



Quick Start Guide



What is Jupyter?

Jupyter is a manifestation of **reproducible research**.

Reproducible Research is a concept, originating in academia, as per which every research submission encompasses the findings, as well as the inputs that lead to it, in a single package that would allow for reproducing the observations elsewhere.

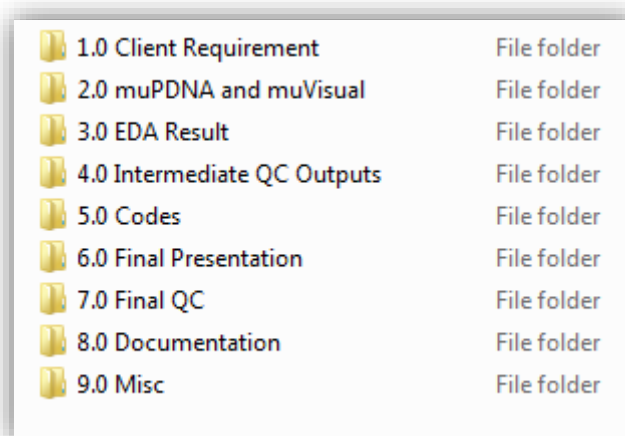
In our context, it means a single, streamlined workflow, in which all aspects of the project – the introductory material, the analyses and the documentation are rolled into a single document, leading to a single source of truth.

Reproducibility, as the name suggests, also advocates reusability of code. This entails creation of reusable codes which allow the same file to be used across multiple similar analyses with minor tweaks, and not having to start from scratch. This also allows for standardization and reduces copy paste (and hence, errors induced by this manner of work).

Jupyter also reinforces the practice of **literate programming**. Literate programming is an approach to make codes readable and comprehensible to humans. The style of writing code is one that follows human logic, or thoughts, rather than the order imposed by computers. This would enable audiences not closely coupled with the coding process to also understand and follow the coders' thought process and logic.

Why Jupyter?

Today, most projects include the following components:



1.0 Client Requirement	File folder
2.0 muPDNA and muVisual	File folder
3.0 EDA Result	File folder
4.0 Intermediate QC Outputs	File folder
5.0 Codes	File folder
6.0 Final Presentation	File folder
7.0 Final QC	File folder
8.0 Documentation	File folder
9.0 Misc	File folder

Fig. 1: The directory structure for a generic project

While there is no contending the amount of work that goes into it, all the *knowledge* is scattered across multiple folders. Notebooks allow to merge many, if not all, of these

components into a single, self-contained workflow. It also fosters transparency, allowing the client to not only appreciate our findings and insights, but also how we got there.

Seriously, why *Jupyter*?

There are multiple implementations of reproducible research, the most popular one other than Jupyter being R-Markdown. Here's why Jupyter is better:

- Supports many languages, including Python, R, SAS, Scala, Perl, PHP and over 40 other languages
- Jupyter allows for sequential processing, allowing the user to see the output of every chunk of code immediately

Creating a Notebook

Once Jupyter is installed, Jupyter Notebook can be opened from the start menu and typing `Jupyter notebook`. If that doesn't work, alternatively you could fire up command prompt and type in `Jupyter notebook`. In either case, a command prompt window opens (in the latter method, the process continues within the same command prompt window you opened). Make sure you do not close this window as long as you are working on Jupyter.

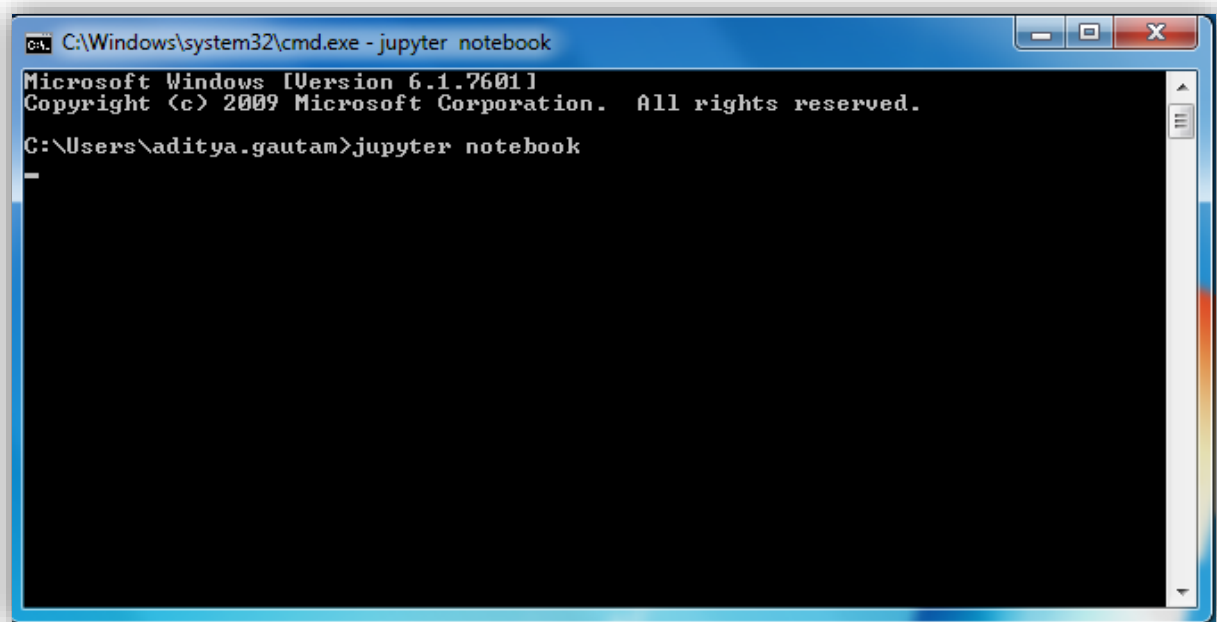


Fig. 2: Starting Jupyter Notebook

Finally, you can create a new Jupyter notebook by navigating to your browser window, where your Jupyter instance is running, and click on `New`:

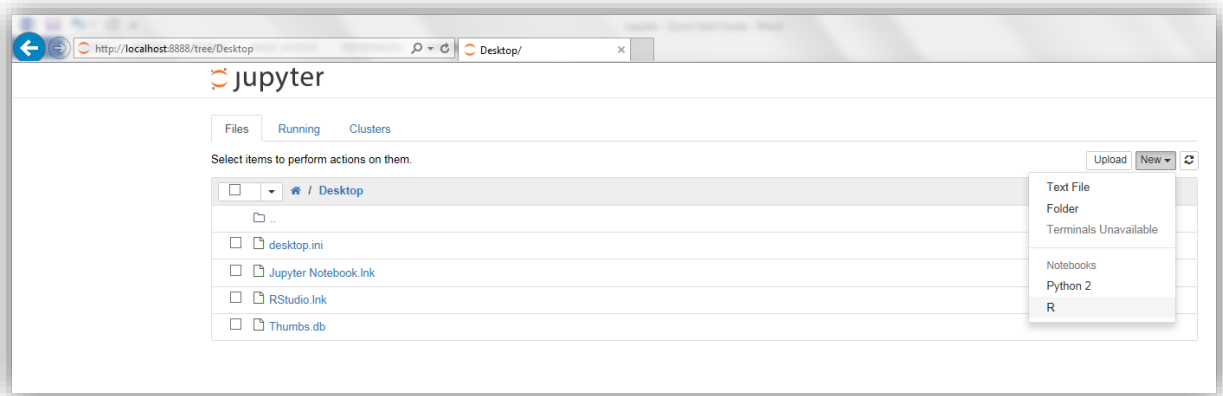


Fig. 3: Creating a new notebook

Post creating a notebook, you can start creating your notebook in a language of your choice, along with commentary.

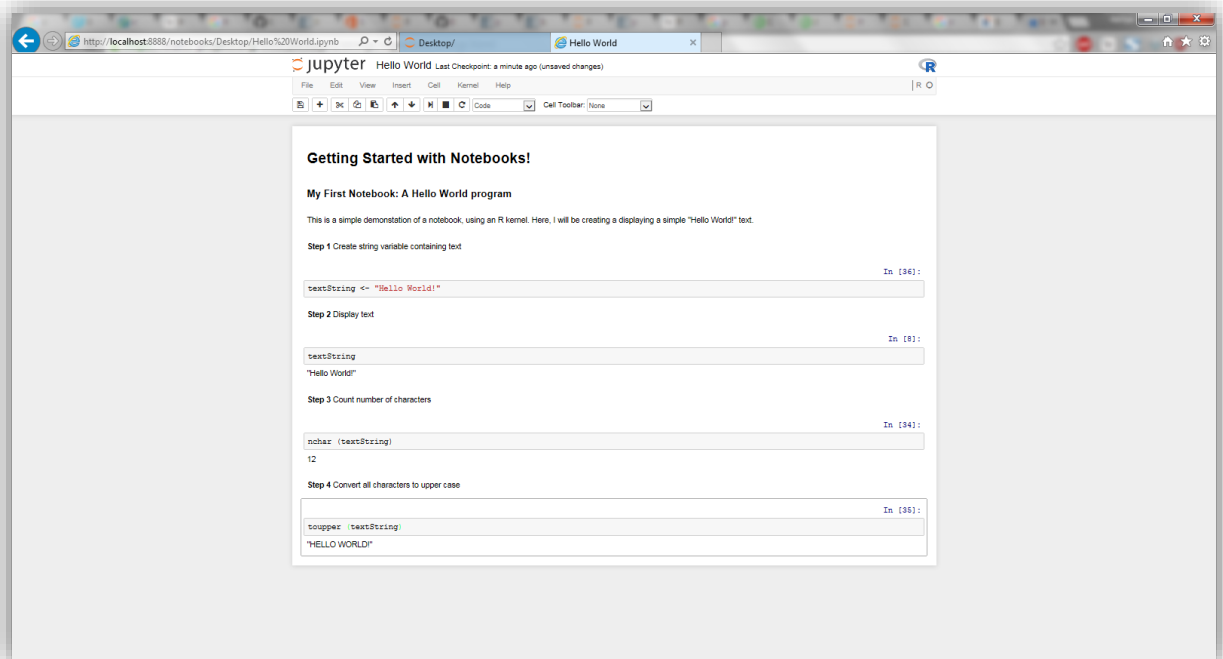


Fig. 4: A Hello World program

Once you are setup with Jupyter, and look forward to use it, it is necessary to understand what Jupyter is, and what it isn't, as we will see in the next couple of sections.

What Jupyter *is*:

A way of working

More than anything else, it is a style of working – working in a cleaner, neater more presentable and reproducible fashion. To achieve this, it also plays the following roles:

A coding environment

In order to encapsulate prose and code into one, Jupyter presents a simple, bare bones coding environment

A presentation tool

With rich text formatting using Markdown, Jupyter helps in creating, neat, tidy if not very pretty client presentable documents

A documentation scheme

With a single document containing all steps, along with reasoning, it inherently contains all the documentation required for the said analyses or process

What Jupyter is *not*.

While Jupyter has many tricks down its sleeve, it is also important to understand what is not.

A new technology or language

It is only a new interface! It is not a new technology, or software or language

A magical tool that fixes everything

It doesn't come with a big red button with 'Fix' written on it. To put everything together would still require proper coding, annotating, and inferring

Another place to dump code

Jupyter is a powerful platform – provided it is used that way. If huge codes are all dumped into a single code cell, it will not serve its purpose, and will still remain as messy as it did using the previous environment

Frequently Asked Questions

Do I have to learn Jupyter?

No. Jupyter is not a new language. You can continue to code in the language you're currently using.

Still, are there any pre-requisites to using Jupyter?

Like mentioned above, you can continue using the language you current are – which means you *do know that one language* to work in, because for the coding bit, you need to know at least one coding language

My work mostly concerns with data manipulation, through SQL or Hive (or other big data technologies), but not so much of R or Python. How can I use Jupyter then?

Jupyter supports multiple kernels (over 40 of them). It also supports something called Magic, which allows you to code in multiple (supported) different languages in a single notebook. Even beyond that, R and Python allow you use packages to run code in a variety of different languages, so it is easy to use Jupyter Notebook for other languages as well.

What all existing languages can be used with Jupyter?

See <http://bit.do/JupyterKernels> to get a complete and up-to-date list of different languages (*kernels*) supported by Jupyter.

What if I don't use any of these languages, but work on BI or reporting tools?

Jupyter can do little in projects that use only GUI development tools such as Tableau, or QlikView. However, the data prep and manipulations part, as well as refresh procedures and documentation can still utilize Jupyter.

How many notebooks can be run simultaneously?

Really depends on the nature of operation being carried out. Computationally expensive operations limit the resources available on your system, so that reduces the number of any kind of process to run on your system, including notebooks. And since

this uses R or Python (or other languages), it is up to the language being used, the nature of operation and resources available that determine number of simultaneous notebooks.

Why should I use Jupyter?

Just some of the most immediate benefits:

- Structured, transparent (to the end customer) code
- Improved documentation (richer, easier documentation)
- Reusability (plug data and play, spend more time deriving insights rather than writing code)
- Integrated workflow (not having to use multiple different software for data ETL, EDA, modeling, presenting results, and documenting)
- Fewer errors (due to standardization and lesser amount of copy-pastes)

How does Jupyter take care of version control?

Jupyter can help document iterations within a notebook, but doesn't have any version control system per se. However, versioning can be done by using Jupyter in conjunction with Git.

What is Markdown and how is it used?

Markdown is a very simple way of formatting text. It involves using a very simple set of symbol prefixes to format text. For a complete list of markdown syntaxes, check out <http://bit.do/MarkdownWiki> (the Wikipedia page for Markdown) for a quick start, or <http://bit.do/MarkdownProject> (the Markdown project page) for a complete set of syntaxes.

Does usage of Jupyter require knowledge of HTML (for the markdown or commentary part)?

No. Markdown is all that is required to format text. HTML may be used to build on Markdown's capabilities and take formatting and presentation to the next level, but for most practical purposes, Markdown alone will suffice.

How different will working with Jupyter be?

Jupyter, as explained, is a different *style* or *way* of working. That being said, it is not an IDE (*Integrated Development Environment*, such as RStudio, Spyder or Eclipse), and as such, there are differences.

Everything is typed in cells. Prose (commentary, annotations, notes, basically anything that is not code) is written in markdown (see previous answer). Code is followed by output (text or graphic). It's like a `code, console; code, console; ... code console;` like layout.

While this is functional, this differs from different individual IDEs, which is only natural because it wouldn't be possible for Jupyter to incorporate all of the different features from each of the 40+ languages' IDEs that it supports.

We work with SAS and not R/Python (Variant: We work only on SQL). How is using Ipython or Jupyter relevant for me?

If you are currently working with RDBMS, then you are in luck. Python is extremely powerful in connecting with other technologies. There are packages for using ODBC / JDBC drivers or other API's through which you can connect to these databases.

Regarding SAS, there is a new open source project called *SAS_Kernel* (available in GitHub) which allows you to run SAS programs from Jupyter notebooks. However this requires SAS 9.4 and above running on Linux and Python 3 running on the same machine. If these conditions are not met (and most likely they won't be since this was announced in April 2016), then you will not be able to use Jupyter.

I work in Linux environment through a Putty terminal. How do I use the notebooks?

You should be able to start Jupyter notebook running on your Linux machine and connect to it through your browser by specifying the IP address. Reach out to I&D to help you out on this.

How is Reproducible Research helping me to reduce errors?

Improved documentation. You have the ability to generate output and store it with code in a single document. This is a very powerful capability of Jupyter notebooks. A structured approach to documenting – coding – QC – documenting should definitely reduce errors.

I still need to use PPT and Excel for sending out deliverables to the client. Hence copy paste is still happening. What then?

Ideally, try using Jupyter as a single platform for even reporting results. However, should the need be pressing, you can explore Python packages which can read / write data into Excel & PowerPoint.

How do we collaborate over Ipython or Jupyter notebooks?

Jupyter allows collaboration through Jupyter Hub.

We do lot of adhoc work in our day to day deliverables? Is it worth the effort?

Yes, as mentioned above – you can document your adhoc deliverables along with the code. It becomes even more important when transitioning this to someone else.