for Data Science

Enhance Jupyter Notebook's productivity with these Tips & Tricks.

Parul Pandey Follow
Dec 19 · 9 min read

Photo by Aaron Burden on Unsplash

Reimagining what a Jupyter notebook can be and what can be done with it.

Netflix aims to provide personalized content to their 130 million viewers. One of the significant ways by which data scientists and engineers at Netflix interact with their data is through **Jupyter notebooks**. Notebooks leverage the use of collaborative, extensible, scalable, and reproducible data science. For many of us, Jupyter Notebooks is the **de facto** platform when it comes to quick prototyping and exploratory analysis. However, there's more to this than meets the eye. A lot of Jupyter functionalities sometimes lies under the hood and is not adequately explored. Let us try and explore Jupyter Notebooks' features which can enhance our productivity while working with them.

. . .

Table of Contents

1. Executing Shell Commands
2. Jupyter Themes
3. Notebook Extensions
4. Jupyter Widgets
5. Qgrid
6. Slideshow
7. Embedding URLs, PDFs, and Youtube Videos

. . .

1. Executing Shell Commands

The notebook is the new shell

The shell is a way to interact textually with the computer. The most popular Unix shell is Bash(***Bourne Again SHell***). Bash is the default shell on most modern implementations of Unix and in most packages that provide Unix-like tools for Windows.

Now, when we work with any Python interpreter, we need to regularly switch between the shell and the IDLE, in case we need to use the command line tools. However, the Jupyter Notebook gives us the ease to execute shell commands from within the notebook by placing an extra `!` before the commands. Any command that works at the command-line can be used in IPython by prefixing it with the `!` character.

```
In [1]: !ls
example.jpeg list tmp
```

```
In [2]: !pwd
/home/Parul/Desktop/Hello World Folder'
```

```
In [3]: !echo "Hello World"
Hello World
```

We can even pass values to and from the shell as follows:

```
In [4]: files= !ls

In [5]: print(files)
['example.jpeg', 'list', 'tmp']

In [6]: directory = !pwd

In [7]: print(directory)
['/Users/Parul/Desktop/Hello World Folder']

In [8]: type(directory)
IPython.utils.text.SList
```

Notice, the data type of the returned results is not a list.

. . .

2. Jupyter Themes

Theme-ify your Jupyter Notebooks!

If you are a person who gets bored while staring at the white background of the Jupyter notebook, themes are just for you. The themes also enhance the presentation of the code. You can find more about Jupyter themes here. Let's get to the working part.

Installation

```
pip install jupyterthemes
```

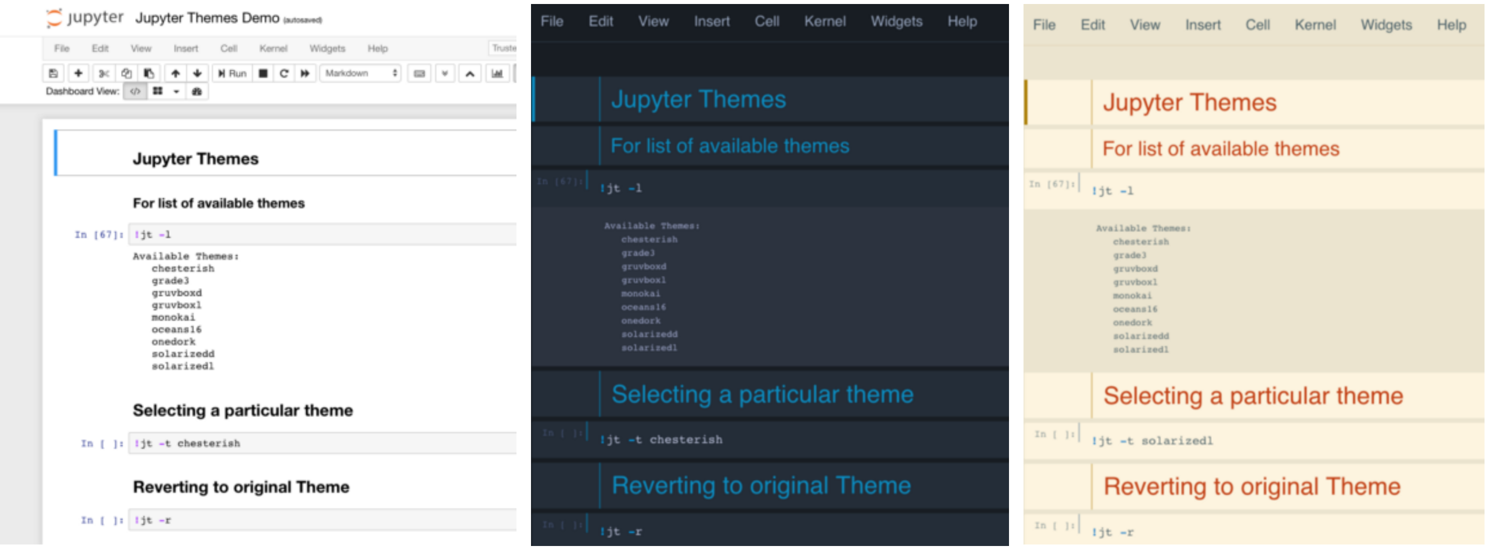
List of available themes

```
jt –l
```

Currently, the available themes are **chesterish, grade3, gruvboxd, gruvboxl monokai, oceans16, onedork, solarizedd ,solarizedl.**

```
# selecting a particular theme

jt –t <name of the theme>

# reverting to original Theme

jt –r
```

- *You will have to reload the jupyter notebook everytime you change the theme, to see the effect take place.*
- *The same commands can also be run from within the Jupyter Notebook by placing '* `!` *' before the command.*


jupyter Jupyter Themes Demo (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Run Markdown
Dashboard View:

Jupyter Themes

For list of available themes

```
In [67]: !jt -l
Available Themes:
   chesterish
   grade3
   gruvboxd
   gruvboxl
   monokai
   oceans16
   onedork
   solarizedd
   solarizedl
```

Selecting a particular theme

```
In [ ]: !jt -t chesterish
```

Reverting to original Theme

```
In [ ]: !jt -r
```

File Edit View Insert Cell Kernel Widgets Help

Jupyter Themes

For list of available themes

```
In [67]: !jt -l
Available Themes:
   chesterish
   grade3
   gruvboxd
   gruvboxl
   monokai
   oceans16
   onedork
   solarizedd
   solarizedl
```

Selecting a particular theme

```
In [ ]: !jt -t chesterish
```

Reverting to original Theme

```
In [ ]: !jt -r
```

Jupyter Themes

For list of available themes

```
In [67]: !jt -l
Available Themes:
   chesterish
   grade3
   gruvboxd
   gruvboxl
   monokai
   oceans16
   onedork
   solarizedd
   solarizedl
```

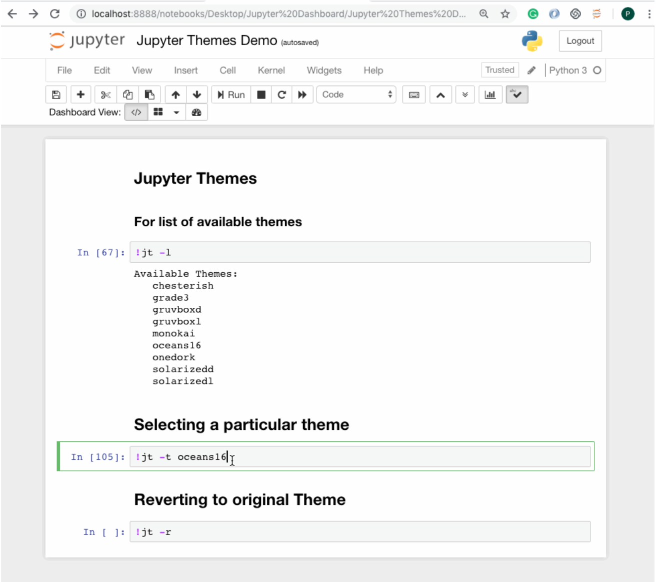
Selecting a particular theme

```
In [ ]: !jt -t solarizedl
```

Reverting to original Theme

```
In [ ]: !jt -r
```

Left: **original** | Middle: **Chesterish** Theme | Right: **solarizedl** theme


localhost:8888/notebooks/Desktop/Jupyter%20Dashboard/Jupyter%20Themes%20D...
jupyter Jupyter Themes Demo (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run Code
Dashboard View:

Jupyter Themes

For list of available themes

```
In [67]: !jt -l
Available Themes:
   chesterish
   grade3
   gruvboxd
   gruvboxl
   monokai
   oceans16
   onedork
   solarizedd
   solarizedl
```

Selecting a particular theme

```
In [105]: !jt -t oceans16
```

Reverting to original Theme

```
In [ ]: !jt -r
```

. . .

3. Notebook Extensions

Extend the possibilities

Notebook extensions let you move beyond the general vanilla way of using the Jupyter Notebooks. Notebook extensions (or nbextensions) are JavaScript modules that you can load on most of the views in your

Notebook's frontend. These extensions modify the user experience and interface.

Installation

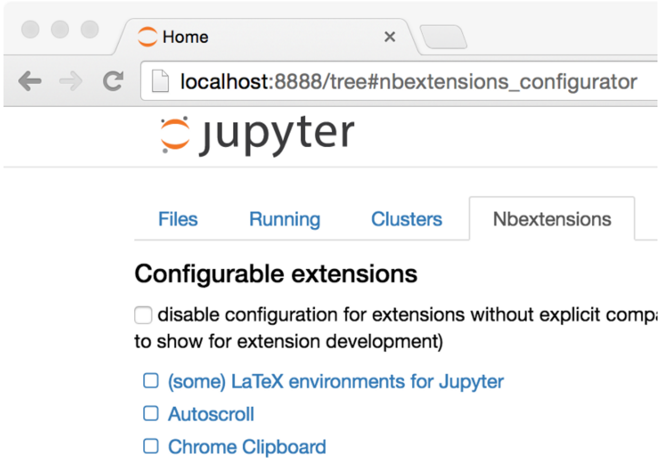
Installation with conda:

```
conda install -c conda-forge
jupyter_nbextensions_configurator
```

Or with pip:

```
pip install jupyter_contrib_nbextensions && jupyter contrib
nbextension install

#incase you get permission errors on MacOS,

pip install jupyter_contrib_nbextensions && jupyter contrib
nbextension install --user
```

Start a Jupyter notebook now, and you should be able to see an **NBextensions Tab** with a lot of options. Click the ones you want and see the magic happen.

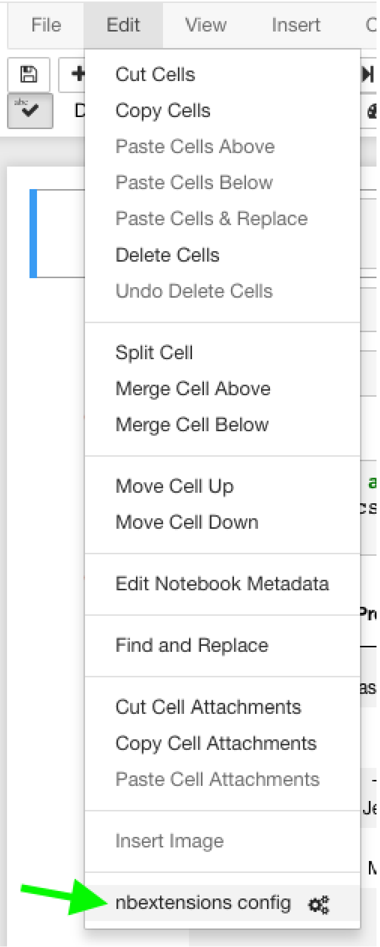


Configurable extensions

□ disable configuration for extensions without explicit compa
to show for extension development)

□ (some) LaTeX environments for Jupyter
□ Autoscroll
□ Chrome Clipboard

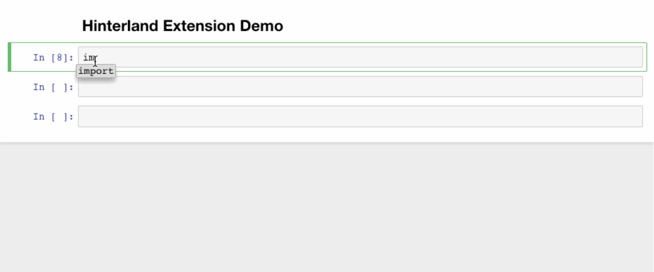
In case you couldn't find the tab, a second small nbextension, can be located under the menu `Edit` .



Let us discuss some of the useful extensions.

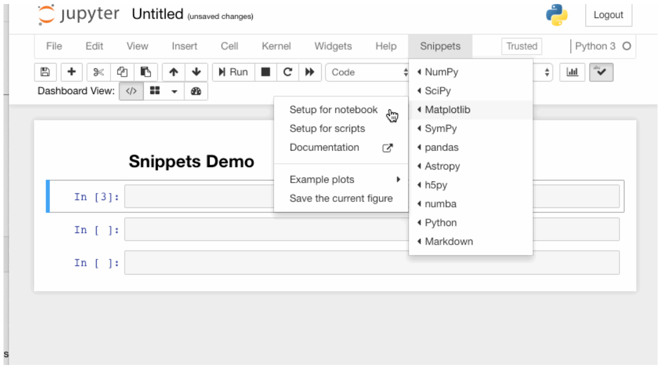
1. Hinterland

Hinterland enables code autocompletion menu for every keypress in a code cell, instead of only calling it with the tab. This makes Jupyter notebook's autocompletion behave like other popular IDEs such as PyCharm.



2. Snippets

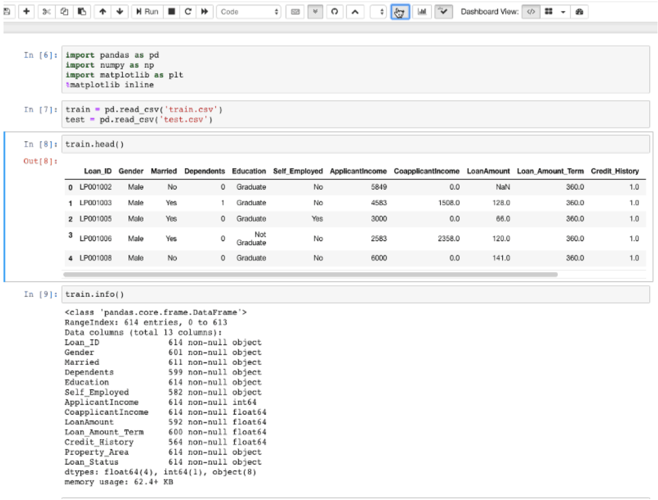
This extension adds a drop-down menu to the Notebook toolbar that allows easy insertion of code snippet cells into the current notebook.



Snippets Demo

3. Split Cells Notebook

This extension splits the cells of the notebook and places then adjacent to each other.



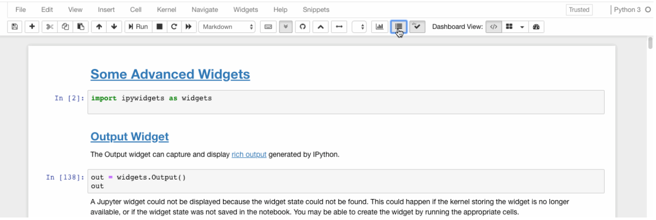
```
In [6]: import pandas as pd
        import numpy as np
        import matplotlib as plt
        %matplotlib inline

In [7]: train = pd.read_csv('train.csv')
        test = pd.read_csv('test.csv')

In [8]: train.head()

Out[8]:
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
0	LP001002	Male	No	0	Graduate	No	5849	0.0	NaN	360.0	1.0
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.0	360.0	1.0
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	360.0	1.0
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	360.0	1.0
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141.0	360.0	1.0

```
In [9]: train.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 13 columns):
Loan_ID              614 non-null object
Gender               601 non-null object
Married              611 non-null object
Dependents           599 non-null object
Education            614 non-null object
Self_Employed        582 non-null object
ApplicantIncome      614 non-null int64
CoapplicantIncome    614 non-null float64
LoanAmount           592 non-null float64
Loan_Amount_Term     600 non-null float64
Credit_History       564 non-null float64
Property_Area        614 non-null object
Loan_Status          614 non-null object
dtypes: float64(4), int64(1), object(8)
memory usage: 62.4+ KB
```

4. Table of Contents

This extension enables to collect all running headers and display them in a floating window, as a sidebar or with a navigation menu. The extension is also draggable, resizable, collapsible and dockable.



Some Advanced Widgets

```
In [2]: import ipywidgets as widgets
```

Output Widget

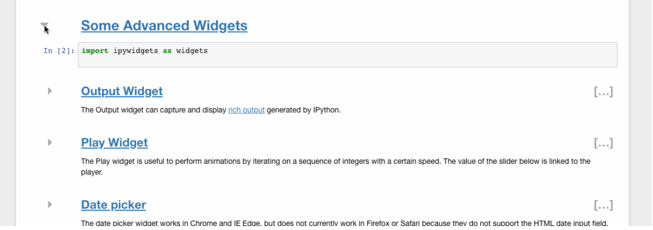
The Output widget can capture and display rich output generated by IPython.

```
In [138]: out = widgets.Output()
          out
```

A Jupyter widget could not be displayed because the widget state could not be found. This could happen if the kernel storing the widget is no longer available, or if the widget state was not saved in the notebook. You may be able to create the widget by running the appropriate cells.

5. Collapsible Headings

Collapsible Headings allows the notebook to have collapsible sections, separated by headings. So in case you have a lot of dirty code in your notebook, you can simply collapse it to avoid scrolling it again and again.



Some Advanced Widgets

```
In [2]: import ipywidgets as widgets
```

Output Widget [...]

The Output widget can capture and display rich output generated by IPython.

Play Widget [...]

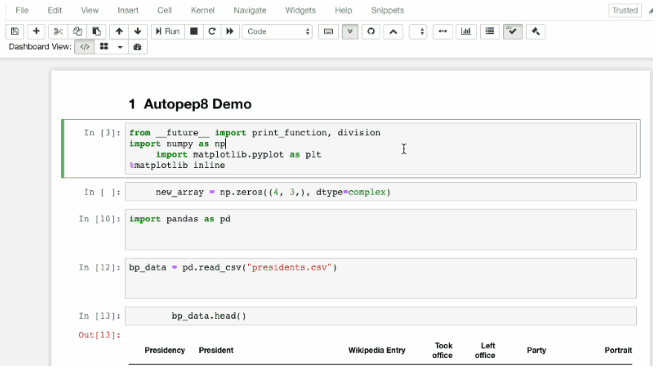
The Play widget is useful to perform animations by iterating on a sequence of integers with a certain speed. The value of the slider below is linked to the player.

Date picker [...]

The date picker widget works in Chrome and IE Edge, but does not currently work in Firefox or Safari because they do not support the HTML date input field.

6. Autopep8

Autopep8 helps to reformat/prettify the contents of code cells with just a click. If you are tired of hitting the spacebar again and again to format the code, autopep8 is your savior.



1 Autopep8 Demo

```
In [3]: from __future__ import print_function, division
        import numpy as np
            import matplotlib.pyplot as plt
        %matplotlib inline

In [ ]:     new_array = np.zeros((4, 3,), dtype=complex)

In [10]: import pandas as pd

In [12]: bp_data = pd.read_csv("presidents.csv")

In [13]:       bp_data.head()
Out[13]:
```

Presidency	President	Wikipedia Entry	Took office	Left office	Party	Portrait

. . .

4. Jupyter Widgets

Make notebooks interactive

Widgets are eventful python objects that have a representation in the browser, often as a control like a slider, textbox, etc. Widgets can be used to build **interactive GUIs** for the notebooks.

Installation

```
# pip
pip install ipywidgets
jupyter nbextension enable --py widgetsnbextension

# Conda
conda install -c conda-forge ipywidgets

#Installing ipywidgets with conda automatically enables the 
extension
```

Let us have a look at some of the widgets. For complete details, you can visit their Github repository.

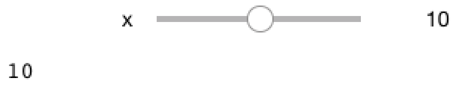
Interact

The `interact` function (`ipywidgets.interact`) automatically creates a user interface (UI) controls for exploring code and data interactively. It is the easiest way to get started using IPython's widgets.

```
# Start with some imports!

from ipywidgets import interact
import ipywidgets as widgets
```

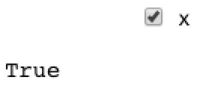
1. Basic Widgets

```
def f(x):
    return x

# Generate a slider
interact(f, x=10,);
```



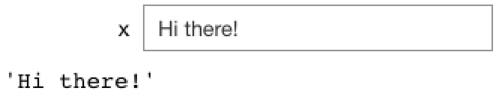
```
10
```

```
# Booleans generate check-boxes
interact(f, x=True);
```



```
True
```

```
# Strings generate text areas
interact(f, x='Hi there!');
```



```
'Hi there!'
```

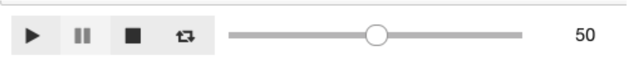
2. Advanced Widgets

Here is a list of some of the useful advanced widgets.

Play Widget

The Play widget is useful to perform animations by iterating on a sequence of integers at a certain speed. The value of the slider below is linked to the player.

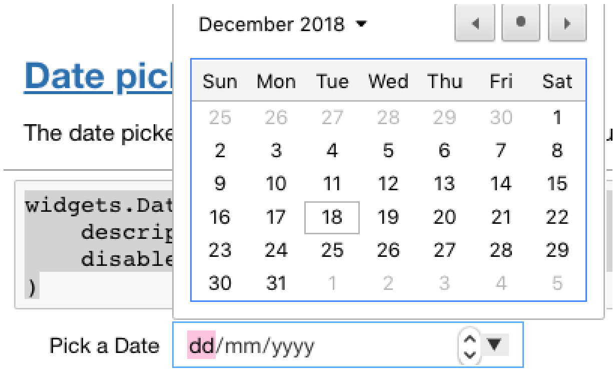
```
play = widgets.Play(
    # interval=10,
    value=50,
    min=0,
    max=100,
    step=1,
    description="Press play",
    disabled=False
)
slider = widgets.IntSlider()
widgets.jslink((play, 'value'), (slider, 'value'))
widgets.HBox([play, slider])
```



Date picker

The date picker widget works in Chrome and IE Edge but does not currently work in Firefox or Safari because they do not support the HTML date input field.

```
widgets.DatePicker(
    description='Pick a Date',
    disabled=False
)
```

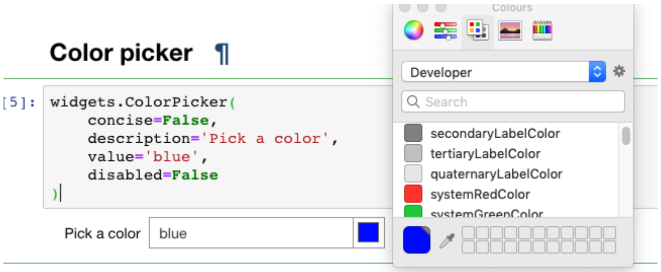


Sun	Mon	Tue	Wed	Thu	Fri	Sat
25	26	27	28	29	30	1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31	1	2	3	4	5

Pick a Date dd/mm/yyyy

Color picker

```
widgets.ColorPicker(
    concise=False,
    description='Pick a color',
    value='blue',
    disabled=False
)
```



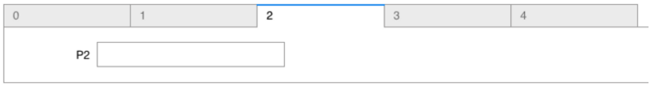
Color picker ¶

```
[5]: widgets.ColorPicker(
         concise=False,
         description='Pick a color',
         value='blue',
         disabled=False
     )
```

Pick a color: blue

Tabs

```
tab_contents = ['P0', 'P1', 'P2', 'P3', 'P4']
children = [widgets.Text(description=name) for name in
tab_contents]
tab = widgets.Tab()
tab.children = children
for i in range(len(children)):
    tab.set_title(i, str(i))
tab
```



. . .

5. Qgrid

Make Data frames intuitive

Qgrid is also a Jupyter notebook widget but mainly focussed at dataframes. It uses SlickGrid to render pandas DataFrames within a Jupyter notebook. This allows you to explore your DataFrames with intuitive scrolling, sorting and filtering controls, as well as edit your DataFrames by double-clicking cells. The Github Repository contains more details and examples.

Installation

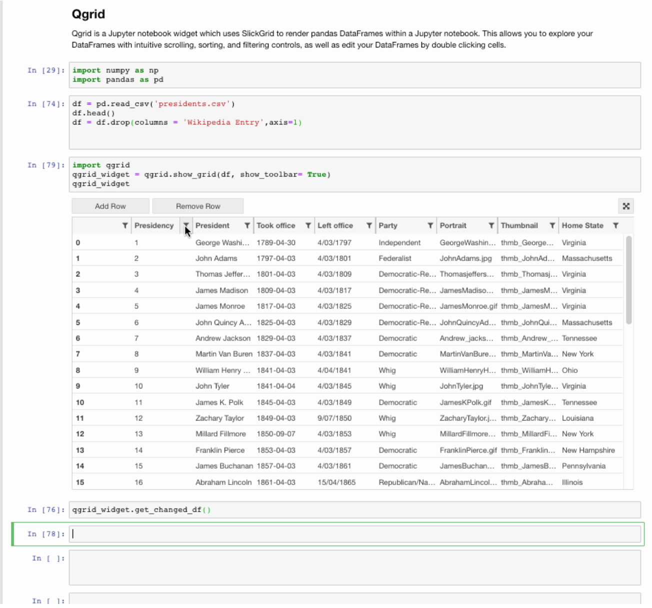
Installing with pip:

```
pip install qgrid
jupyter nbextension enable --py --sys-prefix qgrid

# only required if you have not enabled the ipywidgets
nbextension yet
jupyter nbextension enable --py --sys-prefix
widgetsnbextension
```

Installing with conda:

```
# only required if you have not added conda-forge to your
channels yet
conda config --add channels conda-forge
```

```
conda install qgrid
```



Qgrid

Qgrid is a Jupyter notebook widget which uses SlickGrid to render pandas DataFrames within a Jupyter notebook. This allows you to explore your DataFrames with intuitive scrolling, sorting, and filtering controls, as well as edit your DataFrames by double clicking cells.

```
In [29]: import numpy as np
         import pandas as pd
```

```
In [74]: df = pd.read_csv('presidents.csv')
         df.head()
         df = df.drop(columns = 'Wikipedia Entry',axis=1)
```

```
In [79]: import qgrid
         qgrid_widget = qgrid.show_grid(df, show_toolbar= True)
         qgrid_widget
```

	Presidency	President	Took office	Left office	Party	Portrait	Thumbnail	Home State
0	1	George Washi...	1789-04-30	4/03/1797	Independent	GeorgeWashin...	thmb_George...	Virginia
1	2	John Adams	1797-04-03	4/03/1801	Federalist	JohnAdams.jpg	thmb_JohnAd...	Massachusetts
2	3	Thomas Jeffer...	1801-04-03	4/03/1809	Democratic-Re...	Thomasjeffers...	thmb_Thomasj...	Virginia
3	4	James Madison	1809-04-03	4/03/1817	Democratic-Re...	JamesMadiso...	thmb_JamesM...	Virginia
4	5	James Monroe	1817-04-03	4/03/1825	Democratic-Re...	JamesMonroe.gif	thmb_JamesM...	Virginia
5	6	John Quincy A...	1825-04-03	4/03/1829	Democratic-Re...	JohnQuincyAd...	thmb_JohnQui...	Massachusetts
6	7	Andrew Jackson	1829-04-03	4/03/1837	Democratic	Andrew_jacks...	thmb_Andrew_...	Tennessee
7	8	Martin Van Buren	1837-04-03	4/03/1841	Democratic	MartinVanBure...	thmb_MartinVa...	New York
8	9	William Henry ...	1841-04-03	4/04/1841	Whig	WilliamHenryH...	thmb_WilliamH...	Ohio
9	10	John Tyler	1841-04-04	4/03/1845	Whig	JohnTyler.jpg	thmb_JohnTyle...	Virginia
10	11	James K. Polk	1845-04-03	4/03/1849	Democratic	JamesKPolk.gif	thmb_JamesK...	Tennessee
11	12	Zachary Taylor	1849-04-03	9/07/1850	Whig	ZacharyTaylor.j...	thmb_Zachary...	Louisiana
12	13	Millard Fillmore	1850-09-07	4/03/1853	Whig	MillardFillmore...	thmb_MillardFi...	New York
13	14	Franklin Pierce	1853-04-03	4/03/1857	Democratic	FranklinPierce.gif	thmb_Franklin...	New Hampshire
14	15	James Buchanan	1857-04-03	4/03/1861	Democratic	JamesBuchan...	thmb_JamesB...	Pennsylvania
15	16	Abraham Lincoln	1861-04-03	15/04/1865	Republican/Na...	AbrahamLincol...	thmb_Abraha...	Illinois

```
In [76]: qgrid_widget.get_changed_df()

In [78]: |

In [ ]:

In [ ]:
```

. . .

6. Slideshow

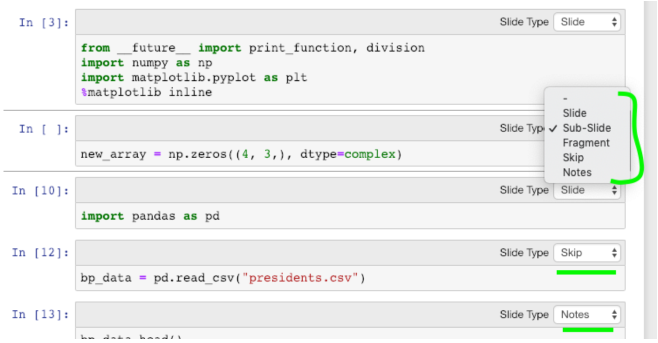
Code is great when communicated.

Notebooks are an effective tool for teaching and writing explainable codes. However, when we want to present our work either we display our entire notebook(with all the codes) or we take the help of powerpoint. Not any more. Jupyter Notebooks can be easily converted to slides and we can easily choose what to show and what to hide from the notebooks.

There are two ways to convert the notebooks into slides:

1. Jupyter Notebook's built-in Slide option

Open a new notebook and navigate to **View → Cell Toolbar → Slideshow.** A light grey bar appears on top of each cell, and you can customize the slides.

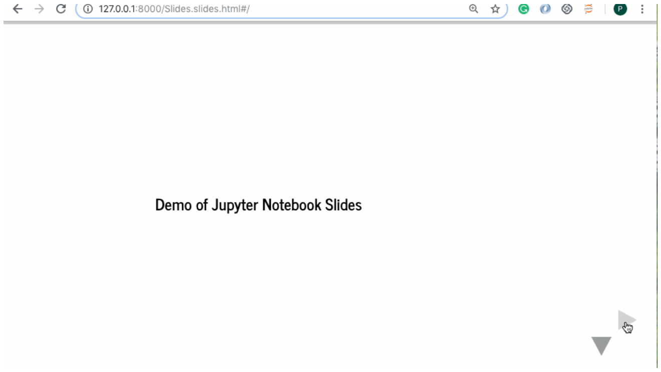


```
In [3]:
from __future__ import print_function, division
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

In [ ]:
new_array = np.zeros((4, 3,), dtype=complex)

In [10]:
import pandas as pd

In [12]:
bp_data = pd.read_csv("presidents.csv")

In [13]:
bp_data.head()
```

Now go to the directory where the notebook is present and enter the following code:

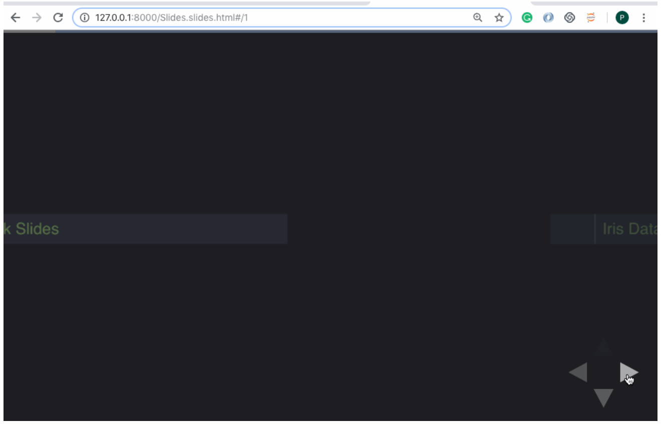
```
jupyter nbconvert *.ipynb --to slides --post serve
# insert your notebook name instead of *.ipynb
```

The slides get displayed at port 8000. Also, a `.html` file will be generated in the directory, and you can also access the slides from there.



127.0.0.1:8000/Slides.slides.html#/
Demo of Jupyter Notebook Slides

This would look even more classy with a themed background. Let us apply the theme '**onedork**' to the notebook and then convert it into a slideshow.



These slides have a drawback i.e. you can see the code but cannot edit it. RISE plugin offers a solution.

2. Using the RISE plugin

RISE is an acronym for **Reveal.js—Jupyter/IPython Slideshow Extension**. It utilized the reveal.js to run the slideshow. This is super useful since it also gives the ability to run the code without having to exit the slideshow.

Installation

1—Using conda (recommended):

```
conda install -c damianavila82 rise
```

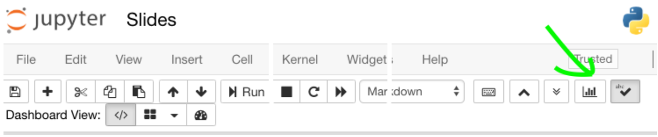
2—Using pip (less recommended):

```
pip install RISE
```

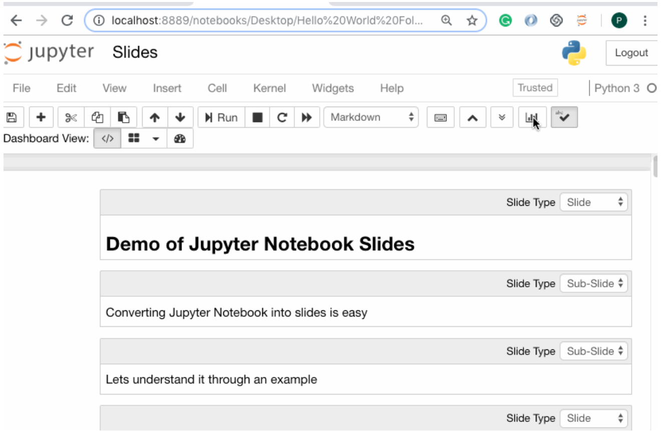
and then two more steps to install the JS and CSS in the proper places:

```
jupyter-nbextension install rise --py --sys-prefix

#enable the nbextension:
jupyter-nbextension enable rise --py --sys-prefix
```

Let us now use RISE for the interactive slideshow. We shall re-open the Jupyter Notebook we created earlier. Now we notice a new extension that says "Enter/Exit RISE Slideshow."



Click on it, and you are good to go. Welcome to the world of interactive slides.


localhost:8889/notebooks/Desktop/Hello%20World%20Fol...
jupyter Slides
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Markdown
Dashboard View:

Slide Type: Slide

Demo of Jupyter Notebook Slides

Slide Type: Sub-Slide

Converting Jupyter Notebook into slides is easy

Slide Type: Sub-Slide

Lets understand it through an example

Slide Type: Slide

Refer to the documentation for more information.

. . .

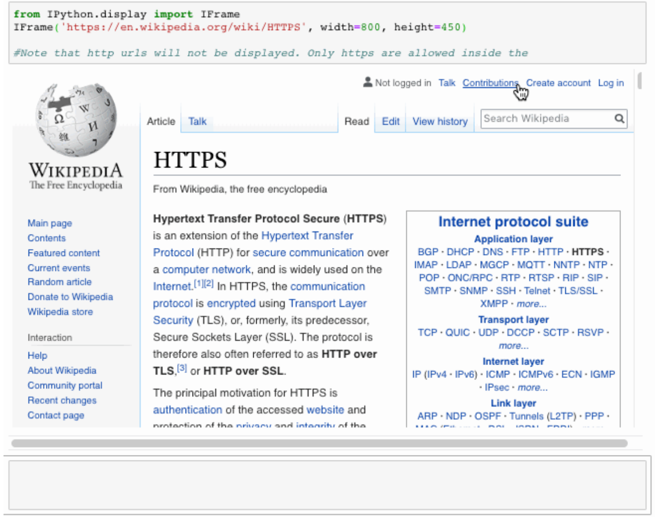
6. Embedding URLs, PDFs, and Youtube Videos

Display it right there!

Why go with mere links when you can easily embed an URL, pdf, and videos into your Jupyter Notebooks using IPython's display module.

URLs

```
#Note that http urls will not be displayed. Only https are
allowed inside the Iframe

from IPython.display import IFrame
IFrame('https://en.wikipedia.org/wiki/HTTPS', width=800,
height=450)
```



```
from IPython.display import IFrame
IFrame('https://en.wikipedia.org/wiki/HTTPS', width=800, height=450)

#Note that http urls will not be displayed. Only https are allowed inside the
```

HTTPS

From Wikipedia, the free encyclopedia

Hypertext Transfer Protocol Secure (**HTTPS**) is an extension of the Hypertext Transfer Protocol (HTTP) for secure communication over a computer network, and is widely used on the Internet.[1][2] In HTTPS, the communication protocol is encrypted using Transport Layer Security (TLS), or, formerly, its predecessor, Secure Sockets Layer (SSL). The protocol is therefore also often referred to as **HTTP over TLS**,[3] or **HTTP over SSL**.

The principal motivation for HTTPS is authentication of the accessed website and

Internet protocol suite

Application layer

BGP · DHCP · DNS · FTP · HTTP · **HTTPS** · IMAP · LDAP · MGCP · MQTT · NNTP · NTP · POP · ONC/RPC · RTP · RTSP · RIP · SIP · SMTP · SNMP · SSH · Telnet · TLS/SSL · XMPP · *more...*

Transport layer

TCP · QUIC · UDP · DCCP · SCTP · RSVP · *more...*

Internet layer

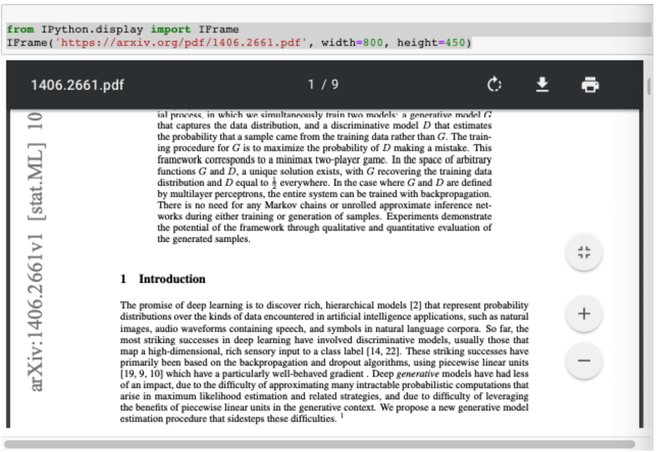
IP (IPv4 · IPv6) · ICMP · ICMPv6 · ECN · IGMP · IPsec · *more...*

Link layer

ARP · NDP · OSPF · Tunnels (L2TP) · PPP ·

PDFs

```
from IPython.display import IFrame
IFrame('https://arxiv.org/pdf/1406.2661.pdf', width=800,
height=450)
```



```
from IPython.display import IFrame
IFrame('https://arxiv.org/pdf/1406.2661.pdf', width=800, height=450)
```

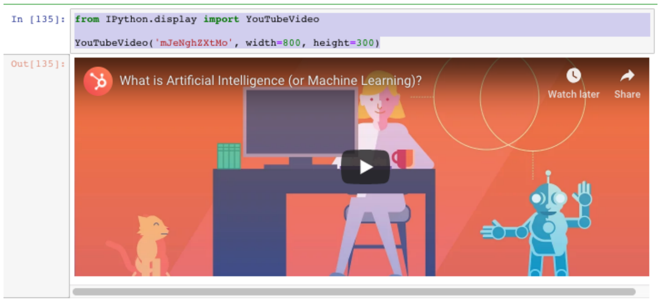
ial process, in which we simultaneously train two models: a generative model G that captures the data distribution, and a discriminative model D that estimates the probability that a sample came from the training data rather than G. The training procedure for G is to maximize the probability of D making a mistake. This framework corresponds to a minimax two-player game. In the space of arbitrary functions G and D, a unique solution exists, with G recovering the training data distribution and D equal to $\frac{1}{2}$ everywhere. In the case where G and D are defined by multilayer perceptrons, the entire system can be trained with backpropagation. There is no need for any Markov chains or unrolled approximate inference networks during either training or generation of samples. Experiments demonstrate the potential of the framework through qualitative and quantitative evaluation of the generated samples.

1 Introduction

The promise of deep learning is to discover rich, hierarchical models [2] that represent probability distributions over the kinds of data encountered in artificial intelligence applications, such as natural images, audio waveforms containing speech, and symbols in natural language corpora. So far, the most striking successes in deep learning have involved discriminative models, usually those that map a high-dimensional, rich sensory input to a class label [14, 22]. These striking successes have primarily been based on the backpropagation and dropout algorithms, using piecewise linear units [19, 9, 10] which have a particularly well-behaved gradient . Deep *generative* models have had less of an impact, due to the difficulty of approximating many intractable probabilistic computations that arise in maximum likelihood estimation and related strategies, and due to difficulty of leveraging the benefits of piecewise linear units in the generative context. We propose a new generative model estimation procedure that sidesteps these difficulties. [1]

Youtube Videos

```
from IPython.display import YouTubeVideo

YouTubeVideo('mJeNghZXtMo', width=800, height=300)
```



```
In [135]: from IPython.display import YouTubeVideo

          YouTubeVideo('mJeNghZXtMo', width=800, height=300)
Out[135]:
```

. . .

Conclusion

These were some of the features of the Jupyter Notebooks that I found useful and worth sharing. Some of them would be obvious to you while some may be new. So, go ahead and experiment with them. Hopefully, they will be able to save you some time and give you a better UI experience. Also feel free to suggest other useful features in the comments.