The Journal of Open Source Software

pylustrator: code generation for reproducible figures for publication

Richard Gerum[1]

1 Department of Physics, University of Erlangen-Nürnberg, Germany

DOI: 10.21105/joss.01989

Software

- Review
- Repository
- Archive

Editor: Lorena A Barba

Reviewers:

- @story645
- @tacaswell

Submitted: 04 October 2019
Published: 20 July 2020

Background

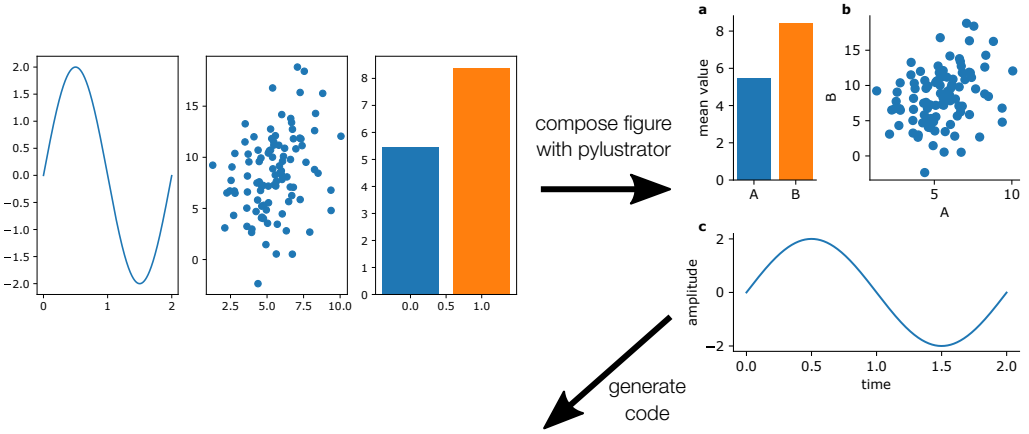
In recent years, more and more researchers have called attention to a growing "reproducibility crisis" (Sayre & Riegelman, 2018). An important factor contributing to problems in the reproducibility of results from published studies is the unavailability of the raw data from the original experiment and the unavailability of the methods or code used for the analysis of raw data (Baker & Penny, 2016). One major step to overcome these shortcomings is publishing all raw data and a documented version of the code used for analysis (Baker, 2016). Ideally, anyone interested should be able to download the raw data and reproduce the figures of the publication exactly.

To address the issue of data availability, researchers are encouraged to make their data available in online repositories such as Dryad ("Dryad," n.d.). However, these data are useless unless the complete analysis procedure, including all analysis and visualisation steps, can be comprehended by other scientists. The best way to achieve this is to provide a complete, well-documented analysis code, including all important steps from the basic artifact corrections to the final plot to be published. Open source scripting languages such as Python (Van Rossum & Drake Jr, 1995) or R (R Core Team, 2019) are ideal for such code because open source languages are accessible to everyone. In addition, interpreted languages do not need to be compiled, therefore present fewer obstacles for the user to run the code. The final part of the data analysis is the visualisation, which is crucial for communicating the results (Tufte, 1893). This paper deals with the visualization step, consisting of two parts: generating simple plots from data, and composing meaningful figures from these plots.

The first part of generating the building blocks of figures, the plots, is already covered in various toolkits, e.g., Matplotlib (Hunter, 2007), Bokeh (Bokeh Development Team, 2019) or Seaborn (Waskom et al., 2017). There already exist some software that records user interactions to generate 3D plots (Ahrens, Geveci, & Law, 2005, p. @westenberger2008avizo,@dragonfly), but no convenient Python toolkit is yet available to generate reproducible figures from simple plots scripts. Matplotlib offers figures composed of several subplots, but to create a complete, publication-ready figure a lot of code is needed to add all formatting, annotation and styling commands. Users often prefer graphical tools like image manipulation software, e.g., GIMP (GIMP Development Team, 2019) or Inkscape (Team, 2019). These offer great flexibility, but do not provide a reproducible way of generating figures and bear the risk of accidentally changing data points. It is also important to note that when using an image manipulation software, every small change in the analysis requires re-editing the figure in the image manipulation software. This process slows down the creation of figures and is prone to errors. The `pylustrator` package was developed to address this issue.

Algorithm and Examples

`Pylustrator` fills the gap from single plots to complete figures via a code-generation algorithm that converts user input into Python code for reproducible figure assembly (Fig. 1). Minor changes to the analysis or new data only require re-execution of the code to update the figure.




```
#% start: automatic generated code from pylustrator
plt.figure(1).set_size_inches(8.000000/2.54, 8.000000/2.54, forward=True)
plt.figure(1).axes[0].set_position([0.191879, 0.148168, 0.798133, 0.742010])
plt.figure(1).axes[0].set_xlabel("data x")
plt.figure(1).axes[0].set_ylabel("data y")
plt.figure(1).axes[1].set_position([0.375743, 0.603616, 0.339534, 0.248372])
plt.figure(1).axes[1].set_xlabel("data x")
plt.figure(1).axes[1].set_ylabel("data y")
plt.figure(1).axes[1].set_ylim(-40.0, 90.0)
#% end: automatic generated code from pylustrator
```

Figure 1: Example of composing a figure with pylustrator.

Using `pylustrator` in any Python file that uses Matplotlib to plot data requires only the addition of two lines of code:

```
import pylustrator
pylustrator.start()
```

The Matplotlib figure is then displayed in an interactive window (Fig. 2) when the command `plt.show()` is called. In this interactive window `pylustrator` allows the user to:

- resize and position plots by dragging with the mouse
- adjust the position of plots legends
- align elements easily through automatic "snap-in" mechanism
- change the size of the whole figure in cm/inch
- add text and annotations and change their style and color
- adjust plot ticks and tick labels

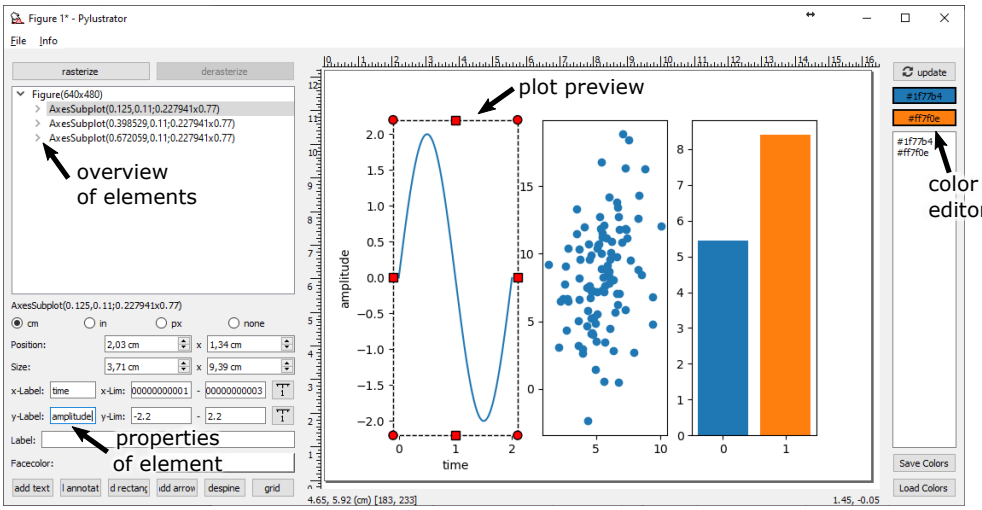


Figure 2: The interface of `pylustrator`. The user can view the elements of the plot, edit their properties, edit them in the plot preview and experiment with different color schemes.

`pylustrator` tracks all changes to the figure and translates them into Python code. To do so, the internal representation of changes has to fulfill some requirements. Changes need to be able to be replaced by newer changes that affect the same property of the same object, they need to be able to be converted to code, and changes need to be retrieved from generated code when loading a file that has already `pylustrator`-generated code in it.

Each change is defined by two parts: the affected object (e.g., a text object) and the affected property (e.g., its color). If a change has the same object and property as a previous change, it overwrites the previous change.

Changes are converted to code by first serializing the affected object by iteratively going up the parent-child tree from, e.g., a text object, to the axis that contains the text to the figure that contains the axis. From this dependency relation, a Python code segment is generated (e.g., `plt.figure(1).axes[0].texts[0]`, the first text of the first axis of figure 1). Then the property command is added (e.g., `.set_color("#ff0000ff")`). When saving, `pylustrator` introspects its current execution stack to find the line of code from where it was called and inserts the automatically generated code directly before the command calling `pylustrator`.

When a file with automatically generated code is loaded (see code example in figure 1), `pylustrator` splits all the automatically generated lines into the affected objects and affected properties. New changes, where both the affected object and the affected property match a previous change, overwrite the previous change. This ensures that previously generated code can be loaded appropriately, and saving the same figure multiple times does not generate the line of code for this change multiple times.

It is important to note that the automatically generated code only relies on Matplotlib and does not need the `pylustrator` package anymore. Thus, the `pylustrator` import can later be removed to allow sharing the code without an additional introduced dependency.

The documentation of `pylustrator` can be found on https://pylustrator.readthedocs.org.

Conclusion

This packages offers an improvement to create publishable figures from single plots based on an open source Python library called `pylustrator`. The figures can be arranged by drag-and-

drop, and the `pylustrator` library generates the corresponding code. This library provides a valuable contribution to improve reproducibility of scientific results.

Acknowledgements

We acknowledge testing, support and feedback from Christoph Mark, Sebastian Richter, and Achim Schilling and Ronny Reimann for the design of the Pylustrator Logo.

References

Ahrens, J., Geveci, B., & Law, C. (2005). Paraview: An end-user tool for large data visualization. *The visualization handbook*, *717*.

Baker, M. (2016). Why scientists must share their research code. *Nature News*. doi:10.1038/nature.2016.20504

Baker, M., & Penny, D. (2016, May). Is there a reproducibility crisis? doi:10.1038/533452A

Bokeh Development Team. (2019). *Bokeh: Python library for interactive visualization*. Retrieved from https://bokeh.org/

Dragonfly. (n.d.). https://www.theobjects.com/dragonfly/.

Dryad. (n.d.). https://datadryad.org.

GIMP Development Team. (2019). *GIMP:GNU image manipulation program*. Retrieved from https://gimp.org/

Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Comput. Sci. Eng.*, *9*(3), 90–95. doi:10.1109/MCSE.2007.55

R Core Team. (2019). *R: A language and environment for statistical computing*. Vienna, Austria: R Foundation for Statistical Computing. Retrieved from https://www.R-project.org

Sayre, F., & Riegelman, A. (2018). The reproducibility crisis and academic libraries. *College & Research Libraries*, *79*(1), 2. doi:10.5860/crl.79.1.2

Team, T. I. (2019). *Inkscape*. Retrieved from https://inkscape.org/

Tufte, E. (1893). *The visual display of quantitative information*. Cheshire, Connecticut: Graphics Press.

Van Rossum, G., & Drake Jr, F. L. (1995). *Python tutorial*. Centrum voor Wiskunde en Informatica Amsterdam, The Netherlands.

Waskom, M., Botvinnik, O., O'Kane, D., Hobson, P., Lukauskas, S., Gemperline, D. C., Augspurger, T., et al. (2017, September). Mwaskom/seaborn: V0.8.1 (september 2017). doi:10.5281/zenodo.883859

Westenberger, P. (2008). AVIZO-3D visualization framework. In *Geoinformatics conference* (pp. 1–11).