Chapter 7

Dynamic Notebooks: Jupyter, Markdown, and Pandoc

In this chapter, we describe how to create, handle, and share dynamic notebooks. The objective is to provide you with easy-to-use interactive notebooks, a modern-day equivalent of traditional laboratory notebooks, where scientists recorded the steps of their investigations, their inputs, and their findings. An interactive notebook is a convenient medium to mix text, commands, and output such as graphs or tables in a shape that you can store and share with third parties. For example, you can easily generate full static and simplified `HTML`, LaTeX, and `pdf` files and `reveal.js` slides (check `https://revealjs.com/#/`) from a `Jupyter`'s notebook with minimal effort. This will help you to achieve the goals of good software engineering (reproducibility, documentation, etc.) outlined in Chapter 2.

In Section 7.1, we will introduce `Jupyter`, a popular and open-source interactive notebook. `Jupyter`'s notebooks are organized around cells, their fundamental building block. The menu options that create and manipulate cells are intuitive to navigate and require very little explanation. `Jupyter`, therefore, provides a simple but flexible environment for researchers and teachers in economics who want to develop software and document their progress with a minimal investment in learning how to work with a notebook. In Section 7.2, we will briefly describe `Markdown`, a lightweight markup language that is both readable and versatile and that provides many of the capabilities of LaTeX.[1] We will close in Section 7.3 with a concise introduction to `Pandoc`, a translator from one markup language to another. `Pandoc` will help you move from `Markdown` to LaTeXor `HTML` (or even `Word`) and vice versa without fuss.

[1]A markup language creates a file that is annotated by tags and where there is a difference between the document and the text that the document generates. LaTeXis an example of a markup language with typesetting instructions. `XML` is an example of a markup language with structural markers. Markup languages stand in comparison with word processors based on the idea os WYSIWYG (what you see is what you get).

7.1 Jupyter

`Jupyter` is a network protocol for interactive computing.[2] The name is a portmanteau of `Julia`, `Python`, and `R`, three programming languages for which it was designed. However, today you can also employ `Jupyter` with many other languages and frameworks such as `Scala`, `Haskell`, `Ruby`, `Spark` and, with the `Cling` interpreter, even with `C++`.[3] `Jupyter` evolved from `IPython`, an (I)nteractive `Python` shell (thus its name). As the shell grew to allow its use with other programming languages, it made sense to transition to a language-agnostic setup.

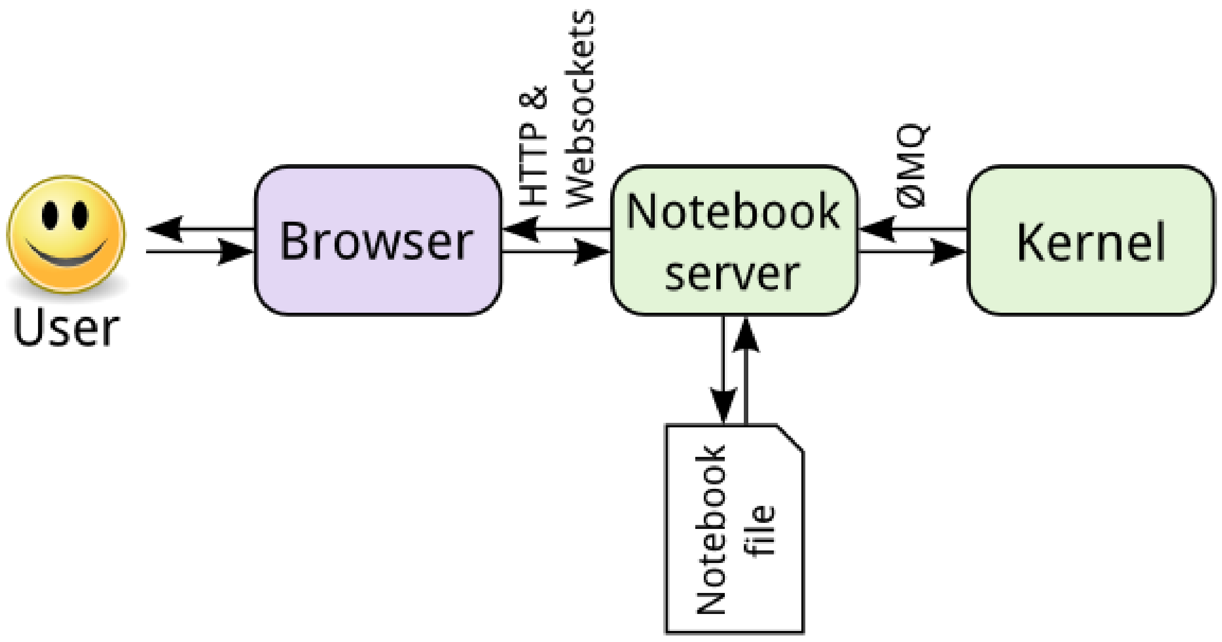


Figure 7.1: `Notebook components`

`Jupyter` allows you to introduce instructions in the language of choice, send them to the kernel of that language for execution, display the result, and store it, and all within through your browser. The structure of the system appears in Figure 7.1, from the `Jupyter`'s documentation webpage.[4]

Technically, `Jupyter` creates a notebook that is a JavaScript Object Notation (JSON; pronounced "jay-son") file and the notebook server communicates both with the browser (front-end; through HTTP and websockets) and the kernel (the back-end; through ØMQ, an

[2]See `http://jupyter.org/` for further details and documentation and `https://blog.jupyter.org/` for a blog on the project with news and updates. An introductory textbook is Toomey (2016).

[3]`https://github.com/root-project/cling`. Note that `Cling` generates interpreted code, not compiled one.

[4]`https://jupyter.readthedocs.io/en/latest/architecture/how_jupyter_ipython_work.html.`

asynchronous messaging library). JSON is an open-standard format for data transmission that aims at being both simple to read and write for humans and easy to parse by computers. It is organized around collections of name/value pairs and ordered lists of values. Normal users of `Jupyter` do not need any knowledge of JSON, although the notebook will have an option to edit its JSON metadata for advanced configuration.[5]

You can run `Jupyter` online at https://try.jupyter.org/ without the need of installing it in your computer (or the support for the programming language that you employ). If you are a `Julia`, you can also run https://www.juliabox.com/. These online tools will give you much of the capabilities of `Jupyter` right away. They are a great way to get a taste for `Jupyter` and for experimenting with languages such as `Julia`, `Python`, and `R` with next to zero start-up cost.

However, eventually, you may want to install `Jupyter` in your computer and not having to depend on a (potentially slow) remote server and an internet connection. In Chapter 8, we will suggest that you install `JuliaPro`, a powerful `Julia` distribution that comes with `Jupyter` bundled. Perhaps you can jump to that chapter and read the section on how to install `JuliaPro` and get `Jupyter` to work with your OS. If you do not want to install `JuliaPro`, you can follow the instructions in https://jupyter.readthedocs.io/en/latest/install.html#install and install `Anaconda`, a very complete `Python` distribution with extra components gathered toward data science such as `Spyder`.[6] Once you have installed any of these tools, `Jupyter` will rely on your default browser to get and display information. If that does not suit you, you can either change your default browser or modify `Jupyter`'s configuration to force the server use a different browser than your default (check the internet to learn how to do it in your OS).

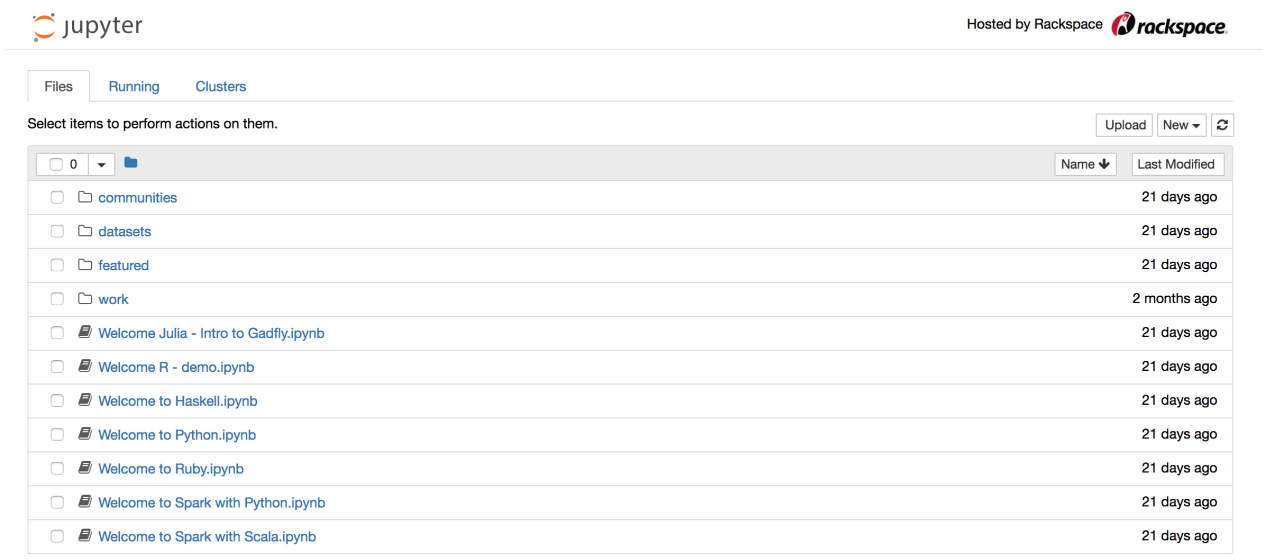
There is also additional optional material for installation. First, you can install `Jupyter Widgets`, which will provide you with widgets such as sliders, textboxs, status bars, etc. You can learn more about them at https://ipywidgets.readthedocs.io/en/stable/. Second, other widgets to include inteactive tables, graphs, and maps and other visualization tools are available at http://jupyter.org/widgets.html. Third, you can install `JupyterHub`, a multi-user Hub, for example, to teach a class on computational methods (https://jupyterhub.readthedocs.io/en/latest/). You probably will not need any of these extensions until you become an expert user of `Jupyter`, so we will ignore them for the moment.

When you enter into the notebook, either online or in the version installed in your com-

[5]See https://www.json.org/. JSON's syntax and conventions would be very familiar to programmers with experience in "curly brackets" languages such as `C`, `C++`, `Java`, `JavaScript`, or `Python`. JSON can interact, however, with nearly all major (and most minor) programming languages.

[6]https://pythonhosted.org/spyder/.

puter,[7] you will see something similar to Figure 7.2. The local address will be of the form `http://localhost:8889/tree?` or similar) (in this and all the next figures, we are deleting the OS- and browser-specific menus at the outer frame of the display; the figure shows what you see inside your browser window).



Select items to perform actions on them.

Name	Last Modified
communities	21 days ago
datasets	21 days ago
featured	21 days ago
work	2 months ago
Welcome Julia - Intro to Gadfly.ipynb	21 days ago
Welcome R - demo.ipynb	21 days ago
Welcome to Haskell.ipynb	21 days ago
Welcome to Python.ipynb	21 days ago
Welcome to Ruby.ipynb	21 days ago
Welcome to Spark with Python.ipynb	21 days ago
Welcome to Spark with Scala.ipynb	21 days ago

Figure 7.2: `Entry into Jupyter`

Let us inspect us the display. First, at the top left, you see the `Jupyter` logo. You can always click on it to go back to the original homepage that appeared at the start of your session. On the top right you have a `Logout` button to left `Jupyter`. Below this top line, and occupying nearly all the display, there is a `Files` tab with a directory of folders (in this case, the online folders provided by the `Project Jupyter`) and notebooks, with extension `ipynb`. The folders and notebooks are ordered either by name or by last modification. You can click on the corresponding buttons to change the criteria and decide whether you want an ascending or descending order. To the left of the name of each folder or file there is white empty box. If you click on it, a menu will appear to perform standard files operations (`duplicate, shutdown, view, edit, delete`). `duplicate` is particularly useful to have a copy of your notebook for experimentation. You also have a checkbox with a drop-down menu to select `Folders, All Notebooks, Running` and `Files`.

To the right of the `Files` tab you will see two other tabs: `Running` and `Clusters`. The `Running` will display the OS terminals and notebooks being run at any given moment

[7]How to enter into the notebook will depend on your choice of interaction with `Jupyter`. In Chapter 8 we describe how to jumpstart `Jupyter` from `JuliaPro`. The online versions are intuitive to open once you go to their webpages and click on the menus.

(we will describe below how to initiate a terminal), with an option to shut them down and information on their last change. The `Clusters` tab is for working with clusters and we can ignore it in this chapter.[8]

In the top right corner of the display, you can find the `new` checkbox, where a drop-down menu allows you to select the language that you want for your notebook or open a plain text file, create a folder, or initiate a terminal with your OS. You also have a button to upload a previous notebook or you can just drag-and-drop an existing one and a refresh button to update the display. If you select, for example, `Julia`, for your new notebook, you will see something similar to Figure 7.3.

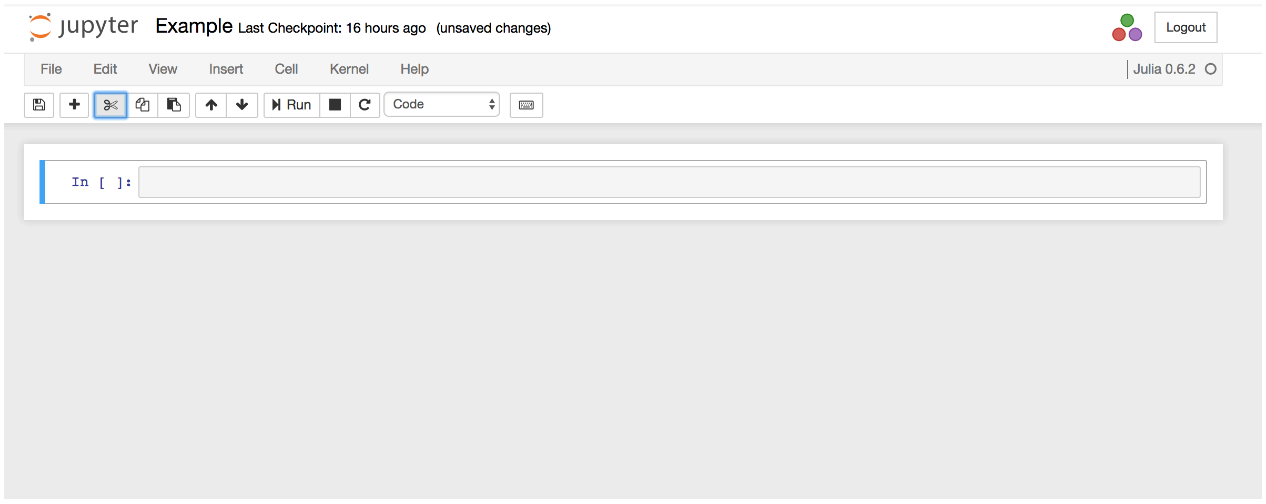


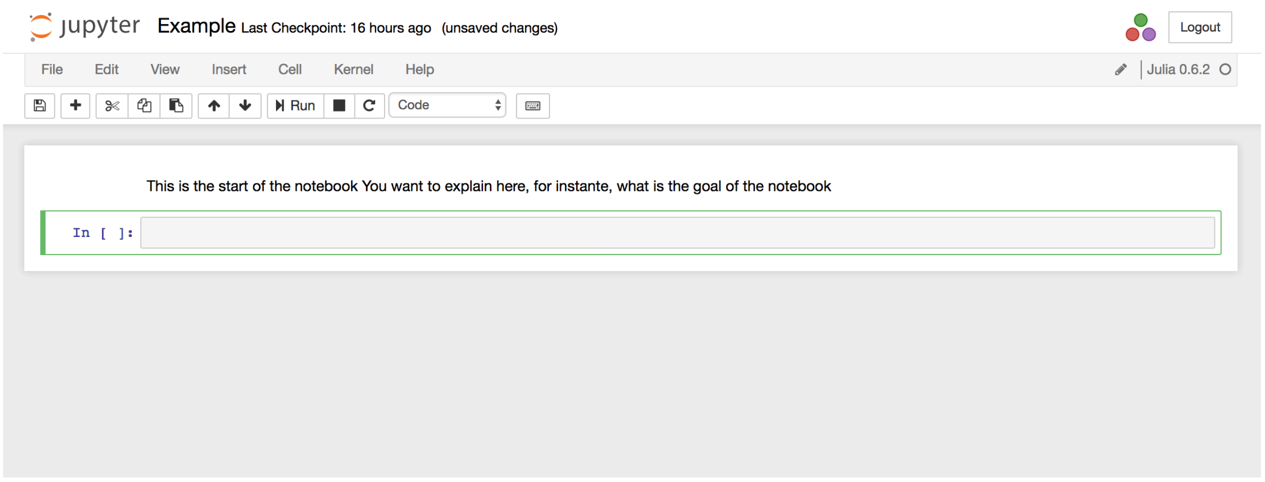
Figure 7.3: `A Julia notebook`

Let us inspect this new display. On the top left, close to the `Jupyter` logo, you will see `Untitled`, the default name for your notebook. If you click on it, you can change the name to something more memorable, in this case `Example`. `Jupyter` will assign the extension `ipynb` automatically.

The next thing you will note is a cell with a blue line at the left that starts with `In []:`. By clicking on it, you can type any text or code that you want. For example, we can type some basic text information and, in the menu button whose first option is `Code`, we can select `Markdown` to tell the notebook this is `Markdown` text, and then we click on `Run`, to get something like Figure 7.4. When we cover `Markdown` in Section 7.2, you will learn how to write sophisticated text with documentation of your code, your procedures, etc. For example you could use text cells to describe the main goal of the project or to explain where the data

[8]This would also require the installation of `IPython parallel`. See `https://github.com/ipython/ipyparallel` for documentation.

comes from and how you are loading it from some files.

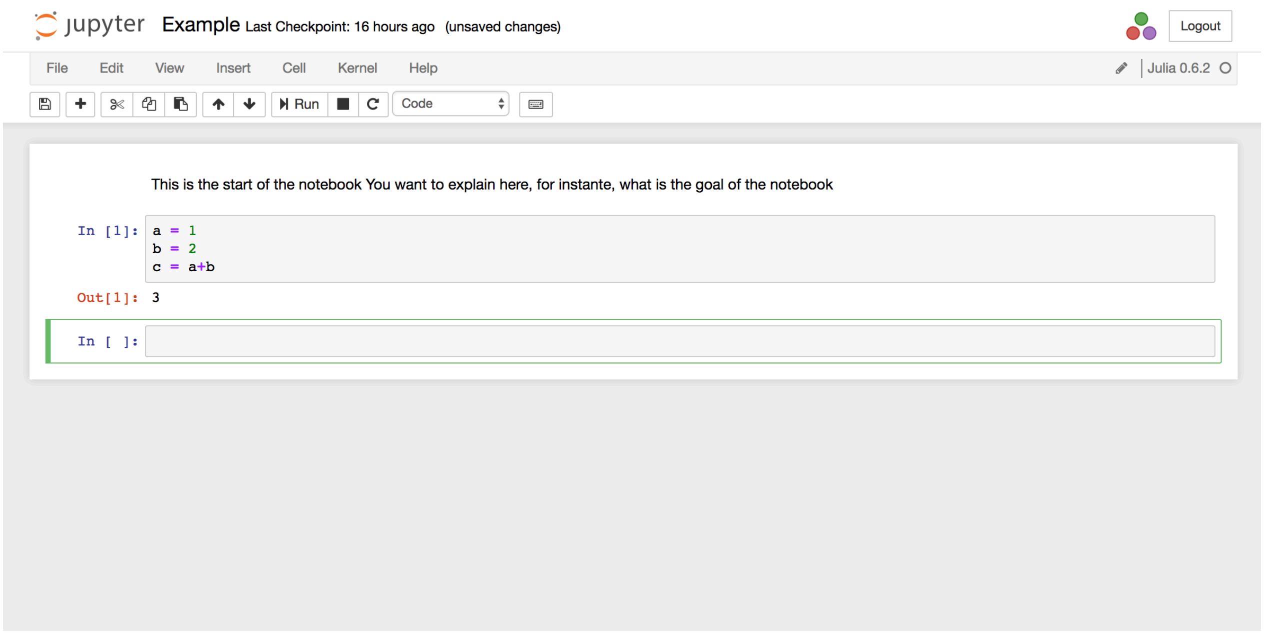


This is the start of the notebook You want to explain here, for instante, what is the goal of the notebook

```
In [ ]:
```

Figure 7.4: `A first cell`

You can do the same with a short piece of code (you do not need to worry too much about the syntax of `Julia` at this moment, although it should be rather self-explanatory) except that you select the option `Code` and click either `Run` or `Swift+return`. The result, with `Out [2]:`, appears in Figure 7.5.



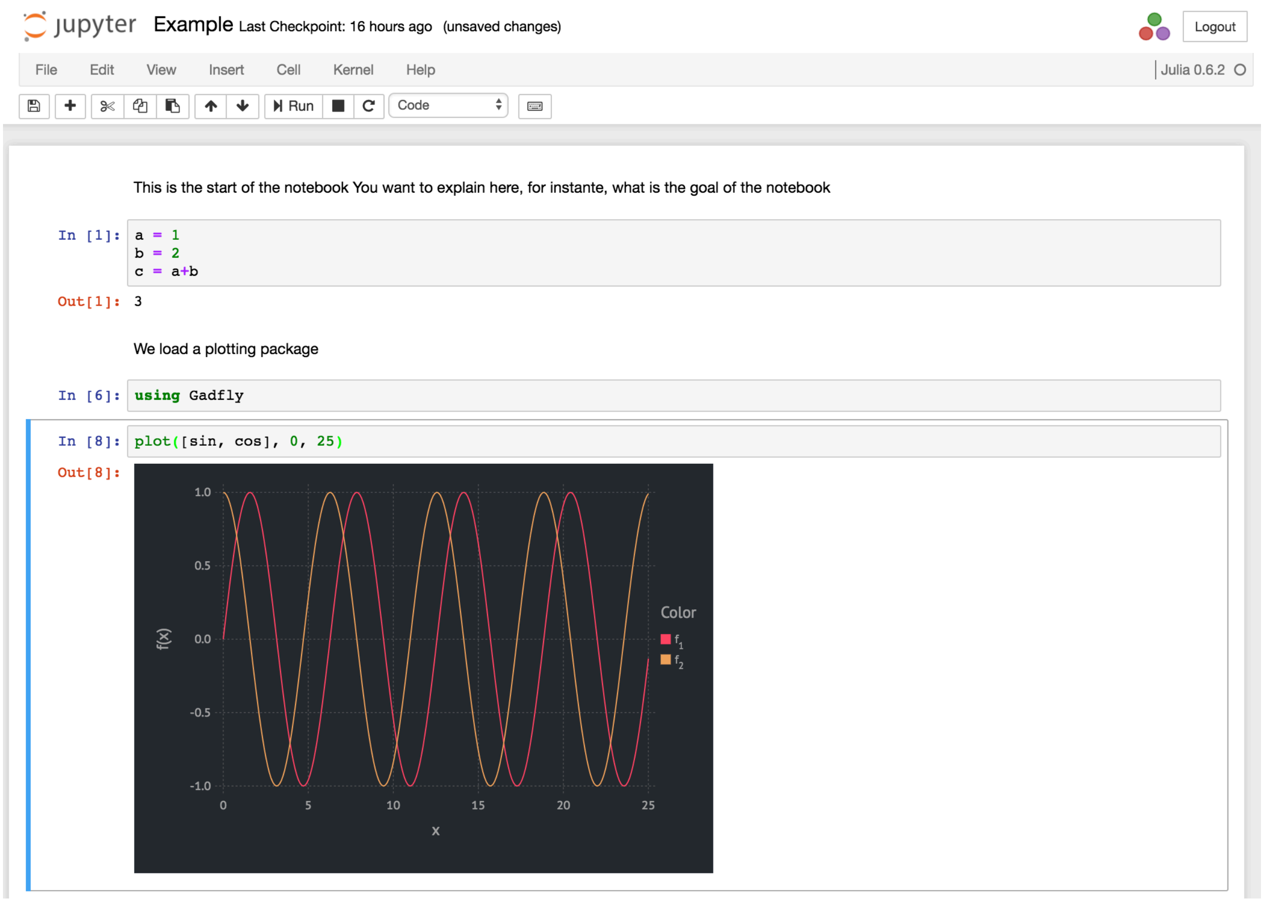
This is the start of the notebook You want to explain here, for instante, what is the goal of the notebook

```
In [1]: a = 1
        b = 2
        c = a+b

Out[1]: 3

In [ ]:
```

Figure 7.5: `More operations`

During the computation time (which, here, would be trivial but that it can take awhile in more complex evaluations), you will see `In [*]:`. There is a stop button to interrupt the evaluation if desired.

You can also run more sophisticated code, such as a creating a graph as displayed in Figure 7.6, where we load a plotting package into `Julia` and use it to plot the sine and cosine functions.


jupyter
Example Last Checkpoint: 16 hours ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Help
Julia 0.6.2
Run
Code
f(x)
x
Color
f1
f2

This is the start of the notebook You want to explain here, for instante, what is the goal of the notebook

```
In [1]: a = 1
        b = 2
        c = a+b
Out[1]: 3
```

We load a plotting package

```
In [6]: using Gadfly
```

```
In [8]: plot([sin, cos], 0, 25)
Out[8]:
```

Figure 7.6: `A first cell`

You can explore the rest of the menus and buttons yourself. You will discover buttons to insert cell, to cut, copy, and paste them, and to move them up and down. Also, you can set up a checkpoint, a state of the notebook to which you can revert if the next steps were not satisfactory for you, and some security options to avoid sharing code with malicious software embedded.

The last interesting feature is in the menu `File>Download as`. You will see a number of choice of formats to download your notebook, including `HTML`, `Markdown`, LaTeX, and `pdf` files. Generating the pdf file requires that you have all the required LaTeXpackages. In Mac, in addition, you need the commands for fonts in the terminal:

```
sudo tlmgr install adjustbox
sudo tlmgr install collection-fontsrecommended
```

and `XQuartz` installed.

Finally, you can share your notebooks on your `Github` repository with `Binder`, https://mybinder.org/.

7.2 Markdown

`Markdown` is a lightweight markup language. With many less instructions than LaTeX, it can provide you with much of the power of the later. `Markdown` allows for literate programming and report generation.

See https://daringfireball.net/projects/markdown/ for the whole syntax and documentation of `Markdown` and Mailund (2017) for a complete review.

Some basic commands. First headers (you can go up to level 6):

```
# Header level 1
## Header level 2
### Header level 3
```

Extensions `Julia`: `Weave`.

Extensions `R`: `Knitr`, `Sweave`, and `rmarkdown`.

7.3 Pandoc

See, again, Mailund (2017) and https://pandoc.org/.