LocalVisualizer

Release v0.0.1

Apr 24, 2021

Contents

CHAPTER 1

Content

1.1 Indices and tables

- genindex
- modindex
- search

1.2 Table of Contents

1.2.1 Credits

Development Lead

- Vishnu P Sreenivasan <psvishnu.91@gmail.com>

Contributors

- Jimmy Retzlaff <https://github.com/jretz/>

1.2.2 Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

Types of Contributions

Report Bugs

Report bugs at https://github.com/psvishnu91/local_visualizer/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

Write Documentation

local_visualizer could always use more documentation, whether as part of the official local_visualizer docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/psvishnu91/local_visualizer/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

Get Started!

Ready to contribute? Here's how to set up *local_visualizer* for local development.

1. Fork the *local_visualizer* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/local_visualizer.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv local_visualizer
$ cd local_visualizer/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 local_visualizer tests
$ python setup.py test or py.test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.6, 2.7, 3.3, 3.4 and 3.5, and for PyPy. Check https://travis-ci.org/psvishnu91/local_visualizer/pull_requests and make sure that the tests pass for all supported Python versions.

Tips

To run a subset of tests:

```
$ py.test tests.test_local_visualizer
```

1.2.3 History

0.2.0 (2017-11-06)

The close method no more deletes the html but only makes the html valid.

0.1.0 (2017-11-05)

- First release on PyPI.

1.2.4 local_visualizer package

Submodules

local_visualizer.local_visualizer module

Simple api to visualize the plots in a script.

Motivation

- **When moving from an IPython notebook to a script, we lose the diagnostics** of visualizing pandas as tables and matplotlib plots.
- ***LocalViz* starts a local http server and creates a html file to** which pandas tables and matplotlib plots can be sent over.
- The html file is dynamically updated for long running scripts.

Usage

Sample Usage:

```
import logging, sys, numpy as np, pandas as pd, matplotlib.pyplot as plt
import local_visualizer

plt.style.use('fivethirtyeight')
logging.basicConfig(stream=sys.stdout, level=logging.DEBUG)

# Create the local visualizer instance
lviz = local_visualizer.LocalViz(html_file='lviz_test.html', port=9112)
# INFO:root:Starting background server at: http://localhost:9112/.
# INFO:local_visualizer:Click: http://carpediem:9112/lviz_test.html or http://
↪localhost:9112/lviz_test.html # noqa

# Create plots which will be streamed to the html file.
lviz.h3('Matplotlib :o')
lviz.p(
    'Wrap your plots in the figure context manager which takes '
    'in the kwargs of plt.figure and returns a plt.figure object.',
)

with lviz.figure(figsize=(10, 8)) as fig:
    x = np.linspace(-10, 10, 1000)
    plt.plot(x, np.sin(x))
    plt.title('Sine test')

lviz.hr()

# Visualize pandas dataframes as tables.
lviz.h3('Pandas dataframes')

df = pd.DataFrame({'A': np.linspace(1, 10, 10)})
df = pd.concat(
    [df, pd.DataFrame(np.random.randn(10, 4), columns=list('BCDE'))],
    axis=1,
```

```
)
lviz.write(df)
lviz.close()
```

Output

This starts a HTTPServer and creates a html file which is dynamically updated each time `lviz` is called. See https://i.imgur.com/jjwvAX2.png for the output of the above commands.

`local_visualizer.local_visualizer.`**`HEADER_LEVELS = [1, 2, 3, 4, 5]`**
The different HTML header levels.

class `local_visualizer.local_visualizer.`**`HtmlGenerator`**(*output_fl=None*)
Bases: `object`

A class which updates a html file and exposes API for the same.

The class also exposes the methods `h1`, `h2`, ..., `h6` for writing headers.

`br`()
Inserts a break line in the html file.

`figure`(**args*, ***kwds*)
Context manager as a stand it replacement for `plt.figure`.

Example usage:

```
with lviz.figure(figsize=(10, 10)) as fig:
    plt.plot(x, y)
    plt.title('This is a title')
```

`header`(*text*, *level=4*)
Creates a header line of given level.

Parameters

- **`text`** (*`str`*) – The html header text.
- **`level`** (*`int`*) – The level of the html header.

`hr`()
Inserts a horizontal line wrapped in blank lines in the html file.

`p`(*text*)
Writes a paragraph tagged text.

Parameters **`text`** (*`str`*) – The html paragraph text.

`write`(*text_or_df*)
Appends the text or a pandas df to the output file.

Parameters **`text_or_df`** (*`str or pandas.DataFrame`*) – The string or the pandas dataframe to be written to file.

class `local_visualizer.local_visualizer.`**`LocalViz`**(*lazy=False*, *html_file=None*, *run_server=True*, *port=9111*)
Bases: `object`

API for creating a html visualizer for python scripts.

All the public methods of *`HtmlGenerator`* are also exposed by this class.

See module docstring for usage.

Variables

- **html_file** (*str or NoneType*) – Path to the html file to write to. If the file exists already it will be overwritten. If `None` is passed in, the class will create a temp file.
- **run_server** (*bool*) – Whether the server should started in the background.
- **port** (*int*) – The port at which the server is to be started.
- **_html_gen** (`HtmlGenerator`) – A container for the html generation.
- **is_started** (*bool*) – Has the start been called.

close(**args*, ***kwargs*)
Writes the closing html tags to the html file.

del_html(**args*, ***kwargs*)
Deletes the generated html file.

Note: Mutates `self.html_file`.

---

inform_cleanup(**args*, ***kwargs*)
Informs the user which html file to delete at the end.

start()
Creates the html file and possibly starts the bgd http server.

Mutates

- `self.html_file`
- `self._html_gen`
- `self.is_started`

local_visualizer.local_visualizer.**delete_files_silently**(*files*)
Deletes a list of files if they exist.

Parameters **files** (*list of str*) – A list of file paths.

local_visualizer.local_visualizer.**run_bgd_server**(*port*, *host='localhost'*)
Creates a simple http server in a daemon thread.

Parameters

- **host** (*str*) – The host id where the server has to be started, ex. `'localhost'`.
- **port** (*int*) – The port where the local server should serve.

Returns A daemon thread running a simple http server in the background.

Type threading.Thread

local_visualizer.local_visualizer.**validate_lviz_started**(*method*)
Decorater for LocalViz methods to ensure the instance has been started.

Module contents

Top-level package for local_viz.

Python Module Index

l

Index