`plotastic`: Bridging Plotting and Statistics in Python

Martin Kuric ⓘ [1],[¶] **and Regina Ebert** ⓘ [1]

1 Department of Musculoskeletal Tissue Regeneration, University of Würzburg, Germany ¶ Corresponding author

Summary

`plotastic` addresses the challenges of transitioning from exploratory data analysis to hypothesis testing in Python's data science ecosystem. Bridging the gap between `seaborn` and `pingouin`, this library offers a unified environment for plotting and statistical analysis. It simplifies the workflow with user-friendly syntax and seamless integration with familiar `seaborn` parameters (y, x, hue, row, col). Inspired by `seaborn`'s consistency, `plotastic` utilizes a `DataAnalysis` object to intelligently pass parameters to `pingouin` statistical functions. Hence, statistics and plotting are performed on the same set of parameters, so that the strength of `seaborn` in visualizing multidimensional data is extended onto statistical analysis. In essence, `plotastic` translates `seaborn` parameters into statistical terms, configures statistical protocols based on intuitive plotting syntax and returns a `matplotlib` figure with known customization options and more. This approach streamlines data analysis, allowing researchers to focus on correct statistical testing and less about specific syntax and implementations.

Statement of need

Python's data science ecosystem provides powerful tools for both visualization and statistical testing. However, the transition from exploratory data analysis to hypothesis testing can be cumbersome, requiring users to switch between libraries and adapt to different syntaxes. `seaborn` has become a popular choice for plotting in Python, offering an intuitive interface. Its statistical functionality focuses on descriptive plots and bootstrapped confidence intervals (Waskom, 2021). The library `pingouin` offers an extensive set of statistical tests, but it lacks integration with common plotting capabilities (Vallat, 2018). `statannotations` integrates statistical testing with plot annotations, but uses a complex interface and is limited to pairwise comparisons (Charlier et al., 2022).

`plotastic` addresses this gap by offering a unified environment for plotting and statistical analysis. With an emphasis on user-friendly syntax and integration of familiar `seaborn` parameters, it simplifies the process for users already comfortable with seaborn. The library ensures a smooth workflow, from data import to hypothesis testing and visualization.

Example

The following code demonstrates how `plotastic` analyzes the example dataset "fmri", similar to Waskom (2021) (Figure 1).

```
### IMPORT PLOTASTIC
import plotastic as plst

# IMPORT EXAMPLE DATA
DF, _dims = plst.load_dataset("fmri", verbose = False)
```

```python
# EXPLICITLY DEFINE DIMENSIONS TO FACET BY
dims = dict(
    y = "signal",     # y-axis, dependent variable
    x = "timepoint",  # x-axis, independent variable (within-subject factor)
    hue = "event",    # color,  independent variable (within-subject factor)
    col = "region"    # axes,   grouping variable
)

# INITIALIZE DATAANALYSIS OBJECT
DA = plst.DataAnalysis(
    data=DF,           # Dataframe, long format
    dims=dims,         # Dictionary with y, x, hue, col, row
    subject="subject", # Datapoints are paired by subject (optional)
    verbose=False,     # Print out info about the Data (optional)
)

# STATISTICAL TESTS
DA.check_normality()   # Check Normality
DA.check_sphericity()  # Check Sphericity
DA.omnibus_rm_anova()  # Perform RM-ANOVA
DA.test_pairwise()     # Perform Posthoc Analysis

# PLOTTING
(DA
 .plot_box_strip()   # Pre-built plotting function initializes plot
 .annotate_pairwise( # Annotate results from DA.test_pairwise()
    include="__HUE" # Use only significant pairs across each hue
    )
)
```
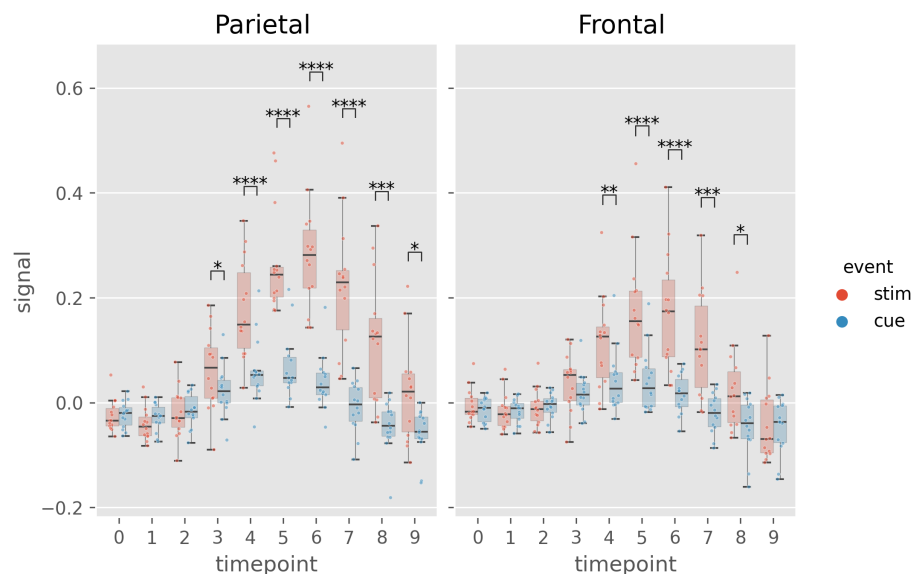

Figure 1: Example figure of `plotastic` (version 0.1). Image style was set by `plt.style.use("ggplot")`

Table 1: Results from `DA.check_sphericity()`. `plotastic` assesses sphericity after grouping the data by all grouping dimensions (hue, row, col). For example, `DA.check_sphericity()` grouped the 'fmri' dataset by "region" (col) and "event" (hue), performing four subsequent sphericity tests for four datasets.

'region', 'event'	spher	W	chi2	dof	pval	group count	n per group
'frontal', 'cue'	True	3.26e+20	-462.7	44	1	10	[14]
'frontal', 'stim'	True	2.45e+17	-392.2	44	1	10	[14]
'parietal', 'cue'	True	1.20e+20	-452.9	44	1	10	[14]
'parietal', 'stim'	True	2.44e+13	-301.9	44	1	10	[14]

Table 2: Results of `DA.omnibus_rm_anova()`. `plotastic` performs one two-factor RM-ANOVA per axis (grouping the data by row and col dimensions) using x and hue as the within-factors. For this example, `DA.omnibus_rm_anova()` grouped the 'fmri' dataset by "region" (col), performing two subsequent two-factor RM-ANOVAs. Within-factors are "timepoint" (x) and "event" (hue). For conciceness, GG-Correction and effect sizes are not shown.

'region'	Source	SS	ddof1	ddof2	MS	F	p-unc	stars
'parietal'	timepoint	1.583	9	117	0.175	26.20	3.40e-24	****
'parietal'	event	0.770	1	13	0.770	85.31	4.48e-07	****
'parietal'	timepoint * event	0.623	9	117	0.069	29.54	3.26e-26	****
'frontal'	timepoint	0.686	9	117	0.076	15.98	8.28e-17	****
'frontal'	event	0.240	1	13	0.240	23.44	3.21e-4	***
'frontal'	timepoint * event	0.242	9	117	0.026	13.031	3.23e-14	****

Overview

The functionality of `plotastic` revolves around a seamless integration of statistical analysis and plotting, leveraging the capabilities of `pingouin`, `seaborn`, `matplotlib` and `statannotations` (Charlier et al., 2022; Hunter, 2007; Vallat, 2018; Waskom, 2021). It utilizes long-format pandas `DataFrames` as its primary input, aligning with the conventions of `seaborn` and ensuring compatibility with existing data structures (McKinney, 2010; Team, 2020; Wickham, 2014).

`plotastic` was inspired by `seaborn` using the same set of intuitive and consistent parameters (y, x, hue, row, col) found in each of its plotting functions (Waskom, 2021). These parameters intuitively delineate the data dimensions plotted, yielding 'facetted' subplots, each presenting y against x. This allows for rapid and insightful exploration of multidimensional relationships. `plotastic` extends this principle to statistical analysis by storing these `seaborn` parameters (referred to as dimensions) in a `DataAnalysis` object and intelligently passing them to statistical functions of the `pingouin` library. This approach is based on the impression that most decisions during statistical analysis can be derived from how the user decides to arrange the data in a plot. This approach also prevents code repetition and streamlines statistical analysis. For example, the subject keyword is specified only once during `DataAnalysis` initialisation, and `plotastic` selects the appropriate paired or unpaired version of the test. Using `pingouin` alone requires the user to manually pick the correct test and to repeatedly specify the subject keyword in each testing function.

In essence, `plotastic` translates plotting parameters into their statistical counterparts. This translation minimizes user input and also ensures a coherent and logical connection between plotting and statistical analysis. The goal is to allow the user to focus on choosing the

correct statistical test (e.g. parametric vs. non-parametric) and worry less about specific implementations.

At its core, `plotastic` employs iterators to systematically group data based on various dimensions, aligning the analysis with the distinct requirements of tests and plots. Normality testing is performed on each individual sample, which is achieved by splitting the data by all grouping dimensions and also the x-axis (hue, row, col, x). Sphericity and homoscedasticity testing is performed on a complete sampleset listed on the x-axis, which is achieved by splitting the data by all grouping dimensions (hue, row, col) (Table 1). For omnibus and posthoc analyses, data is grouped by the row and col dimensions in parallel to the `matplotlib` axes, before performing one two-factor analysis per axis using x and hue as the within/between-factors. (Table 2).

`DataAnalysis` visualizes data through predefined plotting functions designed for drawing multi-layered plots. A notable emphasis within `plotastic` is placed on showcasing individual datapoints alongside aggregated means or medians. In detail, each plotting function initializes a `matplotlib` figure and axes using `plt.subplots()` while returning a `DataAnalysis` object for method chaining. Axes are populated by seaborn plotting functions (e.g., `sns.boxplot()`), leveraging automated aggregation and error bar displays. Keyword arguments are passed to these seaborn functions, ensuring the same degree of customization. Users can further customize plots by chaining `DataAnalysis` methods or by applying common `matplotlib` code to override `plotastic` settings. Figures are exported using `plt.savefig()`.

`plotastic` also focuses on annotating statistical information within plots, seamlessly incorporating p-values from pairwise comparisons using `statannotations` (Charlier et al., 2022). This integration simplifies the interface and enables options for pair selection in multidimensional plots, enhancing both user experience and interpretability.

For statistics, `plotastic` integrates with the `pingouin` library to support classical assumption and hypothesis testing, covering parametric/non-parametric and paired/non-paired variants. Assumptions such as normality, homoscedasticity, and sphericity are tested. Omnibus tests include two-factor RM-ANOVA, ANOVA, Friedman, and Kruskal-Wallis. Posthoc tests are implemented through `pingouin.pairwise_tests()`, offering (paired) t-tests, Wilcoxon, and Mann-Whitney-U.

To sum up, `plotastic` stands as a unified and user-friendly solution catering to the needs of researchers and data scientists, seamlessly integrating statistical analysis with the power of plotting in Python. It streamlines the workflow, translates seaborn parameters into statistical terms, and supports extensive customization options for both analysis and visualization.

Acknowledgments

References

Charlier, F., Weber, M., Izak, D., Harkin, E., Magnus, M., Lalli, J., Fresnais, L., Chan, M., Markov, N., Amsalem, O., Proost, S., Agamemnon Krasoulis, Getzze, & Repplinger, S. (2022). *Trevismd/statannotations: V0.5*. Zenodo. https://doi.org/10.5281/ZENODO.7213391

Hunter, J. D. (2007). Matplotlib: A 2D Graphics Environment. *Computing in Science & Engineering*, *9*(3), 90–95. https://doi.org/10.1109/MCSE.2007.55

McKinney, W. (2010). *Data Structures for Statistical Computing in Python*. 56–61. https://doi.org/10.25080/Majora-92bf1922-00a

Team, T. P. D. (2020). *Pandas-dev/pandas: Pandas*. Zenodo. https://doi.org/10.5281/zenodo.3509134

Vallat, R. (2018). Pingouin: Statistics in Python. *Journal of Open Source Software*, *3*(31), 1026. https://doi.org/10.21105/joss.01026

Waskom, M. L. (2021). Seaborn: Statistical data visualization. *Journal of Open Source Software*, *6*(60), 3021. https://doi.org/10.21105/joss.03021

Wickham, H. (2014). Tidy Data. *Journal of Statistical Software*, *59*, 1–23. https://doi.org/10.18637/jss.v059.i10