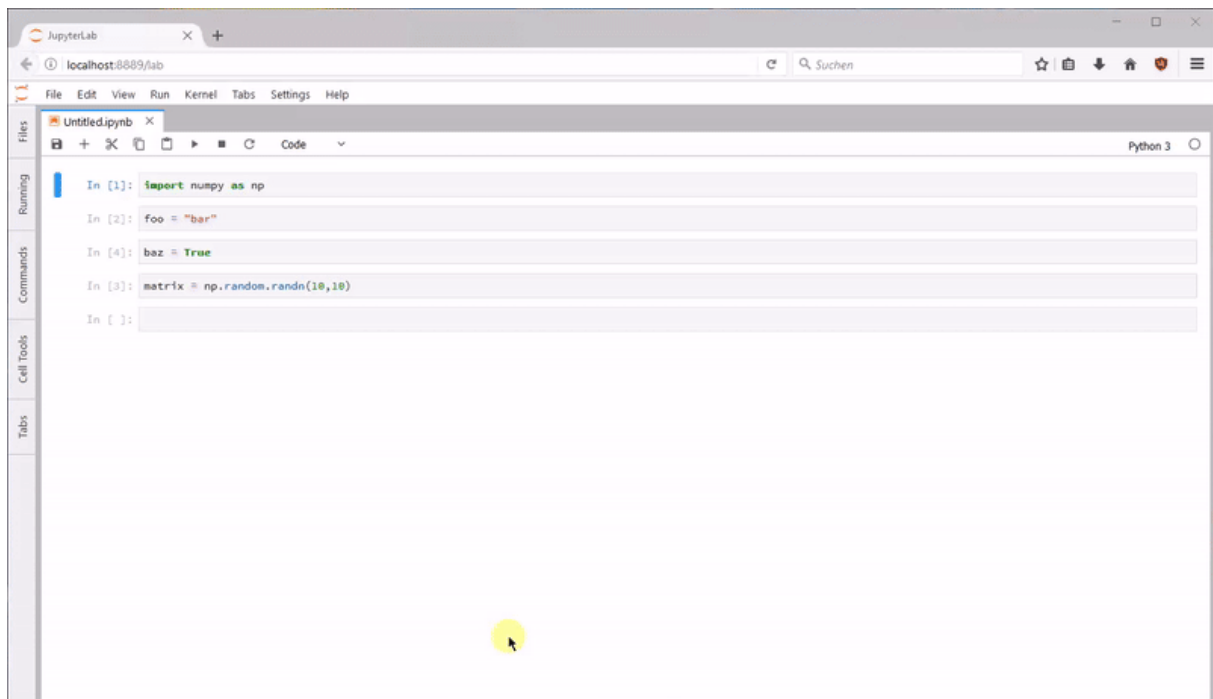

jupyterlab_variableinspector



Jupyterlab extension that shows currently used variables and their values.
Contributions in any form are welcome!

Features



- Allows inspection of variables for both consoles and notebooks.
- Allows inspection of matrices in a datagrid-viewer. This might not work for large matrices.
- Allows an inline and interactive inspection of Jupyter Widgets.

Supported Languages

- This extension is currently targets `python` as a main language but also supports the following languages with different levels of feature completeness
 - `R`
 - `scala` via the almond kernel

How it Works

In order to allow variable inspection, all content that is displayed first need to be sent from the kernel to the front end.

Therefore, opening large data frames with the datagrid viewer can dramatically increase your occupied memory and *significantly slow down* your browser.

Use at your own risk.

Requirements

- JupyterLab >= 3.0

Requirements for python functionality

- [pandas](#) and [numpy](#) are required to enable matrix inspection.
- [pyspark](#) for spark support.
- [tensorflow](#) and [keras](#) to allow inspection of tf objects.
- [torch](#) for PyTorch support.

Requirements for R functionality

- The [repr](#) library.

Requirements for ipywidgets functionality

The variable inspector can also display Jupyter interactive widgets:

The screenshot displays the JupyterLab environment. On the left, the `widgets.ipynb` notebook is open, showing a tree view of the widget hierarchy and two interactive components: a spreadsheet created with `ipysheet` and a 2D line plot created with `bqplot`. The spreadsheet has columns A, B, and C, with values 'Hello' and 'World'. The plot is titled 'First Example' and shows a noisy signal. On the right, the Variable Inspector is open, displaying a table of variables and their contents. The table includes variables like `cell1`, `cell2`, `cell_value`, `dropdown`, `fig`, `node2`, `s`, `scale`, `size`, `slider`, and `tree`. The `dropdown` variable is set to 'baz', and the `slider` variable is set to 4. The `tree` variable shows a tree structure with nodes 1, 2, 3, 4, and 5.

The requirements for this functionality are:

- `ipywidgets`, which can be installed with `pip install ipywidgets`.

Install

To install the extension, execute:

```
1 pip install lckr_jupyterlab_variableinspector
```

Uninstall

To remove the extension, execute:

```
1 pip uninstall lckr_jupyterlab_variableinspector
```

Contributing

Development install

Note: You will need NodeJS to build the extension package.

The `jupyterlab` command is JupyterLab's pinned version of yarn that is installed with JupyterLab. You may use `yarn` or `npm` in lieu of `jupyterlab` below.

```
1 # Clone the repo to your local environment
2 # Change directory to the lckr_jupyterlab_variableinspector directory
3 # Install package in development mode
4 pip install -e "."
5 # Link your development version of the extension with JupyterLab
6 jupyter labextension develop . --overwrite
7 # Rebuild extension Typescript source after making changes
8 jupyterlab build
```

You can watch the source directory and run JupyterLab at the same time in different terminals to watch for changes in the extension's source and automatically rebuild the extension.

```
1 # Watch the source directory in one terminal, automatically rebuilding
  when needed
2 jupyterlab watch
3 # Run JupyterLab in another terminal
4 jupyterlab
```

With the watch command running, every saved change will immediately be built locally and available in your running JupyterLab. Refresh JupyterLab to load the change in your browser (you may need to wait several seconds for the extension to be rebuilt).

By default, the `jupyterlab build` command generates the source maps for this extension to make it easier to debug using the browser dev tools. To also generate source maps for the JupyterLab core extensions, you can run the following command:

```
1 jupyterlab build --minimize=False
```

Development uninstall

```
1 pip uninstall lckr_jupyterlab_variableinspector
```

In development mode, you will also need to remove the symlink created by `jupyterlabextension develop` command. To find its location, you can run `jupyterlabextension list` to figure out where the `labextensions` folder is located. Then you can remove the symlink named `@lckr/jupyterlab_variableinspector` within that folder.

Testing the extension

Frontend tests This extension is using Jest for JavaScript code testing.

To execute them, execute:

```
1 jnpm
2 jnpm test
```

Integration tests This extension uses Playwright for the integration tests (aka user level tests). More precisely, the JupyterLab helper Galata is used to handle testing the extension in JupyterLab.

More information are provided within the ui-tests README.

Packaging the extension

See RELEASE