Jupyter Notes

Giuseppe Giorgio Colabufo

September 6, 2023

Disclaimer

This notebook is created to provide general information and support, but there may be errors or inaccuracies in the content. If you notice any inaccuracies or issues in my notes, please report them.

It is always recommended to verify the information provided with reliable sources and consult qualified experts for specific matters. I am not responsible for any consequences arising from the use of the information contained in this notebook.

To report any errors or provide feedback, I encourage you to contact me at the specified address below:

[giuseppe.colabufo.2016@polytechnique.org]

Thank you for your understanding and cooperation in helping me improve and correct any errors in the document.

Brief Introduction

Jupyter is an interactive development environment widely used in the field of data science and data analysis. In these notes, you will find guidance on maximizing the potential of Jupyter, organizing your notebooks effectively, utilizing different code execution modes, and customizing the JupyterLab interface.

These notes are a compilation of tips, tricks, and features related to using Jupyter and JupyterLab that I have gathered over time and found useful in various situations. They represent a set of practical solutions I have discovered and experimented with to tackle specific challenges in my professional or personal journey.

Within these notes, you will find a variety of tips and ideas, ranging from small tricks that simplify daily tasks to advanced features that can enhance efficiency and user experience with specific tools or technologies. You will also come across code snippets that I have deemed particularly helpful in solving common problems or implementing specific functionalities.

This collection is an (more or less) organic compilation of information that I have updated and expanded over time. I invite you to explore these notes with curiosity and use the proposed solutions as inspiration for your specific needs. Remember, it is always important to adapt and personalize the solutions based on the context in which you apply them. I hope this information proves useful to you and inspires you on your journey. Enjoy exploring and applying these insights!

Table of Contents

[25]:
```python
from IPython.display import HTML, display

def set_background(color):
    script = (
        "var cell = this.closest('.jp-CodeCell');"
        "var editor = cell.querySelector('.jp-Editor');"
        "editor.style.background='{}';"
        "this.parentNode.removeChild(this)"
    ).format(color)

    display(HTML('<img src onerror="{}" style="display:none">'.format(script)))

from IPython.core.magic import register_cell_magic

@register_cell_magic
def background(color, cell):
    set_background(color)
    return eval(cell)
```

1 Jupyter tricks

1.1 Import local Python module

Just use the following:

```
import sys
sys.path.append("/path/to/my_folder")
import my_module
```

if `my_module.py` is in folder `my_folder`.

Note that in path you should use "/" and that "", which is an escape character is wrong.

1.2 Add blank page when exporting to pdf

Simply add `\pagebreak` in a Markdown cell whenever you want to insert a blank page. It will work since Jupyter uses LaTeX to convert notebooks to pdf. Indeed, `\pagebreak` is actually a LaTeX command, rather than a Markdown one. If you want pdf to go to a new page at the beginning of every chapter, it can be simpler to add `\pagebreak` command in the same cell containing the heading, before the heading itself.

```
\pagebreak
# My Heading
```

Source: StackOverflow - pagebreak in markdown while creating pdf.

1.3 Suppress output in Jupyter

Add `%%capture` as the first line of the cell; eg:

[15]:
```
%%capture
print('Hello')
```

This simply discards the output.

Warning! Code will be executed in any case, so this trick is not worthy to save time. If you want to avoid code execution, set cell type as 'Raw'.

1.4 Color cells in Jupyter

1.4.1 Python cells

Add the following code:

```
from IPython.display import HTML, display

def set_background(color):
    script = (
        "var cell = this.closest('.jp-CodeCell');"
        "var editor = cell.querySelector('.jp-Editor');"
        "editor.style.background='{}';"
        "this.parentNode.removeChild(this)"
    ).format(color)
    display(HTML('<img src onerror="{}" style="display:none">'.format(script)))

from IPython.core.magic import register_cell_magic

@register_cell_magic
def background(color, cell):
    set_background(color)
    return eval(cell)
```

And use it like this:

```
%%bgc yellow
'bla'
```

which will result in:

```
[40]: %%bgc yellow
      'bla'
```

```
<IPython.core.display.HTML object>
```

> Apparently you should then remove the cell content and write the desired code. Otherwise it will not be executed and an exception will be raised.

To set the color back to the original one, simply use

```
%%bgc
'bla'
```

Source: StackOverflow: How to change the background color of a single cell in a jupyter notebook / jupyterlab?

1.4.2 Markdown cells

You could do this:

```
<div class="alert-success">
This is a green colored box
</div>

<div class="alert-danger">
This is a red colored box
</div>

<div class="alert-warning">
This is a yellow colored box
</div>

<div class="alert-info">
This is a blue colored box
</div>
```

and obtain

This is a green colored box

This is a red colored box

This is a yellow colored box

This is a blue colored box

Sources: - StackOverflow: How to change the color of text in markdown cell in JupyterLab - IBM Knowledge Center: Markdown for Jupyter notebooks cheatsheet

1.5 Display large dataframe with scrollbar

Import the following:

```
from IPython.display import display, HTML
```

and then use the following command to display the dataframe `df`:

```
# Puts the scrollbar next to the DataFrame
display(
    HTML(
        "<div style='height: 300px; overflow: auto; width: fit-content'>" +
        df.to_html() +
        "</div>"
    )
)
```

Note that you can change height value to your needs (300px is what I used the most).

Here's a complete example to show the result:

```
[4]: import pandas as pd
     from IPython.core.display import display, HTML

     # Sample DataFrame
     data = {
         'Name': ['Alice', 'Bob', 'Charlie', 'David'],
         'Age': [25, 30, 22, 28],
         'Country': ['USA', 'Canada', 'UK', 'Australia']
     }

     df = pd.DataFrame(data)

     # Display DataFrame using HTML formatting
     display(
         HTML(
             "<div style='height: 100px; overflow: auto; width: fit-content'>" +
             df.to_html() +
             "</div>"
         )
)
```

```
<IPython.core.display.HTML object>
```

Source: How to display large matrix in pandas with scroll bars in Jupyter Notebook?.

You can also use this command to show a better formatted dataframe instead of using `print(df)`.

1.6 Interactive plots

1.6.1 Interactive zoom in plots

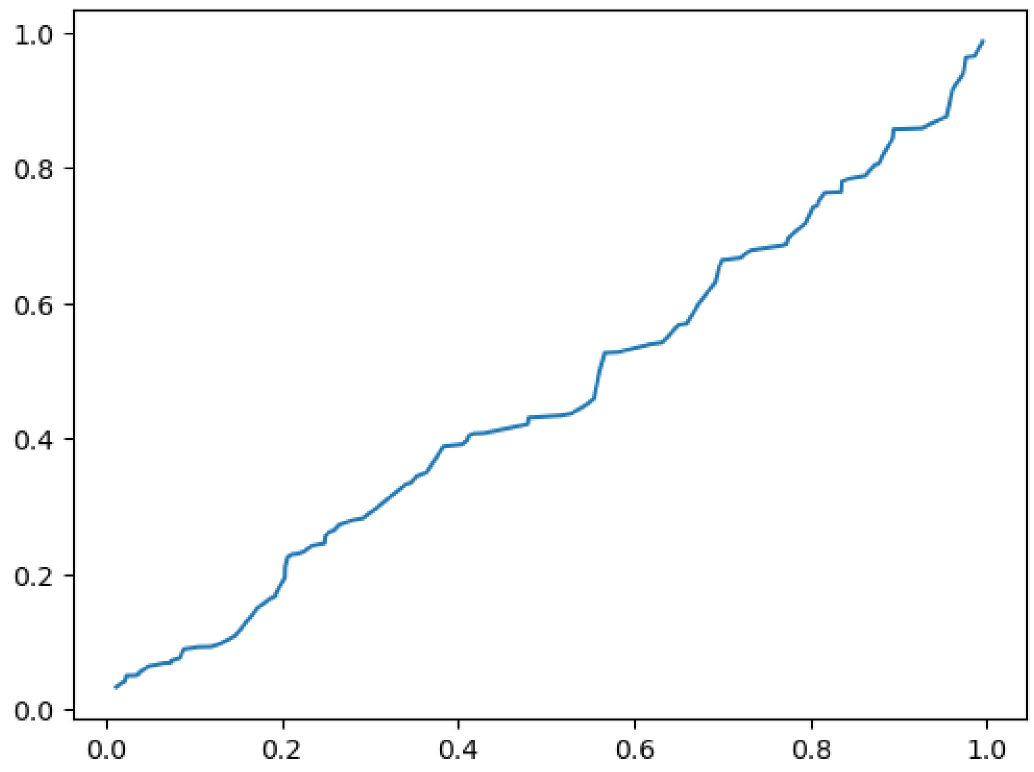
It is possible to zoom when inline plots are activated, through `mpld3` package. First, install it with `!pip install mpld3`, then add to the notebook:

[13]:
```
%matplotlib inline
import mpld3
mpld3.enable_notebook()
```

Here's an example:

[14]:
```
import matplotlib.pyplot as plt
import numpy as np
df = np.random.rand(100, 100)
df.sort()
plt.plot(df[0], df[1])
```

[14]:
```
[<matplotlib.lines.Line2D at 0x2189fc2b8b0>]
```



Source: StackOverflow ipython notebook –pylab inline: zooming of a plot.

1.6.2 More interactive zoom in plots

The `plotly` library, which provides more advanced interactive features including zooming and panning. Here's an example of how you can achieve this using `plotly`:

```
import numpy as np
import plotly.express as px

# Generate some sample data
x = np.linspace(0, 10, 100)
y = np.sin(x)

# Create a plotly figure
fig = px.line(x=x, y=y)

# Update the layout to enable zooming and panning
fig.update_layout(
    xaxis=dict(
        rangeselector=dict(
            buttons=list([
                dict(count=1, label="1d", step="day", stepmode="backward"),
                dict(count=7, label="1w", step="day", stepmode="backward"),
                dict(count=1, label="1m", step="month", stepmode="backward"),
                dict(step="all")
            ])
        ),
        rangeslider=dict(visible=True),
        type="date"
    )
)

# Display the interactive plot
fig.show()
```

In this example, we use `plotly.express` to create a line plot, and then we use the `update_layout` method to configure the x-axis with a range selector and rangeslider for zooming and panning interactions.

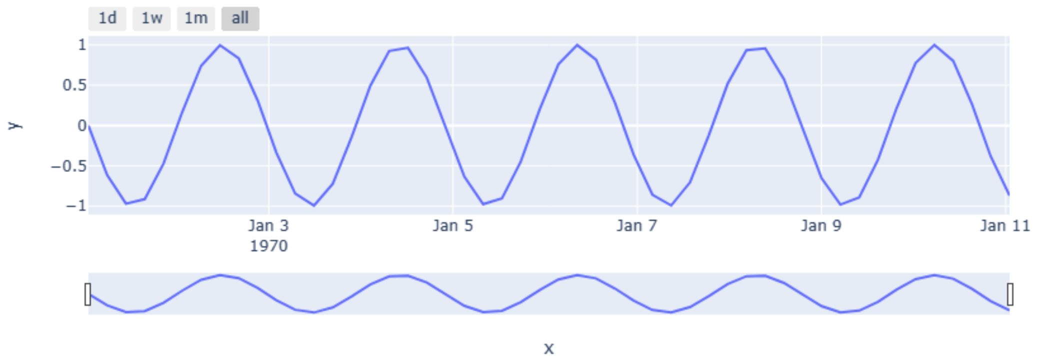
[44]:
```
import numpy as np
import plotly.express as px

# Generate some sample data
x = np.linspace(0, 100*60*60*24*100)
y = np.sin(x)

# Create a plotly figure
fig = px.line(x=x, y=y)

# Update the layout to enable zooming and panning
fig.update_layout(
    xaxis=dict(
        rangeselector=dict(
            buttons=list([
                dict(count=1, label="1d", step="day", stepmode="backward"),
                dict(count=7, label="1w", step="day", stepmode="backward"),
                dict(count=1, label="1m", step="month", stepmode="backward"),
                dict(step="all")
            ])
        ),
        rangeslider=dict(visible=True),
        type="date"
    )
)

# Display the interactive plot
fig.show()
```


1d
1w
1m
all
y
1
0.5
0
−0.5
−1
Jan 3
1970
Jan 5
Jan 7
Jan 9
Jan 11
x

2 Markdown tricks

2.1 Comments

Write a comment in Markdown, i.e. text that is not rendered in the HTML output. If you want a comment that is strictly for yourself (readers of the converted document should not be able to see it, even with "view source") you could (ab)use the link labels (for use with reference style links) that are available in the core Markdown specification:

```
--- Comments follows: ---

[comment]: <> (This is a comment, it will not be included)
[comment]: <> (in  the output file unless you use it in)
[comment]: <> (a reference style link.)

--- End of comments ---
```

Or you could go further:

```
[//]: <> (This is also a comment.)
```

To improve platform compatibility (and to save one keystroke) it is also possible to use `#` (which is a legitimate hyperlink target) instead of `<>`:

```
[//]: # (This may be the most platform independent comment)
```

For maximum portability it is important to insert a blank line before and after this type of comments, because some Markdown parsers do not work correctly when definitions brush up against regular text. The most recent research with Babelmark shows that blank lines before and after are both important. Some parsers will output the comment if there is no blank line before, and some parsers will exclude the following line if there is no blank line after.

In general, this approach should work with most Markdown parsers, since it's part of the core specification. (even if the behavior when multiple links are defined, or when a link is defined but never used, is not strictly specified).

Sources: StackOverflow - Comments in Markdown

2.2 Footnotes

```
Example of footnote.<a name="cite_ref-1"></a>[<sup>[1]</sup>](#cite_note-1)
Example of footnote.<a name="cite_ref-2"></a>[<sup>[2]</sup>](#cite_note-2)



<a name="cite_note-1"></a>1. [^](#cite_ref-1) footnote 1 <br/>
<a name="cite_note-2"></a>2. [^](#cite_ref-2) footnote 2 <br/>
```

Example of footnote.[1] Example of footnote.[2]

1. [^](#cite_ref-1) footnote 1 2. [^](#cite_ref-2) footnote 2

2.3 Reducing font size

```
<sub><sup>Combining the two tags, you will get smaller text:</sup></sub>
```

Combining the two tags, you will get smaller text.

2.4 Tables

The first row of the table defines the headers, then the next row defines the alignment of each column. You duplicated the alignment at the top of the table and where it's actually supposed to go.

The right Markdown should simply be what you have in your syntax, but remove the first row:

```
| Stretch/Untouched | ProbDistribution | Accuracy |
| --- | --- | --- |
| Stretched | Gaussian | .843 |
```

The `---` in between the column definitions `| |` mean that the column is unjustified. In standard Markdown, this would align to the left of the column but in Jupyter notebook, it appears to align to the right instead.

With that, I get this table:

Stretch/Untouched	ProbDistribution	Accuracy
Stretched	Gaussian	.843

If you'd like to left align or centre align, you can use `:-` and `:-:` respectively. Depending on what Jupyter notebook environment you're using, you will need to use `-:` to right align.

Stretch/Untouched	ProbDistribution	Accuracy
Stretched	Gaussian	.843

The first column will be left-aligned, the centre column is right-aligned and the last column is centre aligned.

2.4.1 Tables side-by-side

To obtain two tables side-by-side, use the code:

```
<table>
<tr><th>Table 1 Heading 1 </th><th>Table 1 Heading 2</th></tr>
<tr><td>

|Table 1| Middle | Table 2|
|--|--|--|
|a| not b|and c |

</td><td>

|b|1|2|3|
|--|--|--|--|
|a|s|d|f|
```

```
</td></tr> </table>
```

and you will obtain the following result:

Table 1 Heading 1

Table 1 Heading 2

Table 1	Middle	Table 2
a	not b	and c

b	1	2	3
a	s	d	f

2.5 External resources

Follows a list of interesting blog posts and papers.

- Markdown guide
  - Basic syntax
  - Tables, code blocks, footnotes, definition/task lists, emoji, sub/super-script, URL
  - Indent, color, comments, admonitions, images, symbols, tables
  - Quick cheat sheet
  - Complete list of github markdown emoji markup

3 LaTeX in Jupyter

Yes, you can write in LaTeX in Jupyter notebooks using Markdown cells. Markdown cells in Jupyter notebooks support LaTeX syntax for mathematical equations and symbols. Here's how you can do it:

1. Create a new Markdown cell in your Jupyter notebook.
2. Use the dollar sign ()*toencloseLaTeXcodeforinlineequations.Forexample* :`E=mc^2$`.
3. Use double dollar signs ($$) to enclose LaTeX code for equations on a separate line. For example:

```
$$
\frac{d}{dx} \left( \int_{a}^{x} f(t) \, dt \right) = f(x)
$$
```

4. After writing your LaTeX code, run the cell to render the equations.

Keep in mind that Markdown cells support a subset of LaTeX syntax, primarily focused on mathematical equations. If you need more advanced LaTeX formatting, you might consider using LaTeX in conjunction with other tools or converting the notebook to a LaTeX document using tools like nbconvert.

For instance, the above code would render as such:

$$\frac{d}{dx}\left(\int_a^x f(t)\,dt\right) = f(x)$$

while the inline equation $E = mc^2$.

```
[ ]:
```