Dashboards Documentation

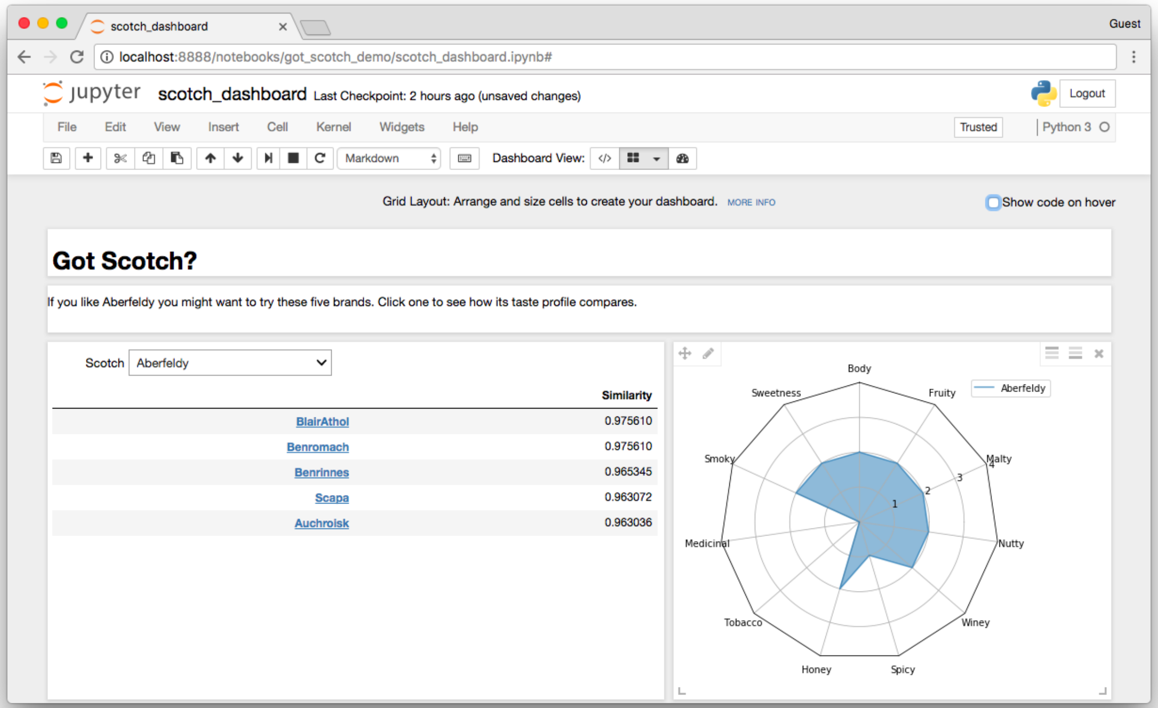
Release 0.6

Project Jupyter

Oct 03, 2017

User Documentation

The dashboards layout extension is an add-on for Jupyter Notebook. It lets you arrange your notebook outputs (text, plots, widgets, . . .) in grid- or report-like layouts. It saves information about your layouts in your notebook document. Other people with the extension can open your notebook and view your layouts.


scotch_dashboard
Guest
localhost:8888/notebooks/got_scotch_demo/scotch_dashboard.ipynb#
jupyter
scotch_dashboard Last Checkpoint: 2 hours ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Markdown
Dashboard View:
Grid Layout: Arrange and size cells to create your dashboard. MORE INFO
Show code on hover
Body
Fruity
Malty
Nutty
Winey
Spicy
Honey
Tobacco
Medicinal
Smoky
Sweetness
Aberfeldy
1
2
3
4

Got Scotch?

If you like Aberfeldy you might want to try these five brands. Click one to see how its taste profile compares.

Scotch: Aberfeldy

	Similarity
BlairAthol	0.975610
Benromach	0.975610
Benrinnes	0.965345
Scapa	0.963072
Auchroisk	0.963036

For a sample of what's possible with the dashboard layout extension, have a look at the demo dashboard-notebooks in the project repository.

CHAPTER 1

Getting started

This document describes some of the basics of installing and enabling the dashboards layout extension.

1.1 Prerequisites

- Jupyter Notebook >=4.2 running on Python 3.x or Python 2.7.x
- Edge, Chrome, Firefox, or Safari

1.2 Installing and Enabling

The following steps install the extension package using `pip` and enable the extension in the active Python environment.

```
pip install jupyter_dashboards
jupyter dashboards quick-setup --sys-prefix
```

Run `jupyter dashboards quick-setup --help` for other options. Note that the second command is a shortcut for the following:

```
jupyter nbextension install --py jupyter_dashboards --sys-prefix
jupyter nbextension enable --py jupyter_dashboards --sys-prefix
```

Alternatively, the following command both installs and enables the package using `conda`.

```
conda install jupyter_dashboards -c conda-forge
```

1.3 Disabling and Uninstalling

The following steps deactivate the extension in the active Python environment and uninstall the package using `pip`.

```
jupyter dashboards quick-remove --sys-prefix
pip uninstall jupyter_dashboards
```

Note that the first command is a shortcut for the following:

```
jupyter nbextension disable --py jupyter_dashboards --sys-prefix
jupyter nbextension uninstall --py jupyter_dashboards --sys-prefix
```

The following command deactivates and uninstalls the package if it was installed using `conda`.

```
conda remove jupyter_dashboards
```

1.4 Legacy Notes

If you installed the dashboard extension against Jupyter notebook 4.0 or 4.1, you may need to manually remove this line from your `jupyter_notebook_config.py` file when uninstalling or upgrading:

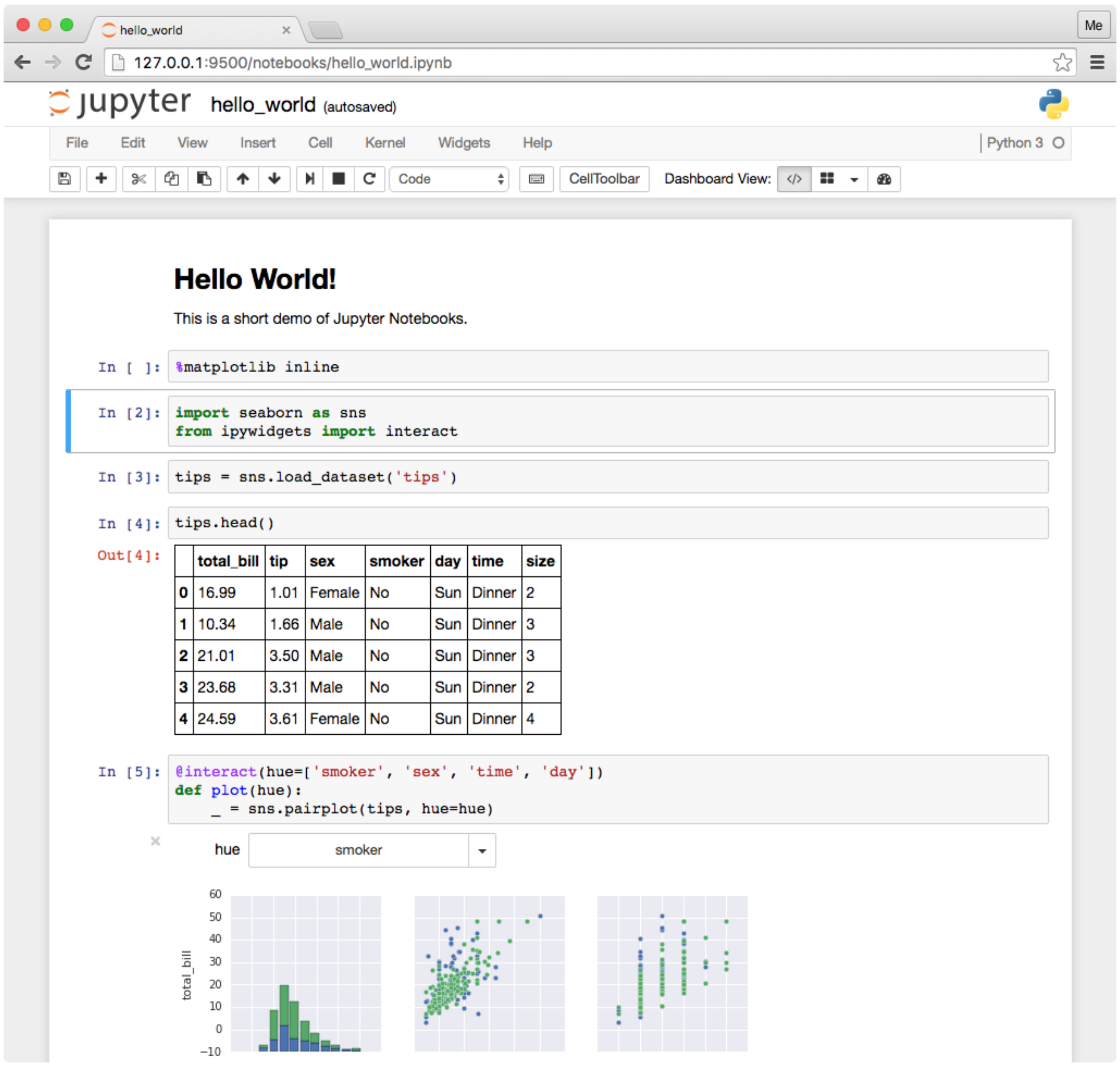
```
# [YOUR_JUPYTER_CONFIG_PATH]/jupyter_notebook_config.py
c.NotebookApp.server_extensions.append('urth.dashboard.nbexts')
```

CHAPTER 2

Creating dashboard layouts

This page provides a brief walkthrough of using the dashboard extension. The extension provides additional, built-in help about all of the dashboard features. The steps below include instructions on how to access the help.

Create a new Jupyter notebook document in a language of your choice. Insert markdown and code into the notebook. Run the cells to generate text, plots, widgets, etc.


hello_world
127.0.0.1:9500/notebooks/hello_world.ipynb
Me
jupyter hello_world (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Python 3
Code
CellToolbar
Dashboard View:
hue
smoker
total_bill
60
50
40
30
20
10
0
−10

Hello World!

This is a short demo of Jupyter Notebooks.

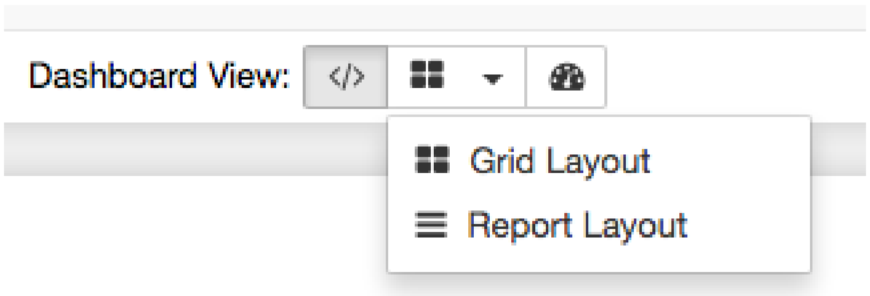
```
In [ ]: %matplotlib inline

In [2]: import seaborn as sns
        from ipywidgets import interact

In [3]: tips = sns.load_dataset('tips')

In [4]: tips.head()
```

Out[4]:

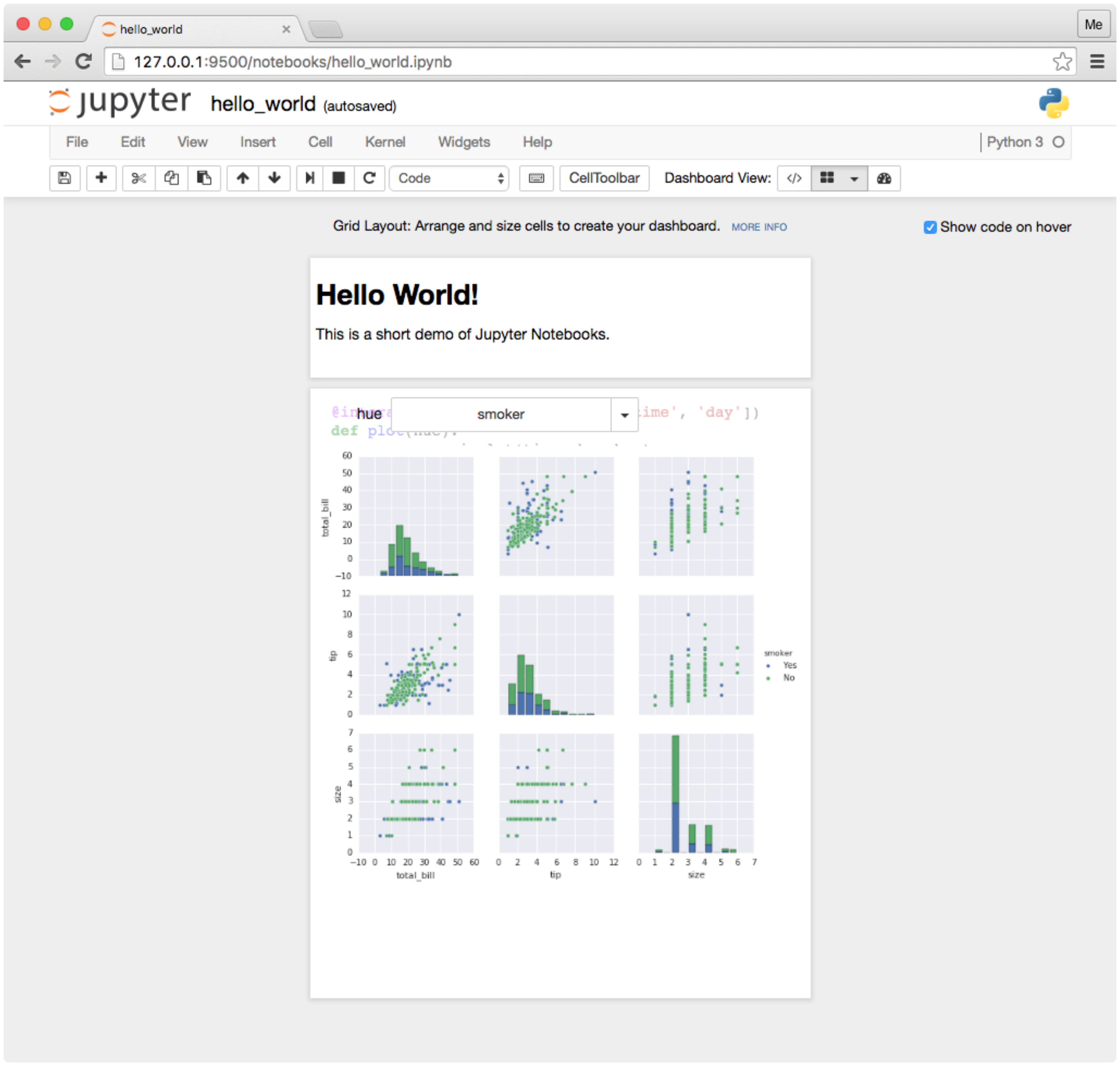
	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

```
In [5]: @interact(hue=['smoker', 'sex', 'time', 'day'])
        def plot(hue):
            _ = sns.pairplot(tips, hue=hue)
```

Select either *Grid Layout* or *Report Layout* in the *Dashboard View* toolbar. Alternatively, use the options in the *View -> Dashboard Layout* menu.



In grid layout, drag handles to resize and move cells in the grid. Click the buttons to add or remove cells in the layout. Use the *Cell -> Dashboard* menu items for batch operations.


hello_world
127.0.0.1:9500/notebooks/hello_world.ipynb
jupyter hello_world (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Python 3
Code
CellToolbar
Dashboard View:
Me

Grid Layout: Arrange and size cells to create your dashboard. MORE INFO

Show code on hover

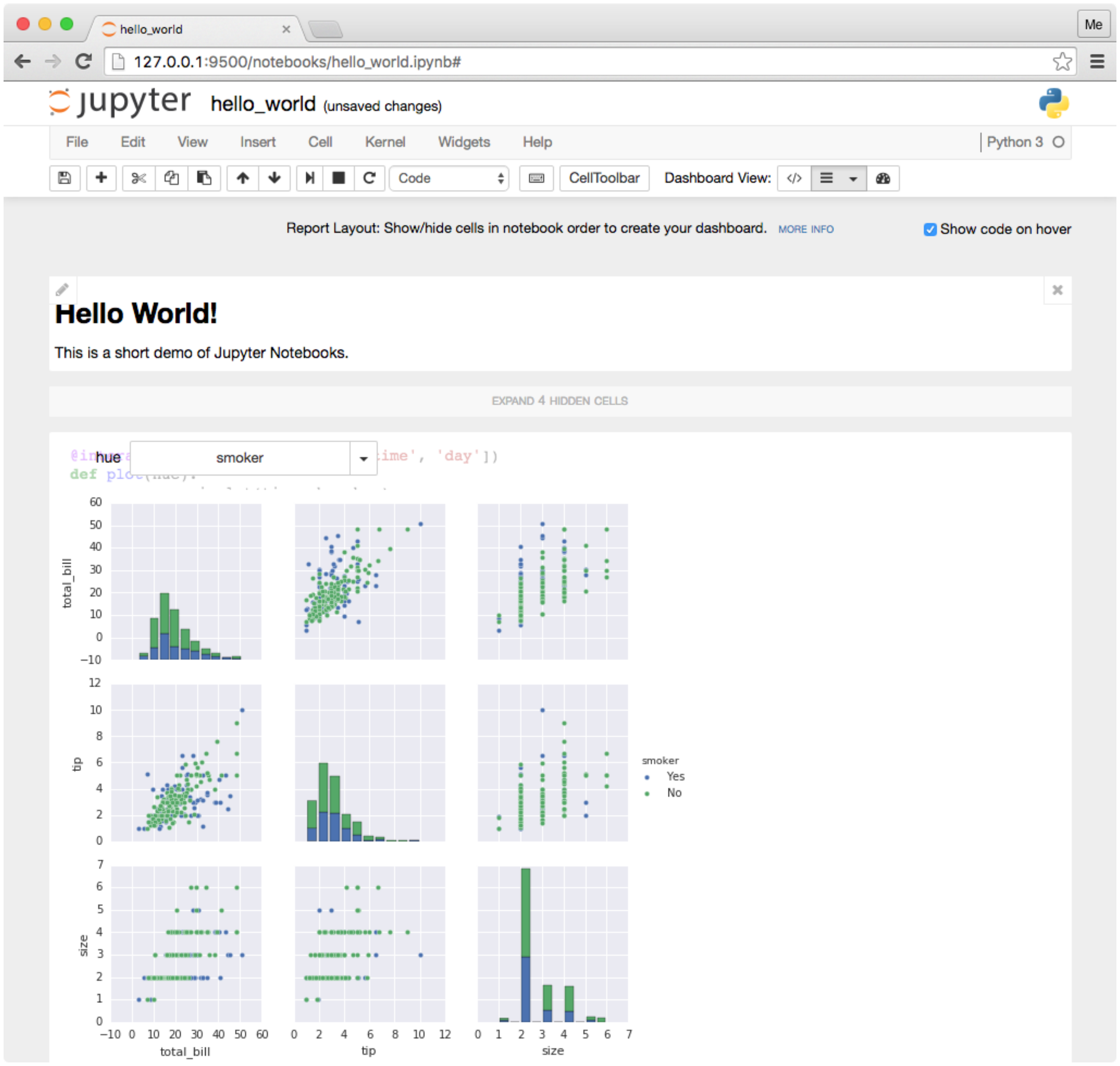
Hello World!

This is a short demo of Jupyter Notebooks.

hue smoker

total_bill
tip
size
smoker
Yes
No

In report layout, click buttons to show or hide cells.

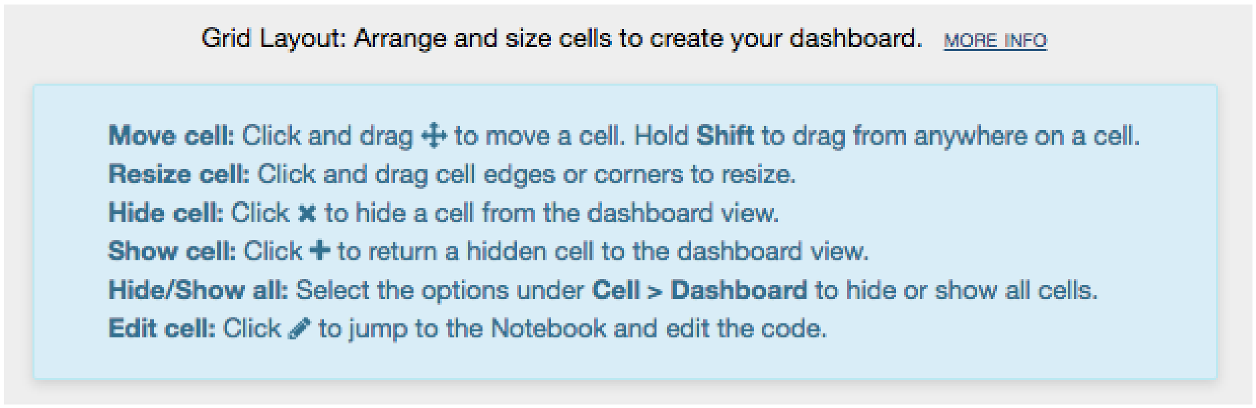


Report Layout: Show/hide cells in notebook order to create your dashboard.

Hello World!

This is a short demo of Jupyter Notebooks.

Click *More Info* at the top of the layout view for help with additional features.

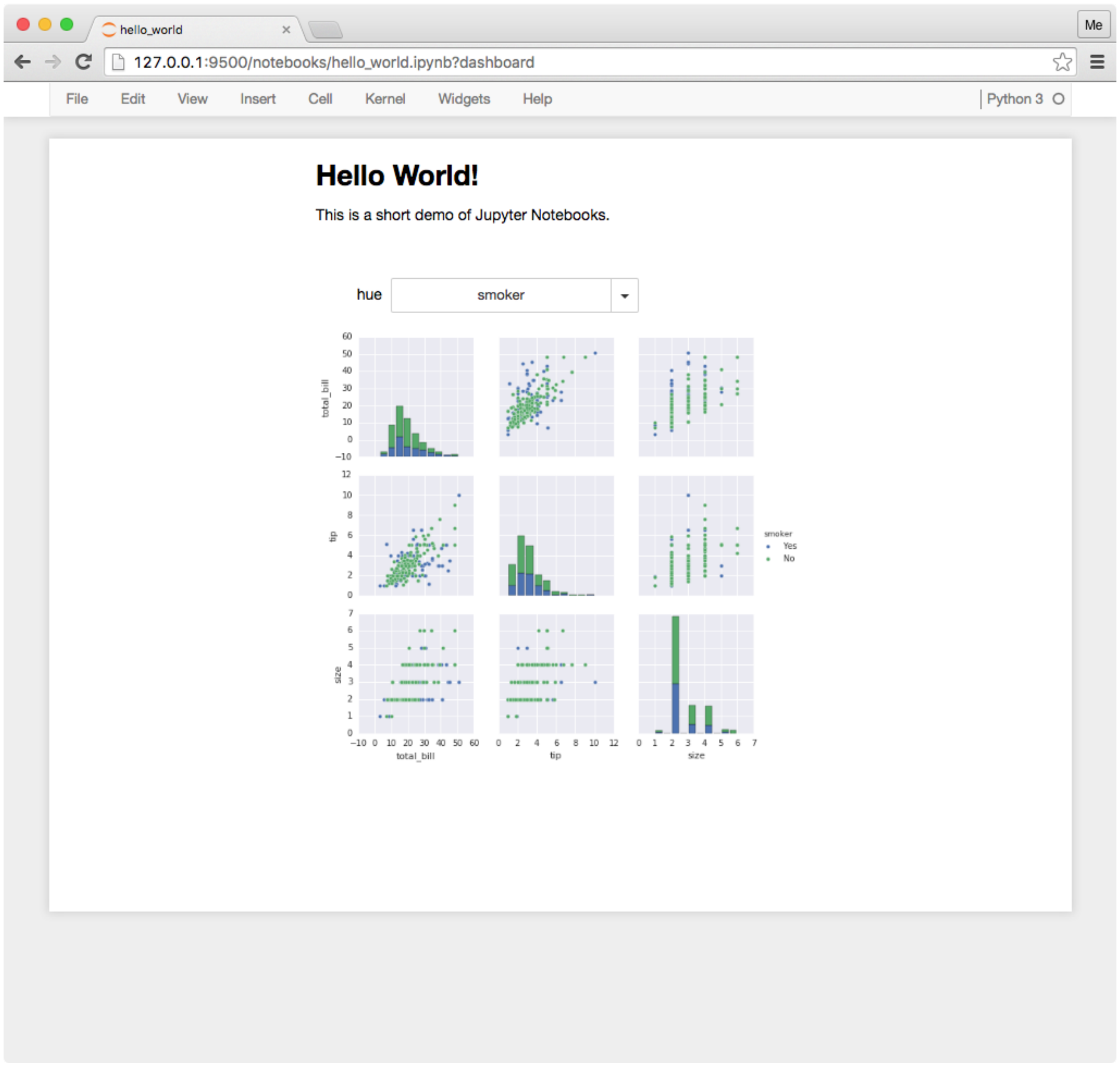


Grid Layout: Arrange and size cells to create your dashboard. MORE INFO

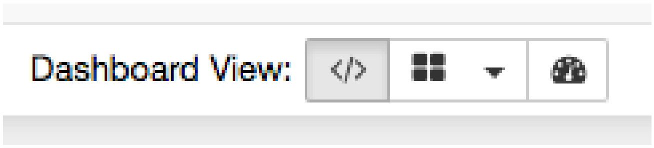
Move cell: Click and drag ✥ to move a cell. Hold **Shift** to drag from anywhere on a cell.
Resize cell: Click and drag cell edges or corners to resize.
Hide cell: Click ✖ to hide a cell from the dashboard view.
Show cell: Click ✚ to return a hidden cell to the dashboard view.
Hide/Show all: Select the options under **Cell > Dashboard** to hide or show all cells.
Edit cell: Click ✎ to jump to the Notebook and edit the code.

Click the *Dashboard Preview* button in the toolbar to view and interact with the cells without the authoring tools.

Alternatively, click the *Dashboard Preview* menu item in the *View* menu.


hello_world
127.0.0.1:9500/notebooks/hello_world.ipynb?dashboard
File Edit View Insert Cell Kernel Widgets Help
Python 3
Hello World!
This is a short demo of Jupyter Notebooks.
hue
smoker
total_bill
tip
size
smoker
Yes
No

Click the *Notebook View* button in the toolbar to return to the notebook editor. Alternatively, click the *Notebook* menu item in the *View* menu.



CHAPTER 3

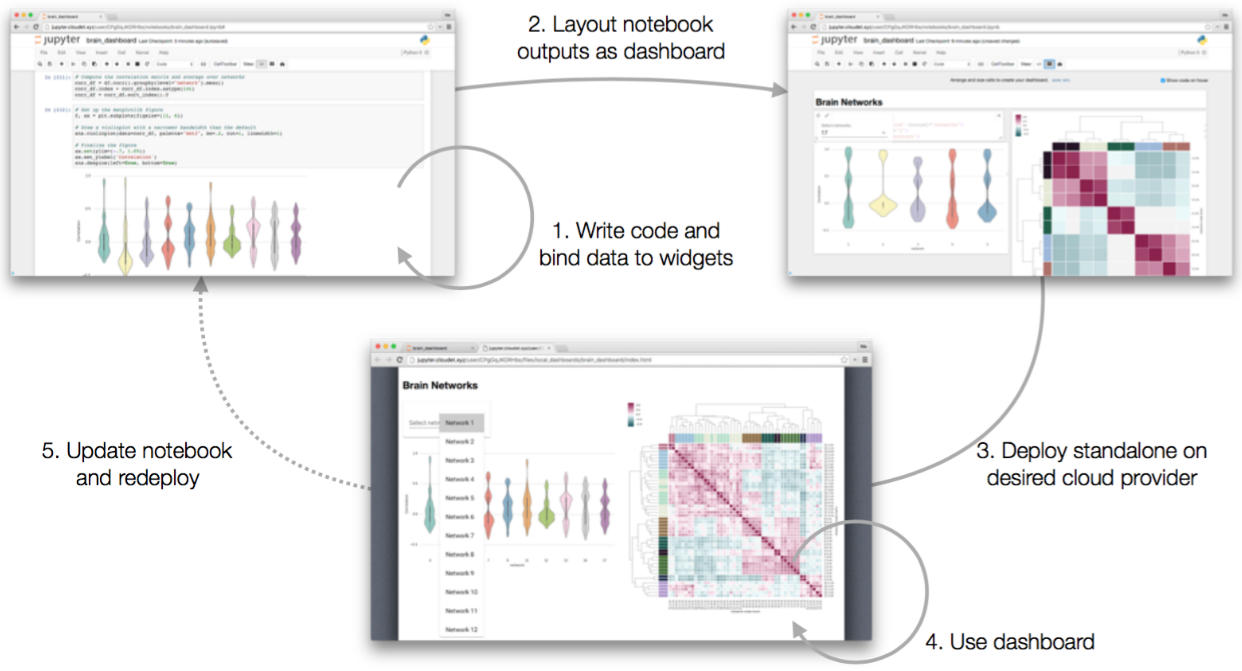
Understanding the use case

The dashboard layout extension is part of a larger Jupyter Dashboards effort meant to address the following problem:

> Alice is a Jupyter Notebook user. Alice prototypes data access, modeling, plotting, interactivity, etc. in a notebook. Now Alice needs to deliver a dynamic dashboard for non-notebook users. Today, Alice must step outside Jupyter Notebook and build a separate web application. Alice cannot directly transform her notebook into a secure, standalone dashboard application.

The solution implemented by the Dashboards effort is the following:

1. Alice authors a notebook document using Jupyter Notebook. She adds visualizations and interactive widgets.
2. Alice arranges her notebook cells in a grid- or report-like dashboard layout.
3. Alice one-click deploys her notebook and associated assets to a Jupyter Dashboards server.
4. Bob visits the dashboards server and interacts with Alice's notebook-turned-dashboard application.
5. Alice updates her notebook with new features and redeploys it to the dashboards server.


2. Layout notebook outputs as dashboard
1. Write code and bind data to widgets
Brain Networks
5. Update notebook and redeploy
3. Deploy standalone on desired cloud provider
4. Use dashboard

The ecosystem of widget and visualization libraries for Jupyter Notebook covers step (1). The dashboard layout extension handles step (2). The other incubating projects in the Jupyter Dashboards effort, namely the dashboard bundlers and dashboard server attempt to handle the remaining steps (3) through (5).

CHAPTER 4

Developer tasks

This document includes instructions development environment for the dashboards layout extension. It also includes common steps in the developer workflow such as running tests, building docs, etc.

Install conda on your system. Then clone this repository in a local directory.

```
# make a directory under ~ to put source
mkdir -p ~/projects
cd !$

# clone this repo
git clone https://github.com/jupyter/dashboards.git
```

Create a conda environment with the necessary dev and test dependencies.

```
cd dashboards
make env
```

Install the necessary JS dependencies. Re-run this command any time your `bower.json` or `package.json` changes.

```
make js
```

Run a notebook server with the dashboard extension enabled.

```
make notebook
```

Travis runs a small set of UI smoke tests using Selenium on Sauce Labs on every merge to the git master branch. You can run these tests locally if you install Selenium.

```
# install selenium first, e.g., on OSX
brew install selenium-server-standalone
# run the smoke tests
make test
```

ReadTheDocs builds the project documentation on every merge to git master. You can build the documentation locally as well.

```
make docs
```

Run `make help` to get a full list of development tasks.

CHAPTER 5

Maintainer tasks

This document includes instructions for typical maintainer tasks.

5.1 Build a package

To build a source tarball in `dist/`, run the following:

```
make sdist
```

5.2 Make a release

To start a new major/minor branch for its first release (e.g., 0.3.0):

```
git checkout master
git pull origin master
git checkout -b 0.3.x

git checkout master

# edit _version.py to bump to next major/minor (e.g., 0.4.0.dev)
# then ...

git add .
git commit -m 'Bump to 0.4.0.dev'
git push origin master
```

To make a patch release on a major/minor branch (e.g., 0.3.0):

```
# cherry-pick commits into the branch from master
# if there's multiple comments on master, do this ...
git checkout master
git checkout -b tmp-backport-0.3.x
git rebase -i 0.3.x
# delete any version bumps or other commits you don't want
# in the stable release branch from master
git checkout 0.3.x
git merge tmp-backport-0.3.x
git branch -D tmp-backport-0.3.x

# if there's only one or two commits, just use cherry-pick
# then ...
git checkout -b 0.3.x

# edit _version.py to remove the trailing 'dev' token
# then ...

git add .
git commit -m 'Release 0.3.0'
git tag 0.3.0

# do the release
make release

# edit _version.py to bump to 0.3.1.dev
# then ...

git add .
git commit -m 'Bump to 0.3.1.dev'

git push origin 0.3.x
git push origin 0.3.0
```

CHAPTER 6

Dashboard metadata and rendering

This page documents:

1. The fields written to notebook documents (`.ipynb` files) by the `jupyter/dashboards` extension
2. The interpretation of these fields in `jupyter/dashboards` and `jupyter-incubator/dashboards_server` to render a notebook in a dashboard layout.

6.1 Versioning

The dashboard metadata specification is versioned independently of the packages that use it. The current version of the specification is v1.

Prior to the v1 specification, the dashboard incubator projects read and wrote a legacy v0 metadata format. The details of this older spec appear on the dashboards wiki for historical purposes.

6.2 Notebook Fields

The following snippet of JSON shows the fields read and written by the dashboard projects. A more formal JSON schema appears later in this document.

```
{
  "metadata": {                               // notebook level metadata
    "extensions": {                           // to avoid future notebook conflicts
      "jupyter_dashboards" : {                // pypi package name
        "version": 1,                         // spec version
        "activeView": "<str:views key>",      // initial view to render
        "views": {
          "<str: tool defined ID 1>": {       // tool-assigned, unique layout ID
            "name": "<str>",                  // user-assigned, unique human readable name
            "type": "grid",                   // layout algorithm to use (grid in this␣
↪example view)
```

```
            "cellMargin": <uint:10>,        // margin between cells in pixels
            "cellHeight": <uint:20>,        // height in pixels of a logical row
            "numColumns": <uint:12>         // total number of logical columns
          },
          "<str: tool defined ID 2>": {     // tool-assigned, unique layout ID
            "name": "<str>",                // user-assigned, unique human readable name
            "type": "report"                // layout algorithm to use (report in this␣
↪example view)
          }
        }
      }
    }
  },
  "cells": [
    {
      "metadata": {
        "extensions": {
          "jupyter_dashboards": {
            "version": 1,                          // spec version
            "views": {
              "<str: tool defined ID 1>": {   // if present, means the grid layout␣
↪algorithm has processed this cell
                "hidden": <bool:false>,       // if cell output+widget are visible in␣
↪the layout
                "row": <uint:0>,              // logical row position
                "col": <uint:0>,              // logical column position
                "width": <uint:6>,            // logical width
                "height": <uint:2>            // logical height
              },
              "<str: tool defined ID 2>": {   // if present, means the report layout␣
↪algorithm has processed this cell
                "hidden": <bool:false>        // if cell output+widget are visible in␣
↪the layout
              }
            }
          }
        }
      }
    }
  ]
}
```

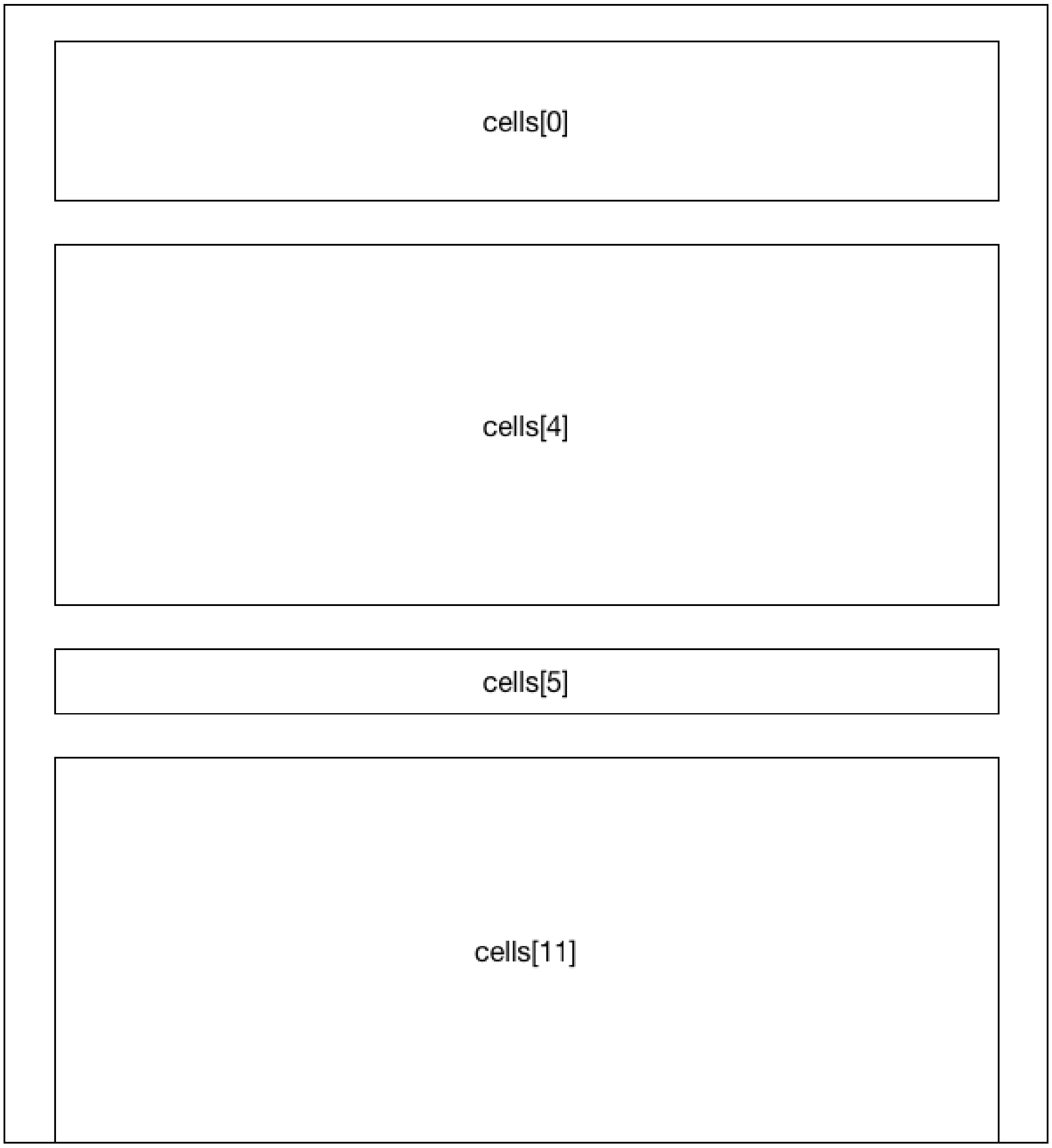
6.3 Rendering

A dashboard renderer is responsible for reading the notebook document, executing cell inputs, and placing *cell outputs* in a *dashboard view*. *Cell outputs* include anything that Jupyter Notebook 4.x renders in the cell output subarea or cell widget subarea in response to kernel messages or client-side events. A *dashboard view* defines how cell outputs are positioned and sized with respect to one another according to a particular layout algorithm.

The notebook can have multiple dashboard view associated with it in the `metadata.extensions.jupyter_dashboards.views` field. This specification defines two *view types*, report and grid, that dictate how a renderer positions and sizes cells on the page.

6.3.1 Report View

The `report` type stacks cell outputs top-to-bottom, hiding cells marked as hidden. The height of each cell varies automatically to contain its content. The width of all cells is equivalent and set by the renderer.



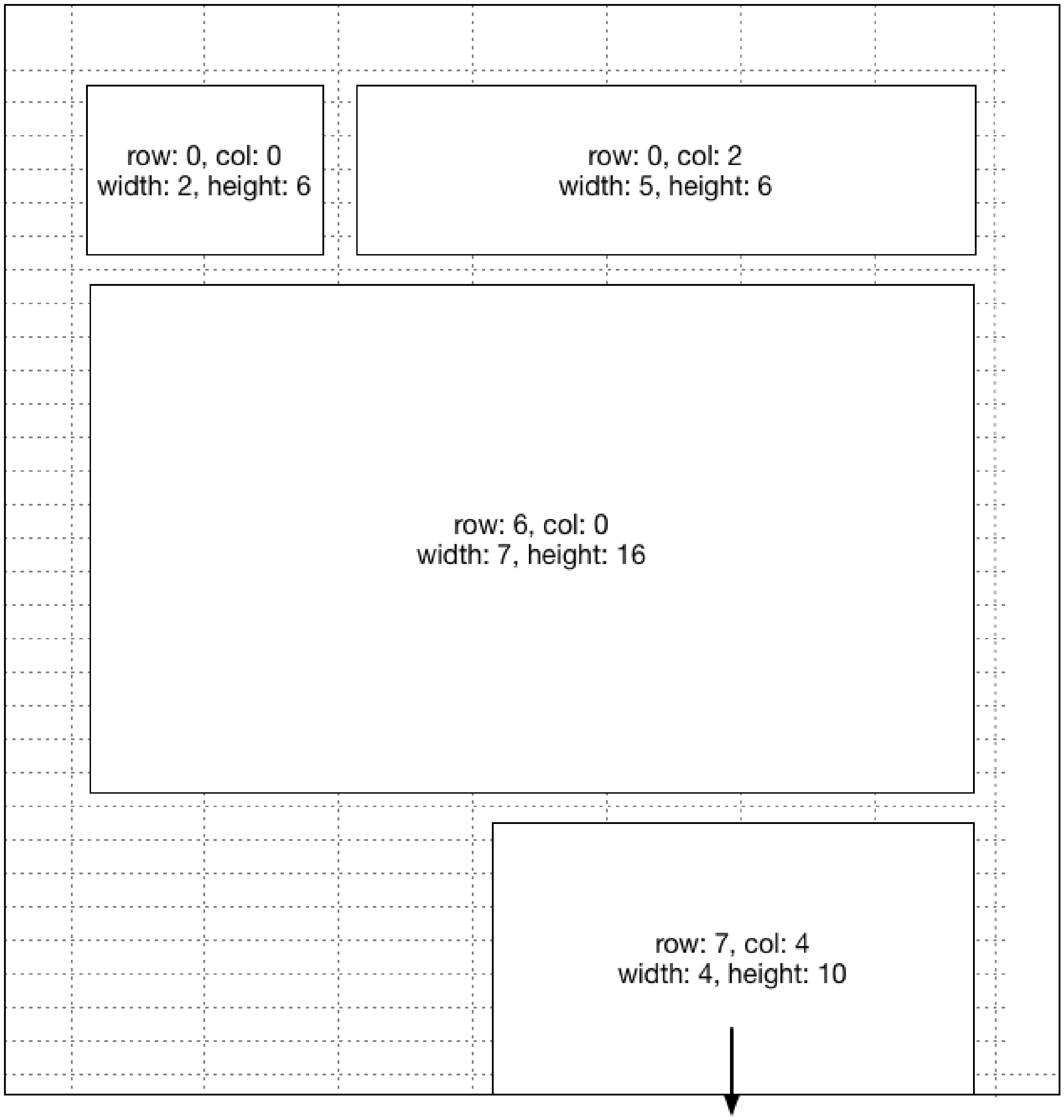
To display a `metadata.jupyter_dashboards.views[view_id]` with type `report` properly, the renderer:

- Must execute cell inputs in the order defined by the notebook `cells` array.
- Must not render nor reserve display space for cells that have `metadata.extensions.jupyter_dashboards.views[<view id>].hidden=true`.
- Must arrange cell outputs top-to-bottom in order of execution (i.e., stacked vertically).

- Must space cell outputs vertically so that they do overlap at any time (e.g., a plot in the top-most cell should not overlap Markdown in the next down cell nor any cell below that).
- Should allow interactive widgets in cell outputs to render content that does overlap other cells (e.g., popups).
- Should wrap cell outputs that have variable length content (e.g., text) at a renderer-determined width (e.g., browser width, responsive container element, fixed width).
- Should include a fixed amount of vertical whitespace between cell outputs.

6.3.2 Grid View

The `grid` type positions cells in a grid with infinitely many rows and a fixed number of columns. The width and height of each cell is expressed in terms of these rows and columns. The physical height of each row is a fixed value while the width of each column is set by the renderer.



Consider a view with tool-assigned ID `view_id` and type `grid` defined in the notebook-level metadata (i.e., `metadata.extensions.jupyter_dashboards.views[view_id]`). Let `view` be a reference to this notebook-level object. Let `cell_view` be a reference to any cell-level object keyed by the same `view_id`. To properly display this view, the renderer:

- Must execute cell inputs in the order defined by the notebook `cells` array.
- Must not render nor reserve display space for cells that have `cell_view.hidden=true`.
- Must define a logical grid with an unbounded number of rows and `view.numColumns` columns per row.
- Must define a screen viewport with infinite height and a renderer-determined width (e.g., browser width, responsive container element, fixed width).
- Must map the grid origin (row zero, column zero) to the top left corner of the viewport.

- Must allocate `view.cellHeight` pixels of space in the viewport to each grid row.
- Must allocate a fixed, renderer-determined number of pixels in the viewport to each grid column.
- Must place a cell's outputs in the `cell_view.row` and `cell_view.col` slot in the grid.
- Must allocate `cell_view.width` columns and `cell_view.height` rows of space in the grid for a cell's output.
- Must separate each slot in the grid on the screen by `view.cellMargin` pixels.
- May clip, scale, wrap, or let overflow cell output that is bigger than its allocated space on the screen.

6.3.3 Other Cases

When presented with a document having no `metadata.extensions.jupyter_dashboards.views` at the notebook-level, a renderer:

- Should process the document as if it defines a `report` view with all cells visible.
- May persist the implicit all-cells-visible report view to the document.

When processing cells that have no view ID corresponding to the current view being displayed, a display-only renderer with no authoring capability should treat such cells as hidden. A renderer with layout authoring capability:

- Should make a best effort attempt at determining the properties for the cell in the view based on the content of the cell.
  - e.g., Set `cell_view.hidden=false` if the cell produces no output.
  - e.g., Set `cell_view.row`, `cell_view.col`, `cell_view.width`, and `cell_view.height` to values that do not overlap other cells in a grid layout.
- Should immediately persist such default values into the document to avoid inferring them again in the future.
- Must allow the user to override the computed default values.

6.4 JSON Schema

The following schema expresses the dashboard layout specification as additions to the existing notebook format v4 schema. The schema below omits any untouched portions of the notebook schema for brevity.

```
{
    "$schema": "http://json-schema.org/draft-04/schema##",
    "description": "IPython Notebook v4.0 JSON schema plus layouts.",
    "properties": {
        "metadata": {
            "properties": {
                "extensions": {
                    "description": "Notebook-level namespace for extensions",
                    "type": "object",
                    "additionalProperties": true,
                    "properties": {
                        "jupyter_dashboards": {
                            "description": "Namespace for jupyter_dashboards notebook␣
↪metadata",
                            "type": "object",
                            "additionalProperties": true,
                            "properties": {
                                "version": {
                                    "description": "Version of the metadata spec",
                                    "type": "integer",
                                    "minimum": 1,
                                    "maximum": 1
                                },
                                "activeView": {
                                    "description": "ID of the view that should render␣
↪by default",
                                    "type": "string"
                                },
                                "views": {
                                    "description": "View definition",
                                    "type": "object",
                                    "additionalProperties": false,
                                    "patternProperties": {
                                        "^[a-zA-Z0-9_-]+$": {
                                            "type": "object",
                                            "oneOf": [{
                                                "$ref": "##definitions / jupyter_
↪dashboards / notebook_grid_view "
                                            }, {
                                                "$ref": "##definitions / jupyter_
↪dashboards / notebook_report_view "
                                            }]
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    },
    "definitions": {
        "raw_cell": {
            "properties": {
                "metadata": {
                    "properties": {
                        "extensions": {
                            "description": "Cell-level namespace for extensions",
                            "type": "object",
                            "additionalProperties": true,
                            "properties": {
                                "jupyter_dashboards": {"$ref": "##definitions/jupyter_
↪dashboards/cell_view"}
                            }
                        }
                    }
                }
            }
        },
        "markdown_cell": {
            "properties": {
                "metadata": {
                    "properties": {
                        "extensions": {
                            "description": "Cell-level namespace for extensions",
                            "type": "object",
                            "additionalProperties": true,
                            "properties": {
                                "jupyter_dashboards": {"$ref": "##definitions/jupyter_
↪dashboards/cell_view"}
                            }
                        }
                    }
                }
            }
        },
        "code_cell": {
            "properties": {
                "metadata": {
                    "properties": {
                        "extensions": {
                            "description": "Cell-level namespace for extensions",
                            "type": "object",
                            "additionalProperties": true,
                            "properties": {
                                "jupyter_dashboards": {"$ref": "##definitions/jupyter_
↪dashboards/cell_view"}
                            }
                        }
                    }
                }
            }
        },
        "jupyter_dashboards": {
            "notebook_grid_view": {
                "description": "Grid view definition",
                "type": "object",
                "additionalProperties": true,
                "properties": {
                    "name": {
                        "description": "Human readable name of the view",
                        "type": "string"
                    },
                    "type": {
                        "description": "Grid view type",
                        "enum": ["grid"]
                    },
                    "cellMargin": {
                        "description": "Margin between cells in pixels",
                        "type": "integer",
                        "minimum": 0
                    },
                    "cellHeight": {
                        "description": "Height of a logical row in pixels",
                        "type": "integer",
                        "minimum": 0
                    },
                    "numColumns": {
                        "description": "Total number of logical columns",
                        "type": "integer",
                        "minimum": 1
                    }
                }
            },
            "notebook_report_view": {
                "description": "Report view definition",
                "type": "object",
                "additionalProperties": true,
                "properties": {
                    "name": {
                        "description": "Human readable name of the view",
                        "type": "string"
                    },
                    "type": {
                        "description": "Report view type",
                        "enum": ["report"]
                    }
                }
            },
            "cell_view": {
                "description": "Namespace for jupyter_dashboards cell metadata",
                "type": "object",
                "additionalProperties": true,
                "properties": {
                    "version": {
                        "description": "Version of the metadata spec",
                        "type": "integer",
                        "minimum": 1,
                        "maximum": 1
                    },
                    "views": {
                        "description": "Layout information for cell in view",
                        "type": "object",
                        "additionalProperties": false,
                        "patternProperties": {
                            "^[a-zA-Z0-9_-]+$": {
                                "type": "object",
                                "oneOf": [{
                                    "$ref": "##definitions / jupyter_dashboards /␣
↪cell_grid_view "
                                }, {
                                    "$ref": "##definitions / jupyter_dashboards /␣
↪cell_report_view "
                                }]
                            }
                        }
                    }
                }
            },
            "cell_grid_view": {
                "description": "Grid view metadata for a cell",
                "type": "object",
                "additionalProperties": true,
                "properties": {
                    "hidden": {
                        "description": "True if cell is hidden in the view",
                        "type": "boolean"
                    },
                    "row": {
                        "description": "Logical grid row",
                        "type": "integer",
                        "minimum": 0
                    },
                    "col": {
                        "description": "Logical grid column",
                        "type": "integer",
                        "minimum": 0
                    },
                    "width": {
                        "description": "Width in logical columns",
                        "type": "integer",
                        "minimum": 1
                    },
                    "height": {
                        "description": "Height in logical rows",
                        "type": "integer",
                        "minimum": 1
                    }
                }
            },
            "cell_report_view": {
                "description": "Report view metadata for a cell",
                "type": "object",
                "additionalProperties": true,
                "properties": {
                    "hidden": {
                        "description": "True if cell is hidden in the view",
                        "type": "boolean"
                    }
                }
            }
        }
    }
}
```

CHAPTER 7

Summary of changes

See `git log` for a more detailed summary of changes.

7.1 0.7

7.1.1 0.7.0 (2017-04-04)

- Updated to support notebook 5.0
- Fixed clipping at cell boundaries

7.2 0.6

7.2.1 0.6.0 (2016-06-17)

- Switch to the v1 dashboard layout specification
- Automatically upgrade existing notebook metadata to the v1 spec
- Update example notebooks for compatibility with `jupyter_declarativewidgets` 0.6.0
- Remove `urth` moniker in favor of `jupyter_dashboards` for CSS classes, notebook metadata, etc.
- Fix gaps in grid when hiding cells

7.3 0.5

7.3.1 0.5.2 (2016-05-11)

- Fix report layout reset when switching between dashboard layout and preview

7.3.2 0.5.1 (2016-05-11)

- Hide errors from declarative widgets in dashboard layout and preview
- Fix the state of the show code checkbox in layout view when switching layout types
- Fix history window slider widgetin the community outreach demo
- Fix missing imports in the declarative widgets scotch demo
- Fix copy/pasted cells receive the same layout metadata
- Fix lost cells in report layout after clear and refresh
- Fix layout toolbar button default state

7.3.3 0.5.0 (2016-04-26)

- Add report layout for simple top-to-bottom, full-width dashboards
- Add buttons to move a cell to the top, bottom, or notebook order in layout mode
- Make compatible with Jupyter Notebook 4.0.x to 4.2.x
- Fix bokeh example race condition
- Fix browser scrolling when dragging cells in layout view

7.4 0.4

7.4.1 0.4.2 (2016-02-18)

- Fix code cell overflow in layout mode
- Fix scroll bars that appear within cells of a certain size
- Fix hidden cells from being cut-off in layout mode
- Fix failure to load extension JS in certain situations
- Fix meetup streaming demo filter box
- Update to Gridstack 0.2.4 to remove a workaround

7.4.2 0.4.1 (2016-02-07)

- Fix gridstack break with lodash>=4.0
- Remove notebook 4.1 cell focus highlight in dashboard preview
- Hide stderr and errors in dashboard preview, send them to the browser console

7.4.3 0.4.0 (2016-01-21)

- Separate `pip install` from `jupyter dashboards [install | activate | deactivate]`
- Match the Python package to the distribution name, `jupyter_dashboards`
- Fix cell overlap when one cell has the minimum height

- Prevent stderr and exception messages from displaying in dashboard modes
- Update demo notebooks to stop using deprecated `UrthData.setItem` from declarative widgets.

7.5 0.3

7.5.1 0.3.0 (2015-12-30)

- Make compatible with Jupyter Notebook 4.1.x
- Remove all download and deployment related backend code in. Refer users to the separate `jupyter_cms` and `jupyter_dashboards_bundlers` packages for these features.
- Keep compatible with Jupyter Notebook 4.0.x

7.6 0.2

7.6.1 0.2.2 (2015-12-15)

- Revert to old jupyter_notebook_server.py config hack to remain compatible with jupyter_declarativewidgets and jupyter_cms (until they change too)

7.6.2 0.2.1 (2015-12-15)

- Fix errors on install when profiles don't exist
- Fix styling leaking out of dashboard mode

7.6.3 0.2.0 (2015-12-01)

- Default to showing code instead of blank cells in layout mode
- Add menu items for packed vs stacked cell layout
- Make compatible with Jupyter Notebook 4.0.x
- Make compatible with jupyter_declarativewidgets 0.2.x
- System tests using Selenium locally, SauceLabs via Travis

7.7 0.1

7.7.1 0.1.1 (2015-12-02)

- Backport of UX fixes from 0.2.0
- Keep compatible with IPython Notebook 3.2.x
- Keep compatible with declarative widgets 0.1.x

7.7.2 0.1.0 (2015-11-17)

- First PyPI release
- Compatible with IPython Notebook 3.2.x on Python 2.7 or Python 3.3+

CHAPTER 8

Indices and tables

- genindex
- modindex
- search