



Open in app

Get started



Published in Towards Data Science



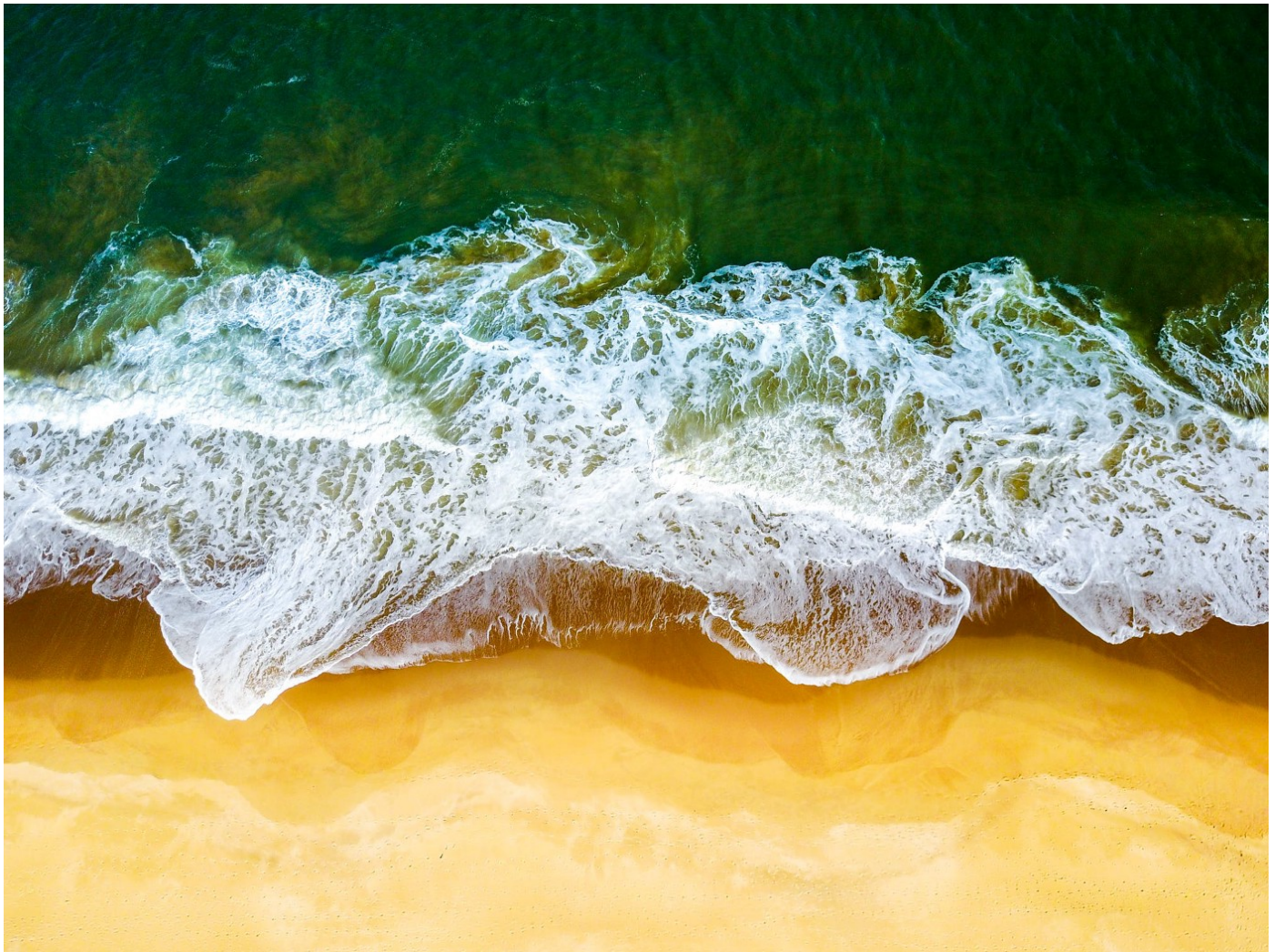
Will Koehrsen

Follow

Dec 8, 2018 · 7 min read · Listen



Save

(Source)

How to Write a Jupyter Notebook Extension

Make the Jupyter Notebook your playground





Open in app

Get started

numerous existing extensions, we can also write our own extension to do extend the functionality of Jupyter.

In this article, we'll see how to write a Jupyter Notebook extension that adds a default cell to the top of each new notebook which is useful when there are libraries you find yourself importing into every notebook. If you want the background on extensions, then check out [this article](#). The complete code for this extension is [available on GitHub](#).

The final outcome of the Default cell extension is shown below:



Extension to add a default cell to the top of every notebook.

Notes before Starting

(If you don't yet have Jupyter Extensions, [check out this article](#) or just run the following code in a command prompt: `pip install jupyter_contrib_nbextensions && jupyter contrib nbextensions install` and then start a new notebook server and navigate to the extensions tab).

Unfortunately, there's not much official documentation on writing your own extension. My tactic was to read the [other extensions](#), copy and paste copiously, and experiment until I figured it out. [This Stack Overflow](#) question and answer provided the basic code for this extension.



[Open in app](#)[Get started](#)

Structure of a Jupyter Notebook Extension

There are 3 parts (at minimum) to any Jupyter Notebook extension:

1. `description.yaml` : A configuration file read by Jupyter
2. `main.js` : The Javascript code for the extension itself
3. `README.md` : A markdown description of extension

(We can also have more functions in other files or `css` for styling).

These 3 files should live in a single directory which we'll call `default_cell` . This folder, in turn, needs to be in the `nbextensions` subdirectory of the `jupyter_contrib_extensions` library (what you installed with `pip`. To find the location of a library installed with `pip` run `pip view package`.)

My `jupyter_contrib_extensions` directory is:

```
/usr/local/lib/python3.6/site-  
packages/jupyter_contrib_nbextensions/nbextensions
```

So the file structure looks like (with `nbextensions` as shown above):

```
nbextensions/  
  default_cell/  
    - description.yaml  
    - main.js  
    - README.md
```

When you run `jupyter notebook` , Jupyter looks in this location for extensions, and shows them on the extensions tab on the server:





Open in app

Get started

Quit

Logout

jupyter

Files

Running

Clusters

Nbextensions



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

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

Jupyter Notebook Extensions tab

When you make a change to the extension files while developing and you want to see the effects in a Jupyter Notebook, you need to run the command `jupyter contrib nbextensions install` to rewrite the Jupyter config files. Then, restart the notebook server to see your changes.

With the details out of the way, let's go through the three files we need.

description.yaml

YAML (YAML Ain't Markup Language) is a human-readable standard for writing configuration and header files. The YAML file describes the extension to the Jupyter Extensions Configurator to be rendered on the extensions tab.

This is a fairly easy file to copy + paste and modify to our needs.



[Open in app](#)[Get started](#)

This is then rendered nicely on the `NBExtensions` tab in the notebook:

Default cell

Adds and runs a default cell at the top of each new notebook. Also adds a button to insert and run the default cell above the current cell. Helpful for commonly used imports.

section: notebook

require path: `default_cell/main`

compatibility: 3.x, 4.x, 5.x, 6.x

The compatibility indicates the versions of Jupyter for which the extension works. I just added all of them (3.x — 6.x) and it seemed to work fine!

main.js

This is the heart of the application where the actual logic for the extension lives. Jupyter



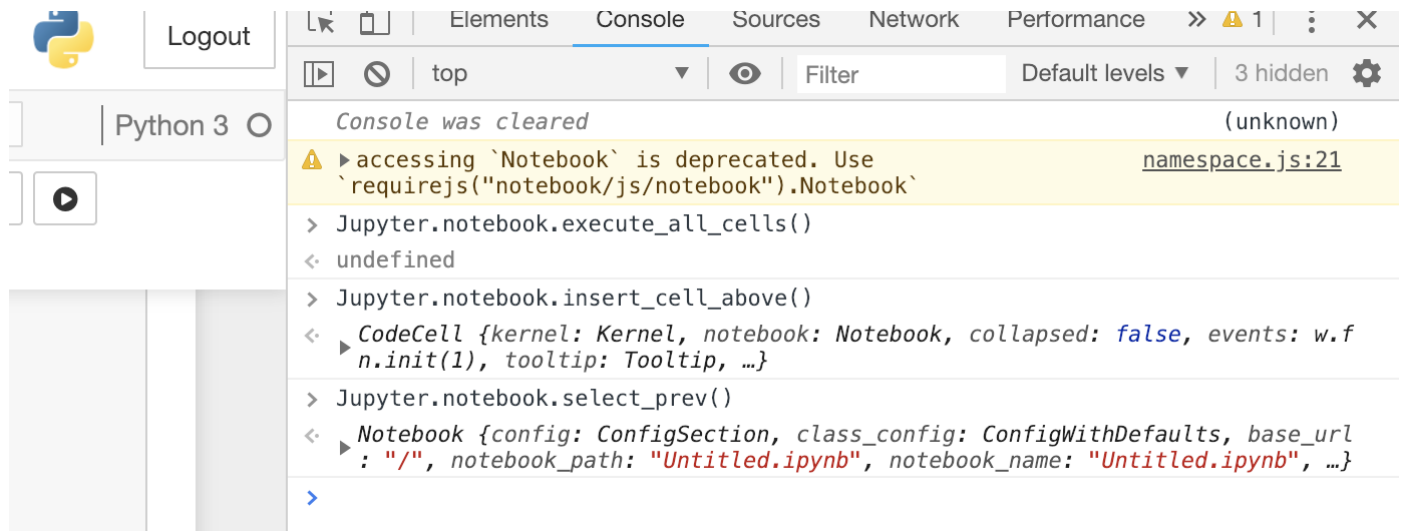


Open in app

Get started

It can be a little difficult to figure out what commands to use to make the notebook do what you want. One way to experiment is using the Chrome developer tools (cntrl + shift + i on Windows) or right-click > inspect.

With the developer tools open, we can use the `console` tab to run commands.



Console in Chrome developer tools

Try opening the developer tools in a Jupyter Notebook and play with options that start with `Jupyter.notebook`. Any commands you enter will have to be in Javascript. An example of this behavior can be seen in the clip below. I open up the developer tools and then run a few commands to execute cells, insert a new cell, and select the previous cell.





Open in app

Get started

```

In [4]: # Standard data science libraries
import pandas as pd
import numpy as np
from scipy import stats
import featuretools as ft

# Visualization
import matplotlib.pyplot as plt
import seaborn as sns
plt.style.use('bmh')

# Options for pandas
pd.options.display.max_columns = 20

# Display all cell outputs
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = 'all'

executed in 4ms, finished 14:38:56 2018-12-08

In [5]: # Standard data science libraries
import pandas as pd
import numpy as np
from scipy import stats
import featuretools as ft

# Visualization
import matplotlib.pyplot as plt
import seaborn as sns
plt.style.use('bmh')

# Options for pandas
pd.options.display.max_columns = 20

# Display all cell outputs
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = 'all'

executed in 4ms, finished 14:38:56 2018-12-08

```

Developing the Javascript code required a lot of experimenting like this! It also helps to read the other extensions to figure out what you need to do.

The final `main.js` is below:





Open in app

Get started

The most important part of the code is the `add_cell` function.

```
var add_cell = function() {  
  Jupyter.notebook.  
    insert_cell_above('code').  
    // Define default cell here  
    set_text(`Define default cell here`);  
    Jupyter.notebook.select_prev();  
    Jupyter.notebook.execute_cell_and_select_below();  
};
```

This adds a code cell above the currently selected cell with the python code written in the parenthesis (in the `set_text` call). This is where the default code cell should be defined. The function then executes the cell and selects the one below.





Open in app

Get started



The `load_ipython_extension` function first checks the number of cells in the notebook. If there is only one cell — a new notebook — it calls the `add_cell` function which places the default cell at the top of the notebook.

```
// Run on start
function load_ipython_extension() {

// Add a default cell if a new notebook
  if (Jupyter.notebook.get_cells().length===1){
    add_cell();
  }
  defaultCellButton();
}
```

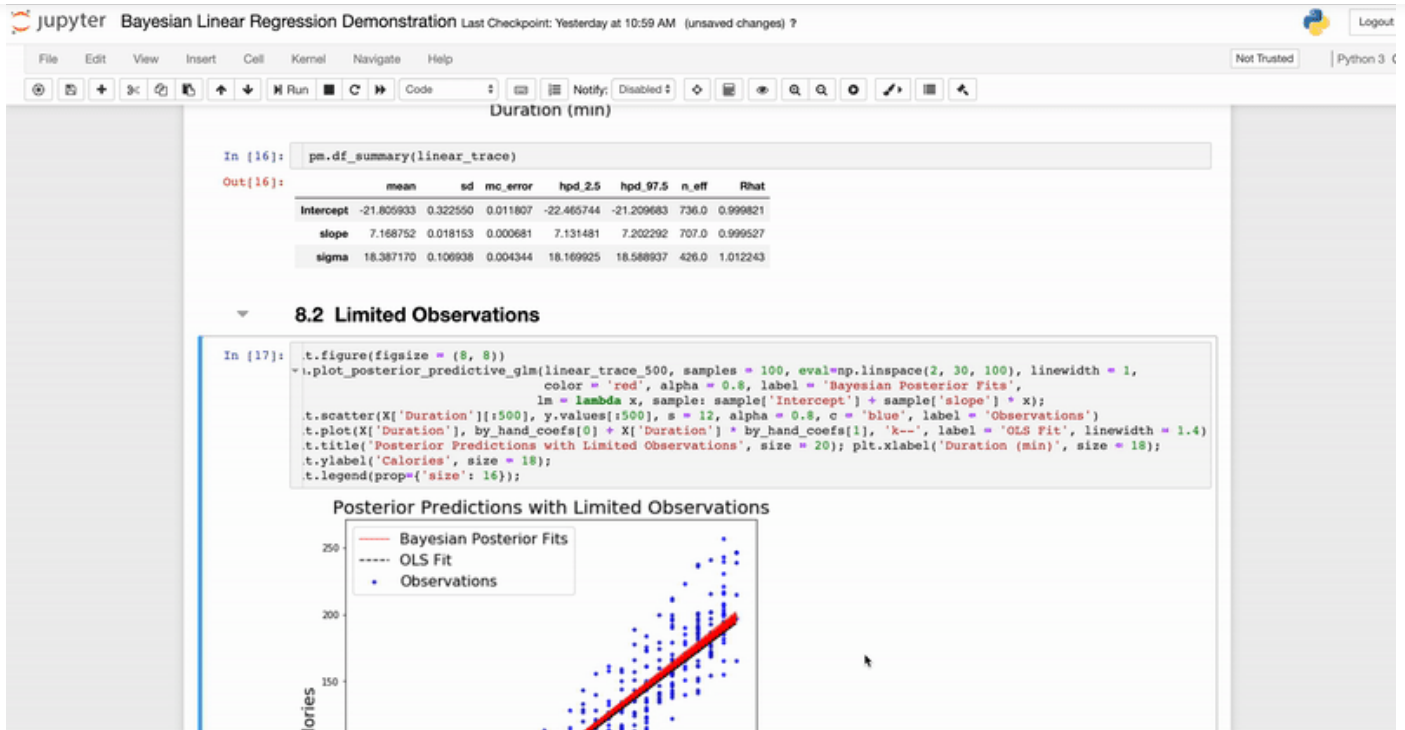
It then runs the `defaultCellButton` function which places a button on the Jupyter Notebook toolbar. We can use this button to add and run the default cell above the currently selected cell. This could be useful when we have an already started notebook and we want our normal imports.





Open in app

Get started



Function of defaultCellButton

There is an almost unlimited number of tasks we could accomplish with Javascript in the browser. This is only a simple application, but there are many more complex Jupyter extensions (such as the [Variable Inspector](#)) that demonstrate more of the capabilities. We can also write applications in Javascript that call on Python scripts for even greater control.

README.md

A readme [markdown file](#) should be familiar to anyone with even a little experience programming. Here is where we explain what our application does and how to use it. This is displayed on the extensions tab:

▼ /nbextensions/default_cell/README.md

Default Cell

Adds and runs a default cell at the top of each new notebook. To change the default cell, edit the `add_cell` function in the `main.js` file. The relevant section is:

```
Jupyter.notebook.insert_cell_above('code').set_text(``)
```

Set the `text` to the default notebook cell you would like.

This extension also adds a button to the toolbar allowing you to insert and run the default cell above your current cell. This can be helpful if you open an already started notebook and notice





Open in app

Get started

(The actual code is pretty boring but for completeness, here it is)

Default Cell

=====

Adds and runs a default cell at the top of each new notebook. To change the default cell, edit the ``add_cell`` function in the ``main.js`` file. The relevant section is

```
`Jupyter.notebook.insert_cell_above('code', 0).set_text(``)
```

Set the ``text`` to whatever you would like.

This extension also adds a button to the toolbar allowing you to insert and run the default cell above your current cell. This can be helpful if you open an already started notebook and notice you are missing some common imports.

This extension is a work in progress and any help would be appreciated. Feel free to make contributions on GitHub or contact the author (Will Koehrsen) at wjk68@case.edu

Once you have the three required files, your extension is complete.

To see the extension working, make sure the `default_cell` directory is in the correct location, run `jupyter contrib nbextensions install` and start up a Jupyter Notebook server. If you navigate to the `NBExtensions` tab, you will be able to enable the extension (if you don't have the tab, open a notebook and got to `Edit > nbextensions config`).

☐ Code prettify☒ Collapsible Headings☒ Default cell☐ Exercise☒ Codefolding☐ Comment/Uncomment Hotkey☐ Equation Auto Numbering☐ Exercise2

Extension enabled





Open in app

Get started



Default cell jupyter notebook extension

This extension isn't life-changing, but it might save you a few seconds!

Conclusions

Part of the joy of gaining computer literacy is realizing that if you want to accomplish something on a computer, chances are that you probably can with the right tools and willingness to learn. This small example of making a Jupyter Notebook extension demonstrates that we are not limited by what we get out of the box. We just need a few lines of code to accomplish our goals.

Hopefully, either this extension proves useful to you or inspires you to write your own. I have gotten a lot of use from extensions and am looking forward to seeing what else people can develop. Once you do develop an extension, share it so others can marvel at your code and benefit from your hard work.

As always, I welcome feedback and constructive criticism. I can be reached on Twitter



[Open in app](#)[Get started](#)

Sign up for The Variable

By Towards Data Science

Every Thursday, the Variable delivers the very best of Towards Data Science: from hands-on tutorials and cutting-edge research to original features you don't want to miss. [Take a look.](#)

[Get this newsletter](#)

[About](#) [Help](#) [Terms](#) [Privacy](#)

Get the Medium app

