3 Great Additions for your Jupyter Notebooks

June 28, 2019

I love Jupyter notebooks and the power they provide.

They can be used to present findings as well as share code in the most effective manner which was not easy with the previous IDEs.

Yet there is something still amiss.

There are a few functionalities I aspire in my text editor which don't come by default in Jupyter.

But fret not. Just like everything in Python, Jupyter too has third-

Buy me a coffee

SEARCH...

RECENT POSTS

The 5 Sampling Algorithms every Data Scientist need to know

Bayesian Bandits explained simply

Minimal Pandas Subset for Data Scientists

The Hitchhikers guide to handle Big Data using Spark

3 Great Additions for your Jupyter Notebooks

An End to End Introduction to GANs

The Hitchhiker's Guide to Feature Extraction

The Nation of a Billion Votes

party extensions.

This post is about some of the most useful extensions I found.

1. Collapsible Headings

The one extension, I like most is collapsible headings.

It makes the flow of the notebook easier to comprehend and also helps in creating presentable notebooks.

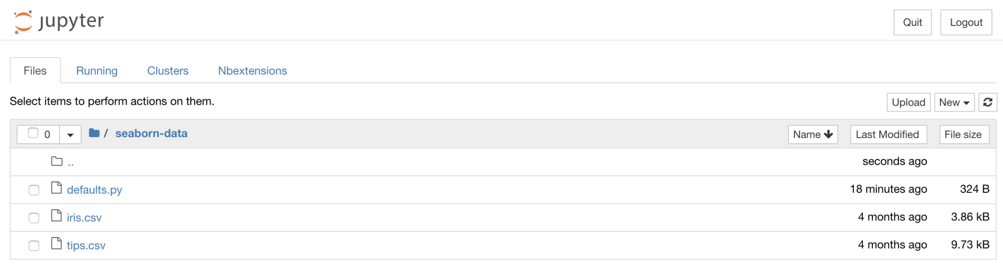
To get this one, install the `jupyter_contrib_nbextensions` package with this command on the terminal window:

```
conda install -c conda-forge jupyter_contrib_nbextensions
```

Once the package is installed, we can start jupyter notebook using:

```
jupyter notebook
```

Once you go to the home page of your jupyter notebook, you can see that a new tab for NBExtensions is created.



Select items to perform actions on them.

```
0  / seaborn-data
..                          seconds ago
defaults.py                 18 minutes ago   324 B
iris.csv                    4 months ago     3.86 kB
tips.csv                    4 months ago     9.73 kB
```

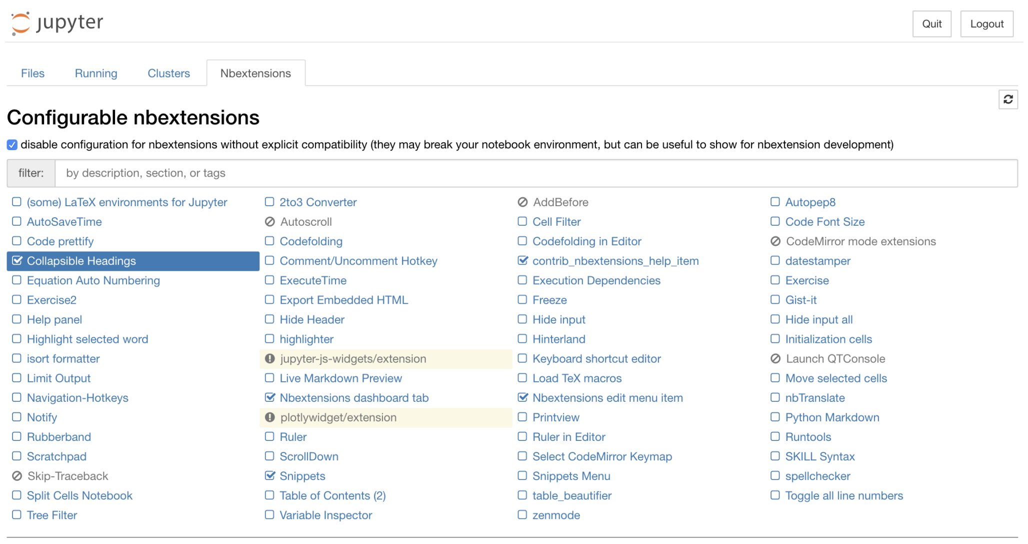
And we can get a lot of extensions using this package.

A primer on *args, **kwargs, decorators for Data Scientists

Python's One Liner graph creation library with animations Hans Rosling Style

email address

SUBSCRIBE



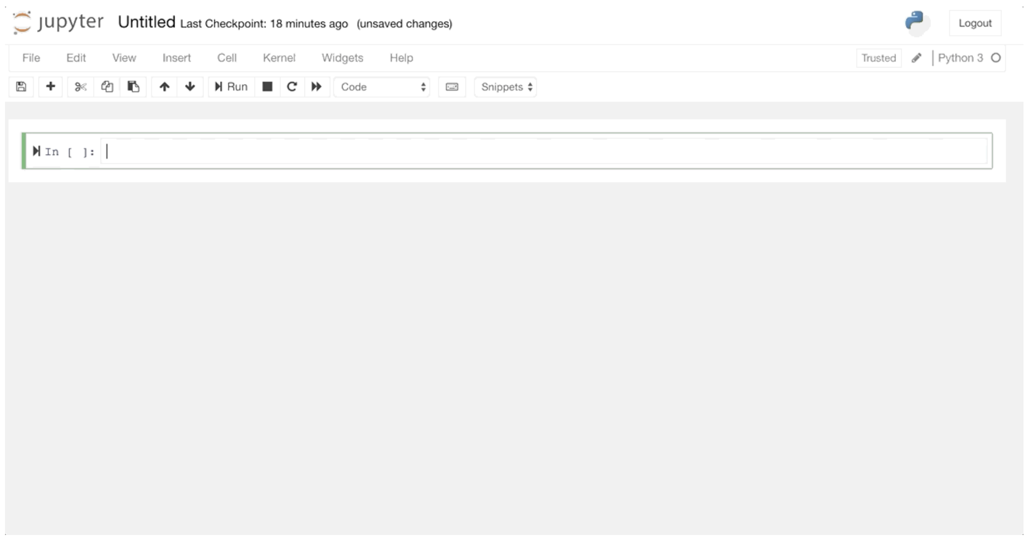
Configurable nbextensions

☑ disable configuration for nbextensions without explicit compatibility (they may break your notebook environment, but can be useful to show for nbextension development)

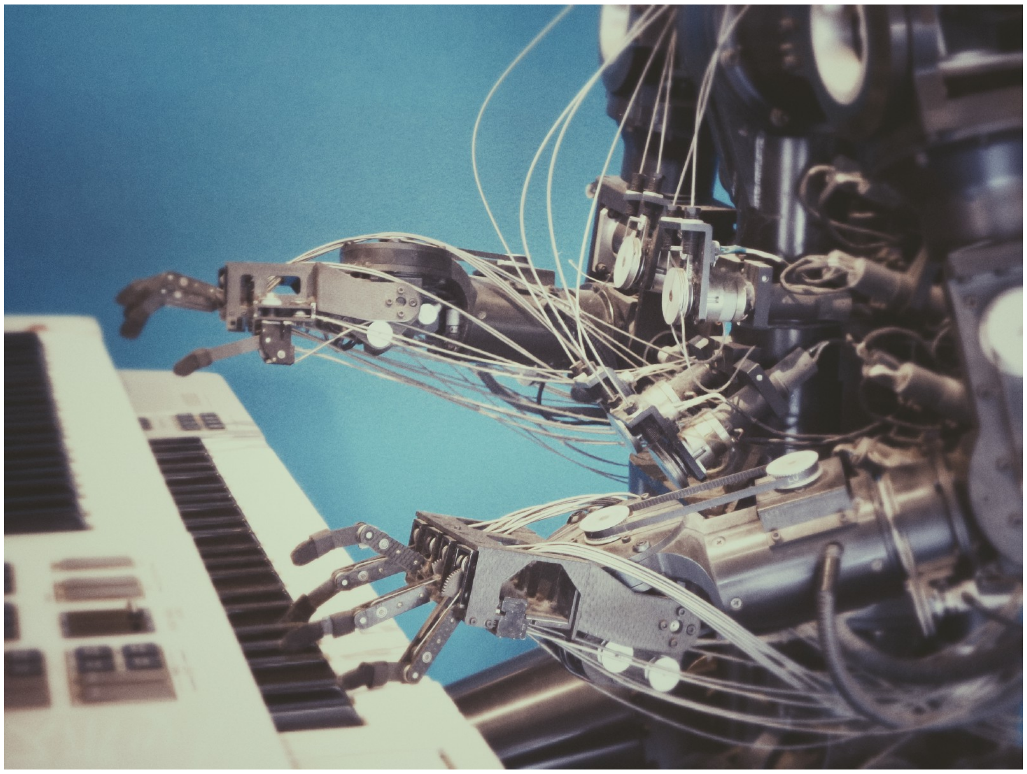
filter: by description, section, or tags

☐ (some) LaTeX environments for Jupyter	☐ 2to3 Converter	⊘ AddBefore	☐ Autopep8
☐ AutoSaveTime	⊘ Autoscroll	☐ Cell Filter	☐ Code Font Size
☐ Code prettify	☐ Codefolding	☐ Codefolding in Editor	⊘ CodeMirror mode extensions
☑ Collapsible Headings	☐ Comment/Uncomment Hotkey	☑ contrib_nbextensions_help_item	☐ datestamper
☐ Equation Auto Numbering	☐ ExecuteTime	☐ Execution Dependencies	☐ Exercise
☐ Exercise2	☐ Export Embedded HTML	☐ Freeze	☐ Gist-it
☐ Help panel	☐ Hide Header	☐ Hide input	☐ Hide input all
☐ Highlight selected word	☐ highlighter	☐ Hinterland	☐ Initialization cells
☐ isort formatter	❗ jupyter-js-widgets/extension	☐ Keyboard shortcut editor	⊘ Launch QTConsole
☐ Limit Output	☐ Live Markdown Preview	☐ Load TeX macros	☐ Move selected cells
☐ Navigation-Hotkeys	☑ Nbextensions dashboard tab	☑ Nbextensions edit menu item	☐ nbTranslate
☐ Notify	❗ plotlywidget/extension	☐ Printview	☐ Python Markdown
☐ Rubberband	☐ Ruler	☐ Ruler in Editor	☐ Runtools
☐ Scratchpad	☐ ScrollDown	☐ Select CodeMirror Keymap	☐ SKILL Syntax
⊘ Skip-Traceback	☑ Snippets	☐ Snippets Menu	☐ spellchecker
☐ Split Cells Notebook	☐ Table of Contents (2)	☐ table_beautifier	☐ Toggle all line numbers
☐ Tree Filter	☐ Variable Inspector	☐ zenmode	

This is how it looks:



2. Automatic Imports



Automation is the future.

One thing that bugs me is that whenever I open a new Jupyter notebook in any of my data science projects, I need to copy paste a lot of libraries and default options for some of them.

To tell you about some of the usual imports I use:

- Pandas and numpy — In my view, Python must make these two as a default import.
- Seaborn, matplotlib, plotly_express
- change some pandas and seaborn default options.

Here is the script that I end up pasting over and over again.

```
import pandas as pd
import numpy as np

import plotly_express as px
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

```
*# We dont Probably need the Gridlines. Do we? If yes comment
this line*
sns.set(style="ticks")

# pandas defaults
pd.options.display.max_columns = 500
pd.options.display.max_rows = 500
```

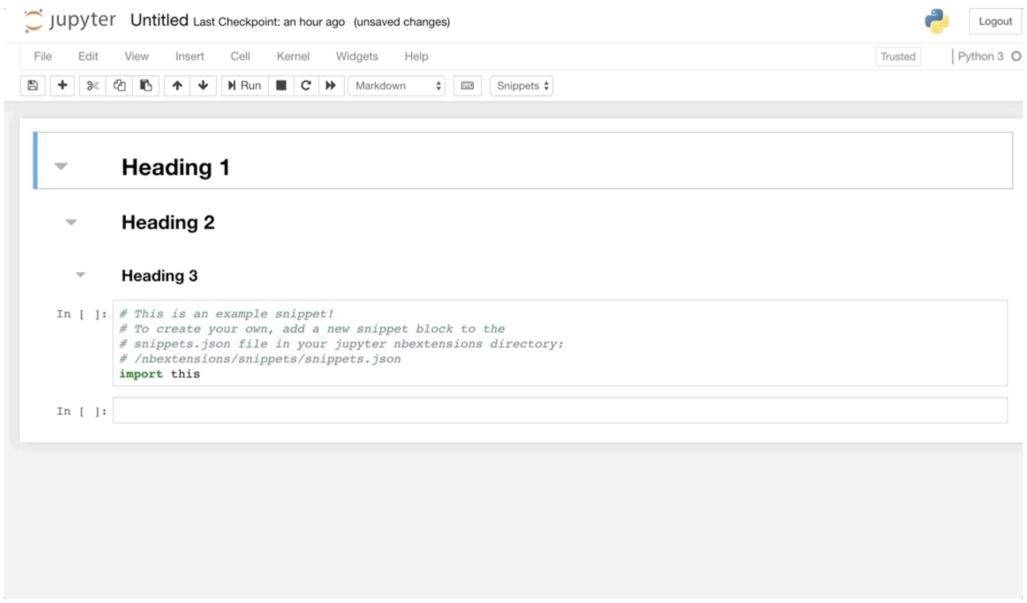
Is there a way I can automate this?

Just go to the nbextensions tab and select the snippets extension.

You will need to make the following changes to the snippets.json file. You can find this file at `/miniconda3/envs/py36/share/jupyter/nbextensions/snippets` location. The py36 in this location here is my conda virtualenv. It took me some time to find this location for me. Yours might be different. Please note that you don't have to change at the site-packages location.

```
{
    "snippets" : [
        {
            "name" : "example",
            "code" : [
                "# This is an example snippet!",
                "# To create your own, add a new snippet block to
the",
                "# snippets.json file in your jupyter
nbextensions directory:",
                "# /nbextensions/snippets/snippets.json",
                "import this"
            ]
        },

        {
            "name" : "default",
            "code" : [
                "# This is A snippet for all data related tasks",
                "import pandas as pd"
                "import numpy as np"
                "import plotly_express as px"
                "import seaborn as sns"
                "import matplotlib.pyplot as plt"

                "%matplotlib inline"
                "# We dont Probably need the Gridlines. Do we? If
yes comment this line"
                "sns.set(style='ticks')"
                "# pandas defaults"
                "pd.options.display.max_columns = 500"
                "pd.options.display.max_rows = 500"
            ]
        }
    ]
}
```

You can see this extension in action below.



```
In [ ]: # This is an example snippet!
        # To create your own, add a new snippet block to the
        # snippets.json file in your jupyter nbextensions directory:
        # /nbextensions/snippets/snippets.json
        import this

In [ ]:
```

Pretty cool. Right? I also use this to create basic snippets for my deep learning notebooks and NLP based notebooks.

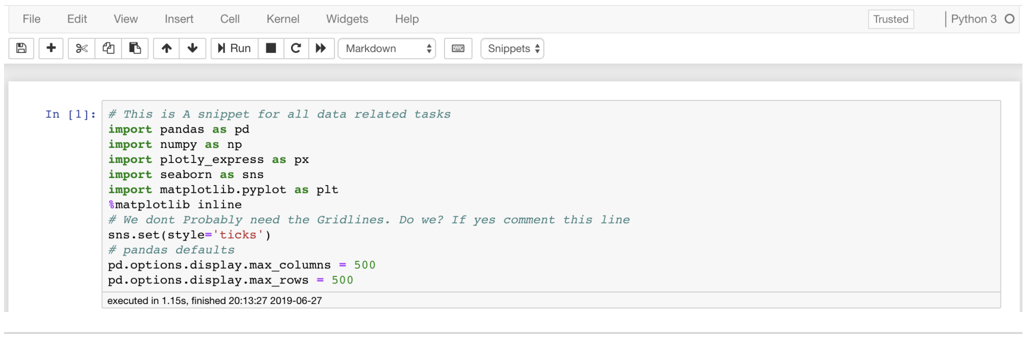
---

3. Execution Time

We have used `%time` as well as decorator based timer functions to measure time for our functions. You can also use this excellent extension to do that.

Plus it looks great.

Just select the ExecutionTime extension from the NBextensions list and you will have an execution result at the bottom of the cell after every cell execution as well as the time when the cell was executed every cell execution as well as the time when the cell was executed.

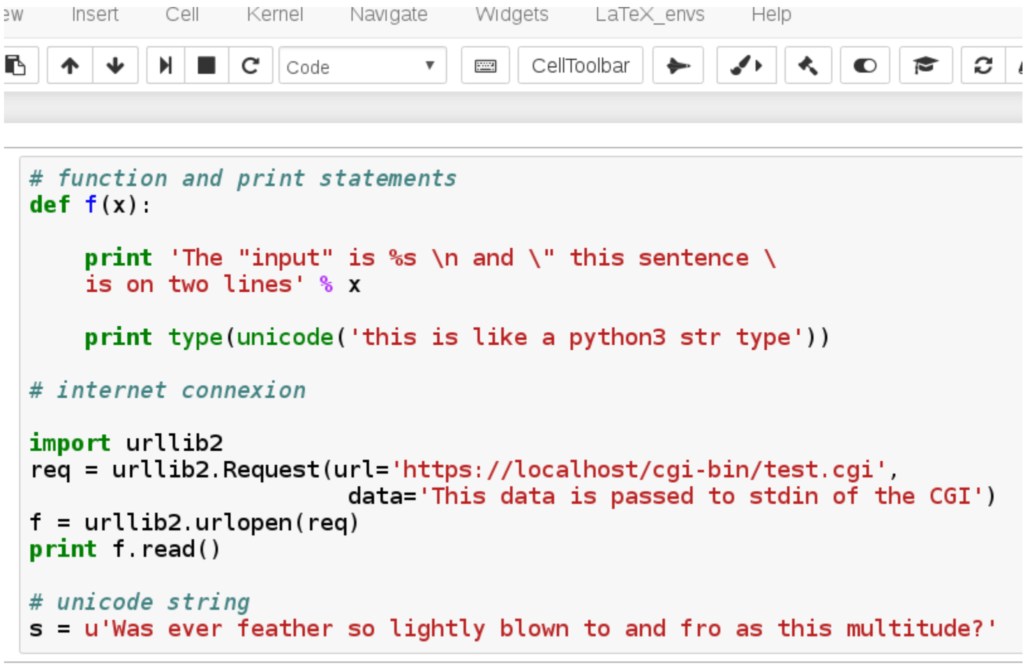


```
In [1]: # This is A snippet for all data related tasks
        import pandas as pd
        import numpy as np
        import plotly_express as px
        import seaborn as sns
        import matplotlib.pyplot as plt
        %matplotlib inline
        # We dont Probably need the Gridlines. Do we? If yes comment this line
        sns.set(style='ticks')
        # pandas defaults
        pd.options.display.max_columns = 500
        pd.options.display.max_rows = 500
        executed in 1.15s, finished 20:13:27 2019-06-27
```

Other Extensions

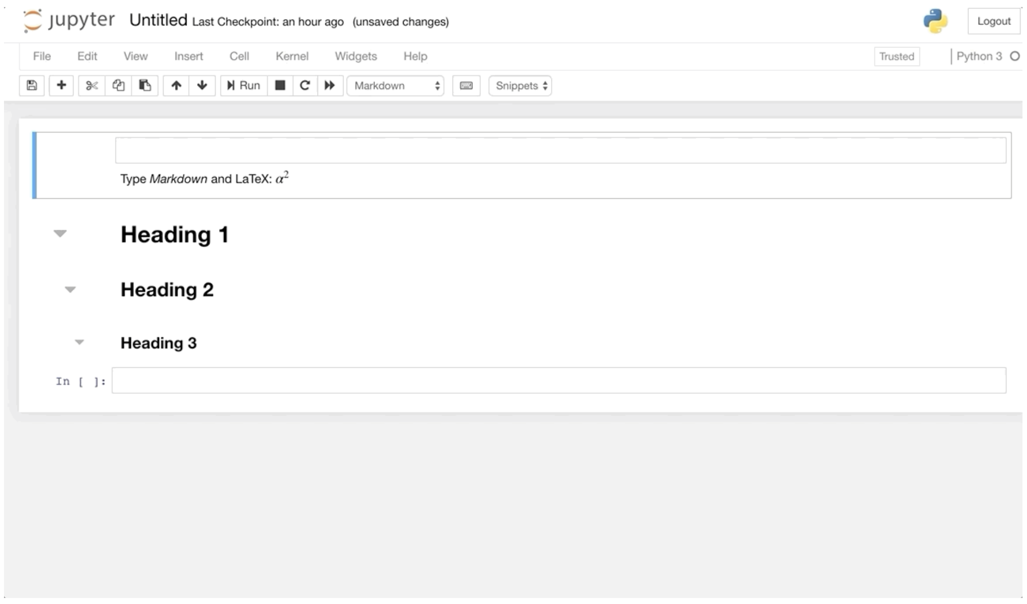
NBExtensions has a lot of extensions. Some other extensions from NBExtensions I like and you might want to look at:

- **Limit Output:** Ever had your notebook hang since you printed a lot of text in your notebook. This extension limits the number of characters that can be printed below a code cell
- **2to3Convertor:** Having problems with your old python2 notebooks. Tired of changing the print statements. This one is a good one.



```
# function and print statements
def f(x):

    print 'The "input" is %s \n and \" this sentence \
    is on two lines' % x

    print type(unicode('this is like a python3 str type'))

# internet connexion

import urllib2
req = urllib2.Request(url='https://localhost/cgi-bin/test.cgi',
                      data='This data is passed to stdin of the CGI')
f = urllib2.urlopen(req)
print f.read()

# unicode string
s = u'Was ever feather so lightly blown to and fro as this multitude?'
```

- **Live Markdown Preview:** Some of us like writing our blogs using Markdown in a jupyter notebook. Sometimes it can be hectic as you make errors in writing. Now you can see Live-preview of the rendered output of markdown cells while editing their source.


jupyter Untitled Last Checkpoint: an hour ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Markdown
Snippets
Type *Markdown* and LaTeX: α^2
Heading 1
Heading 2
Heading 3
In [ ]:

Conclusion

I love how there is a package for everything with Python. And that holds good with the Jupyter notebook too.

The `jupyter_contrib_nbextensions` package works great out of the box.

It has made my life a lot easier when it comes to checking execution times, scrolling through the notebook, and repetitive tasks.

There are many other extensions this package does provide. Do take a look at them and try to see which ones you find useful.

Also, if you want to learn more about Python 3, I would like to call out an excellent course on Learn **Intermediate level Python** from the University of Michigan. Do check it out.

I am going to be writing more of such posts in the future too. Let me know what you think about the series. Follow me up at **Medium** or Subscribe to my **blog** to be informed about them. As always, I welcome feedback and constructive criticism and can be reached on Twitter **@mlwhiz**.

PYTHON

Deep Learning Specialization on Coursera

« PREVIOUS
An End to End Introduction to GANs

NEXT »
The Hitchhikers guide to handle Big Data using Spark