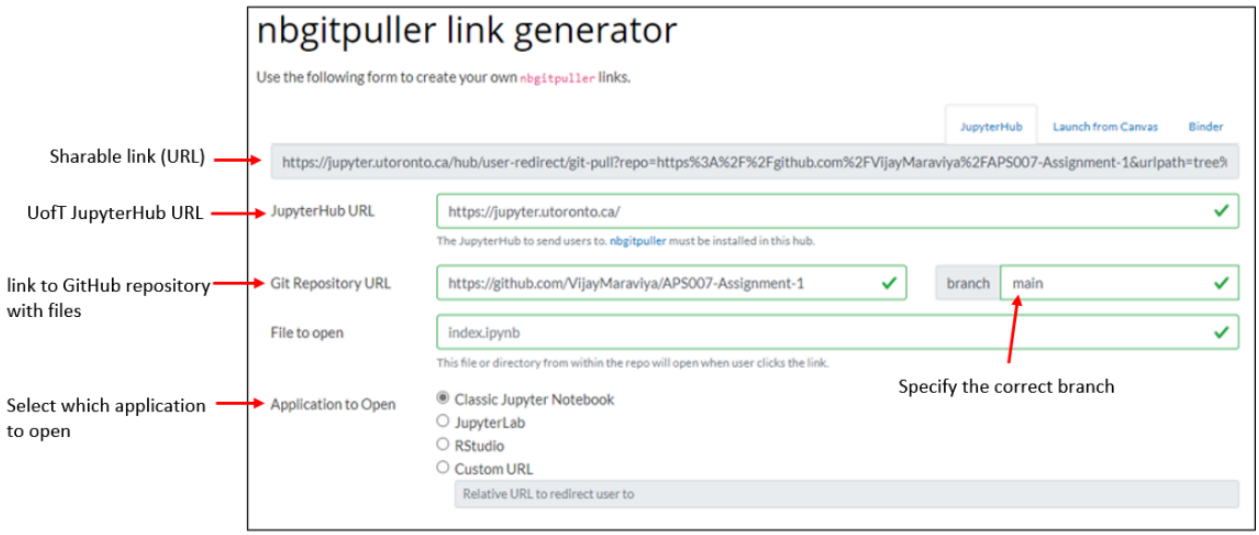
The 3 ways to distribute files for use in JupyterHub are described below:

One-click download (nbgitpuller)

This method requires files to be uploaded in a GitHub repository. A sharable link can be created using nbgitpuller. When clicked on this link, students will be taken directly to the UofT JupyterHub and files will be fetched automatically into students file explorer inside JupyterHub. To distribute files using this method, follow the steps as described:

Step 1: Create public repository on *GitHub* and upload files to the repository.

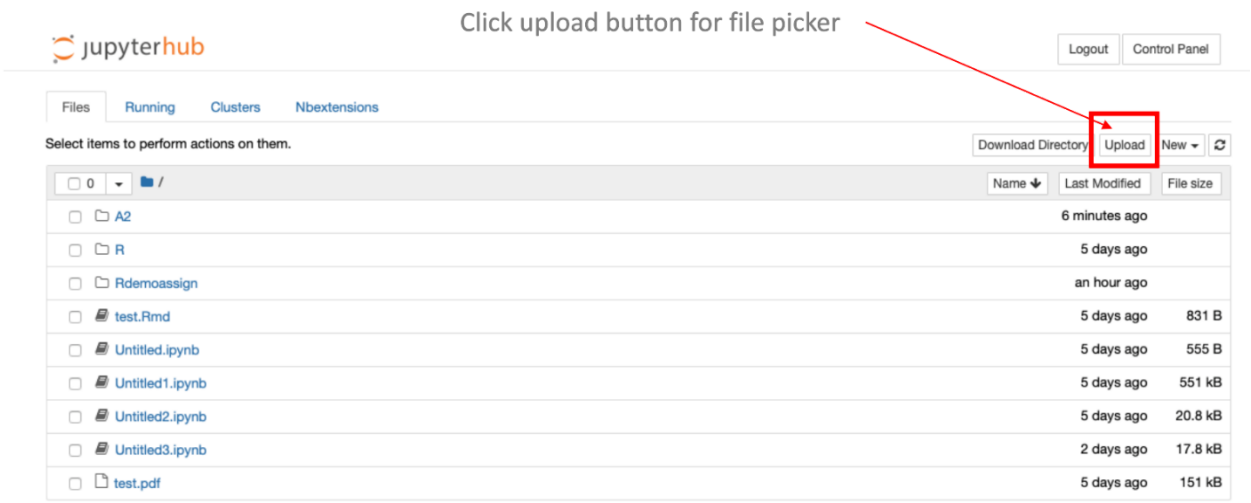
Step 2: Go to "*https://jupyterhub.github.io/nbgitpuller/link*" and fill out all the required details as shown in the image below.



Upload/download files manually

Professors can post all the files required for the assignment on Quercus from where the students can download and upload manually to their JupyterHub directory.

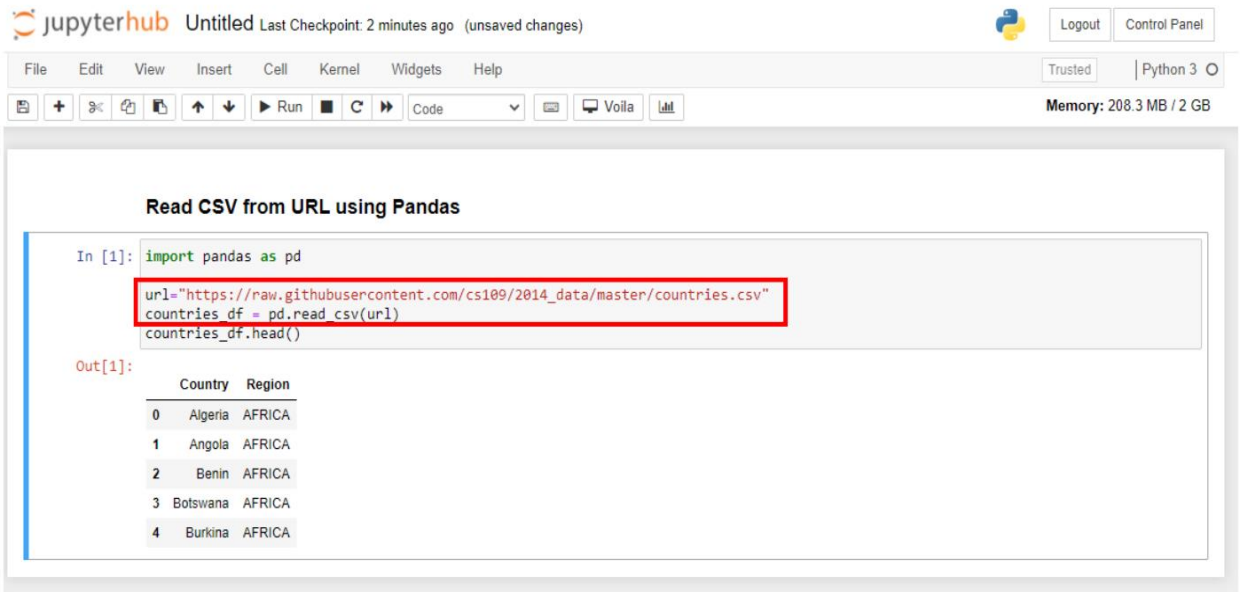
Clicking on the upload button will open the file picker to select and upload files from student's computer to JupyterHub.



Read files from URL using supported Python modules

Many python modules provide the functions than can read specific file formats from a URL. This is not a feature of JupyterHub. Example for one such module, Pandas, is shown below which reads csv file in Pandas DataFrame object.

Read table data from URL in Python



Additional Resources

- A sample Jupyter notebook: Jupyter Notebook Viewer | Exploratory data analysis in Python