

Violin SuperPlots: visualizing replicate heterogeneity in large data sets.

AUTHOR(S)

Martin Kenny, Ingmar Schoen

CITATION

Kenny, Martin; Schoen, Ingmar (2021): Violin SuperPlots: visualizing replicate heterogeneity in large data sets.. Royal College of Surgeons in Ireland. Journal contribution.
<https://hdl.handle.net/10779/rcsi.15073512.v3>

HANDLE

[10779/rcsi.15073512.v3](https://hdl.handle.net/10779/rcsi.15073512.v3)

LICENCE

CC BY 4.0

This work is made available under the above open licence by RCSI and has been printed from <https://repository.rcsi.com>. For more information please contact repository@rcsi.com

URL

https://repository.rcsi.com/articles/journal_contribution/Violin_SuperPlots_visualizing_replicate_heterogeneity_in_large_data_sets_/15073512/3

Supplementary Information accompanying the article

Violin SuperPlots: Visualising replicate heterogeneity in large datasets

M. Kenny and I. Schoen

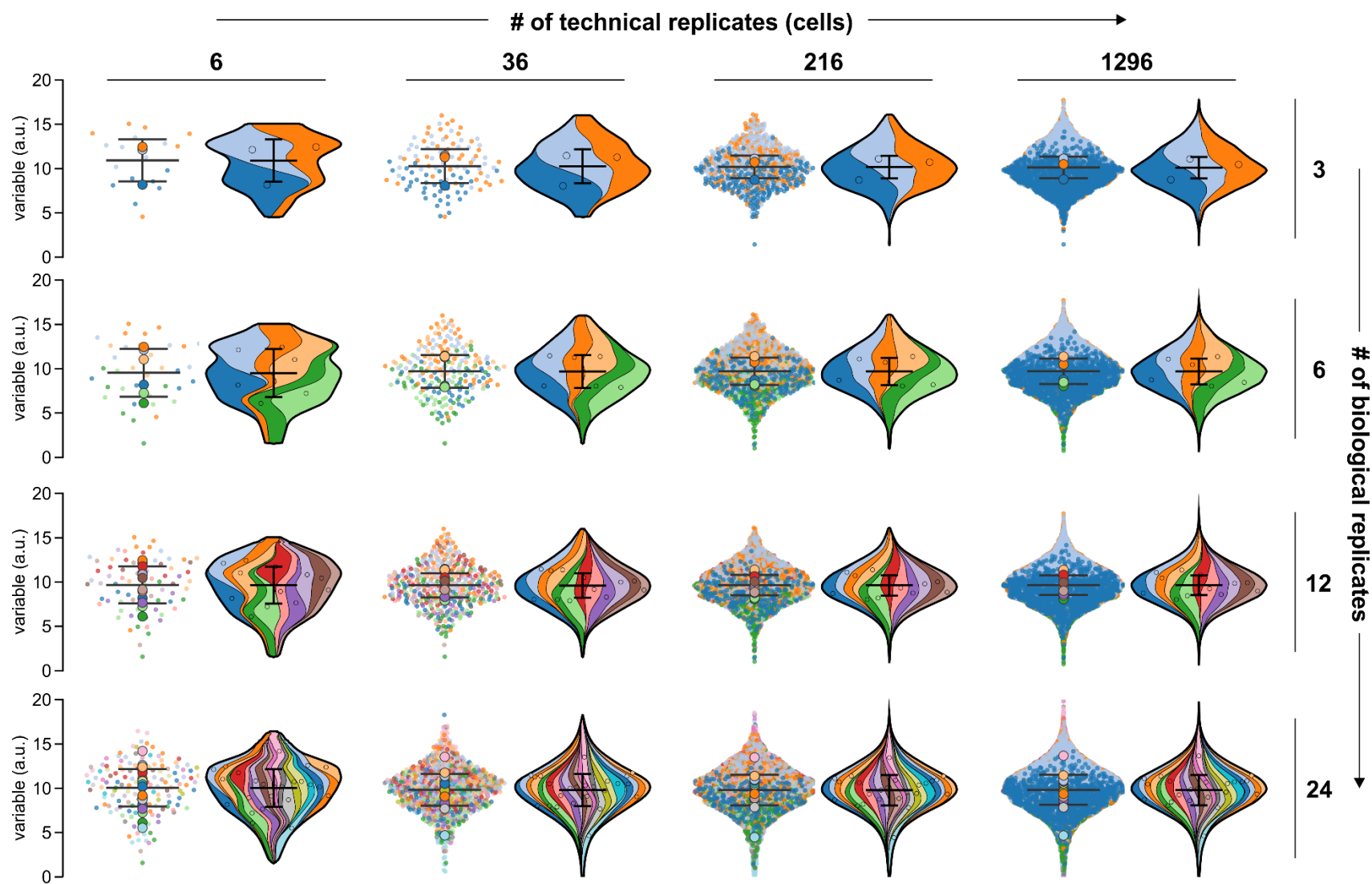
Molecular Biology of the Cell, Volume 32, July 15, 2021

doi: [10.1091/mbc.E21-03-0130](https://doi.org/10.1091/mbc.E21-03-0130)

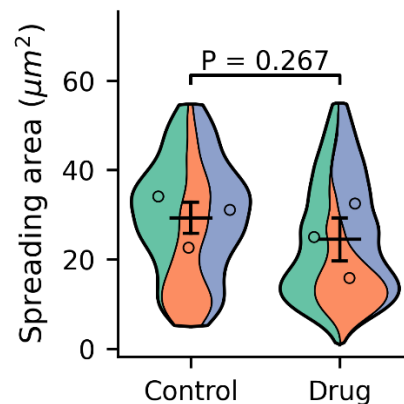
Table of contents

Supplementary Figures	page 2
User documentation of the 'Superviolin' Python package	page 6
Appendix A: Basic implementation of Violin SuperPlots in MATLAB	page 13

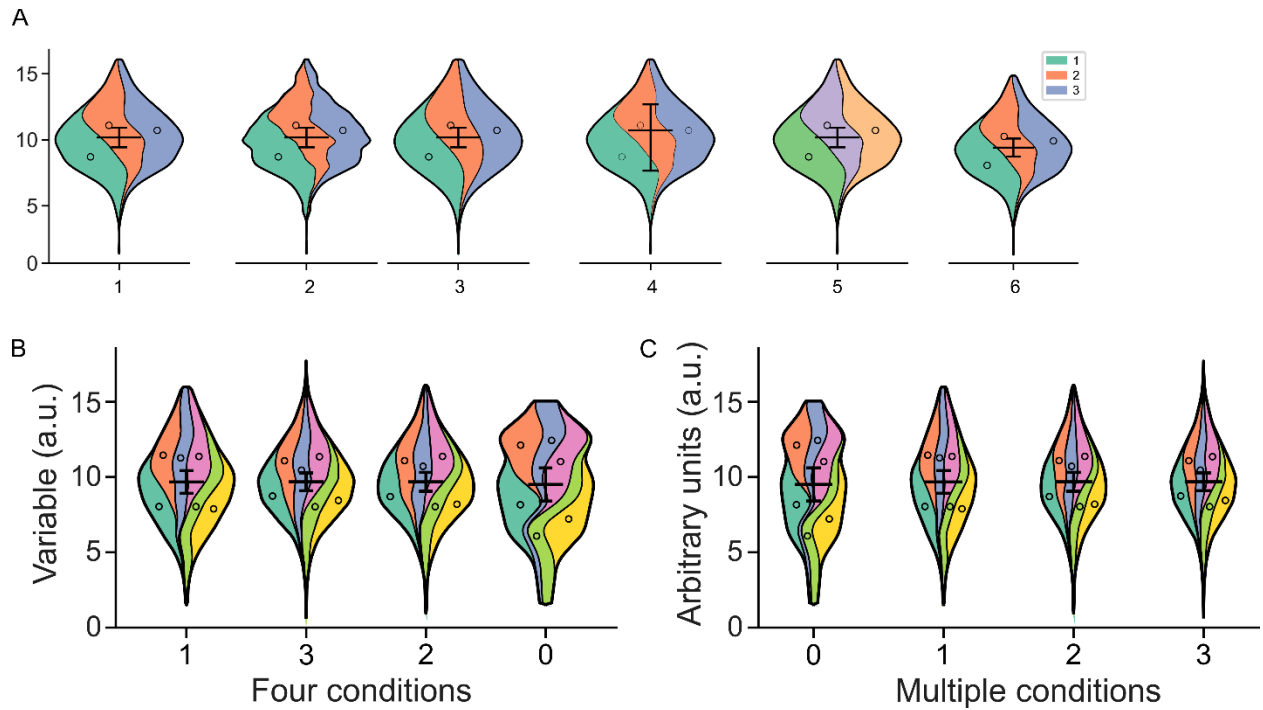
Supplementary Figures



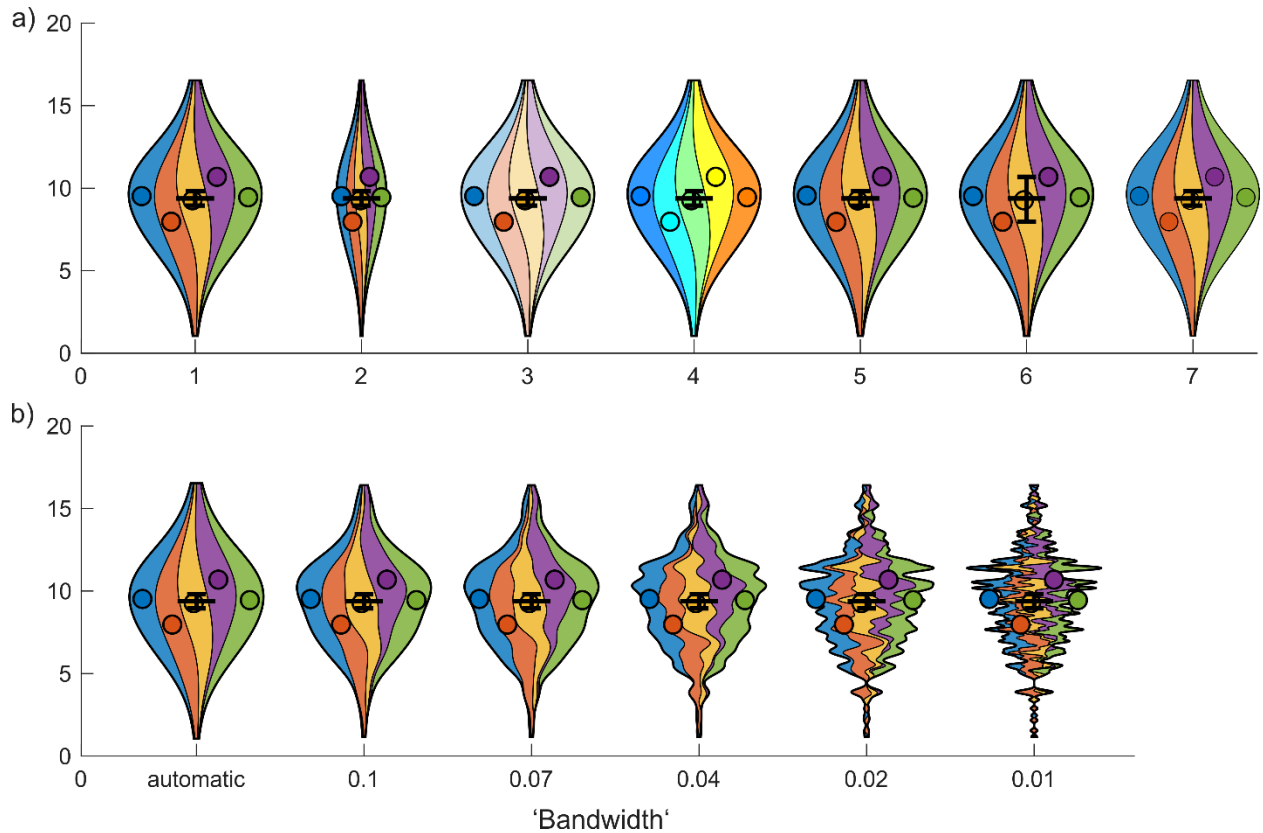
◀ **Supplementary Figure S1.** *Comparison of Beeswarm SuperPlots and Violin SuperPlots using simulated data with varied numbers of biological (3, 6, 12, 24) and technical (6, 36, 216, 1296) replicates.* Technical replicates representing e.g. single cell data were randomly drawn from normal distributions with means and standard deviations varying between biological replicates. Beeswarm SuperPlots were created using the SuperPlotsOfData web app by Joachim Goedhart (<https://huygens.science.uva.nl/SuperPlotsOfData/>). The same data was plotted using the Superviolin Python package and is shown side-by-side for comparison. Violin SuperPlots provide a fast visual assessment of heterogeneity between and within biological replicates in all cases, with a remarkably similar perception despite the large variation in numbers. Limitations arise for many (>18) biological replicates when the colour code is not unique anymore, as well as for few (<10) technical replicates where the density estimation kernel is smoothing out the discrete distribution of the raw data. Beeswarm SuperPlots work well for low numbers of technical and biological replicates. However, the heterogeneity within biological replicates becomes difficult to assess for 6 or more colour-coded biological replicates. Moreover, with increasing number of data points from either many technical replicates and/or many biological replicates, not all data points are visible anymore despite their partial transparency, which can lead to the complete masking of datasets in the background, as seen especially for >200 technical replicates.



Supplementary Figure S2. *Violin SuperPlot generated by the running the superviolin demo command from the Anaconda command line prompt.*



Supplementary Figure S3. Examples for the usage of optional name-value pairs and their effect on the appearance of Violin SuperPlots in the Python Superviolin package. For a detailed description of optional parameters, see section 6 (pages 11-12). a) 1: default settings (the smoothing bandwidth (bw) calculated by Scott's Method was 0.341). 2: bw = 0.2. 3: bw = 0.6. 4: centre_val: median (value used to plot the middle line of the summary statistics), error_bars: CI (use 95% confidence intervals for the error bars), sep_linewidth: 0.2 (changes the width of the lines separating the stripes in each Violin SuperPlot). 5: cmap: Accent (the colours used to represent each replicate on the Violin SuperPlot). 6: show_legend: yes (whether to show a legend which provides the replicate names). b) xlabel: Four conditions, ylabel: Variable (AU), order: 1, 3, 2, 0 (order of the conditions on the x-axis). c) total_width: 0.5 (the width of each Violin SuperPlot in the figure), xlabel: Multiple conditions, ylabel: Arbitrary units (AU).



Supplementary Figure S4. Examples for the usage of optional name-value pairs and their effect on the appearance of Violin SuperPlots in the MATLAB implementation. For a detailed description of optional parameters, see Appendix A (pages 13-15). Simulated data contained 50-100 technical replicates each per biological replicate. a) 1: default settings. 2: 'Width', 0.3; changes the overall width of the violin. 3: 'FaceAlpha', 0.35; changes the transparency of stripe colours. 4: 'LUT', 'jet'; changes the look up table for the coloured stripes. 5: 'Centrals', 'robustmean'; determines which centrality measure is displayed as a circle overlaid with each stripe. 6: 'Errorbars', 'ci'; changes the error bars. 7: 'LineWidth', 0.1; changes the line width of stripe outlines and marker edges. b) Variation of the 'Bandwidth' parameter which determines the smoothing of the histogram data. Small bandwidths more closely reflect the raw data at the cost of losing the easy visual appreciation of the histogram distributions.

User documentation of the ‘Superviolin’ Python package

The superviolin package can be used to create Violin SuperPlots. As well as a tutorial on how to use the package, this documentation provides an introduction to Python installation and package management to enable non-programmers to easily setup a Python workspace to use this software.

When using this software for your research, please cite our article.

1. Python installation for non-programmers

If you haven’t used Python before, you will be installing a Python distribution called Anaconda, which is widely used by scientific Python developers. A distribution is a bundle that contains the components necessary to run Python, as well as some domain-specific packages. For example, the Anaconda distribution is intended for use as a data science solution and contains packages for machine learning, scientific data analysis, and much more. It ships with over 200+ packages, including the package dependencies for Superviolin.

However, as this guide is aimed at non-programmers, we will be installing the Miniconda distribution, which has the same functionality as Anaconda, but saves on hard drive space and installation time by only installing the basic packages required for it to function. If there are other packages you want to use in the future, you can install them manually later on.

To install Miniconda:

1. Download the Python 3.x Miniconda installer for your OS at <https://docs.conda.io/en/latest/miniconda.html>.
Superviolin requires the installed Python version to be at least 3.6.2
2. Run the installer and follow the on-screen instructions, using the default settings you are prompted with.
3. Congratulations! You now have a functional Python environment on your computer!

You can now proceed to installing Superviolin. Python has a so-called package manager which is used to install, update, and remove packages. It ensures that all package

dependencies are maintained and will warn you if something would be broken by a new package installation.

Superviolin uses basic functions from Numpy, Scipy, and Pandas which aren't likely to be changed dramatically, so you don't need to worry about the version number. As long as you installed Python 3.6.2 or later, Superviolin should install just fine.

Installing Python packages requires you to enter text commands in the terminal (Mac, Linux) or in Anaconda Prompt (Windows). Spacing and upper/lower case must be maintained to ensure the commands run as expected. Most common programming errors occur due to these mistakes and they affect both experienced and beginners alike, though experienced programmers tend to get these errors less often.

To install Superviolin, open up the terminal or Anaconda Prompt and enter:

```
pip install superviolin
```

The package manager will list the dependencies required and install them for you. Once you see a message stating superviolin was successfully installed, you can proceed to the next section and try out the package with your data.

2. Superviolin command overview

The superviolin CLI has 3 commands:

demo

This command creates a Violin SuperPlot using dummy data that ships with the package. Run `superviolin demo` from your command prompt after installation to be sure the package is working. It should generate a figure similar to **Supplemental Figure S2** above. If you get an error, please contact Martin Kenny at mkenny5@tcd.ie to report your issue so that it can be corrected.

init

The `superviolin init` command generates an 'args.txt' file in the current directory, which will be used to generate a Violin SuperPlot based on the Excel or csv file of your data, located in the same folder.

plot

The `superviolin plot` command renders the Violin SuperPlot as a figure. This layout can be edited prior to saving. A paired or unpaired t-test is run if there are 2 conditions to be compared. A one-way ANOVA followed by Tukey's tests with Bonferroni corrections are run when there are 3 or more conditions. Statistical test results are output on the Anaconda command line interface, as well as saved in a 'posthoc_statistics.txt' file in the same folder as the data. There is no strict limit for the number of conditions that can be plotted/compared, apart from the size of the plot window which limits the navigation of the plot.

NOTE: To run any of these commands, enter it into your terminal or Anaconda Prompt.

3. Generating a Violin SuperPlot

Superviolin comes with a command-line interface (CLI) so you can generate SuperPlots using the terminal or Anaconda Prompt. You can also import the "Superviolin" class into your Python scripts and extend its functionality to create your own specific brand of Violin SuperPlot, but first, let's understand how it works. This section also contains a simple how-to for non-programmers.

1. Prepare your data as a *.csv or Excel file and place it into an empty folder. The data should be arranged in either (A) the tidy data format where you have a column for each variable/property and a row for each observation, or (B) 'untidy' data in an Excel workbook where each sheet is an experimental replicate containing columns for each condition. The tidy data format is preferred because it is more flexible. For tidy data, the *Condition* column specifies if an observation belongs to a different condition (control vs drug, different drug concentrations, wild type vs mutant, etc.), the *Value* column(s) specify the value(s) of the measured variable(s) (spreading area, circularity, intensity, etc.), and the *Replicate* column specifies which replicate the observation is from (donor ID, date, well, etc.).

Condition	Area	Ellipticity	Replicate
Control	25.2	1.434	Donor 1
Control	17.1	1.017	Donor 1
...
Drug	26.6	1.418	Donor 1

Drug	12.4	1.592	Donor 1
...
Control	28.9	1.481	Donor 2
Control	35.3	1.294	Donor 2
Control	39.5	1.374	Donor 2
...
Drug	9.8	1.767	Donor 2
Drug	18.9	1.660	Donor 2
Drug	32.2	1.376	Donor 2
...
Control	34.1	1.374	Donor 3
...
Drug	25.5	1.810	Donor 3
...

Example of a data file in tidy format (either csv or Excel file).

For 'untidy' data, column names have to be exactly the same across replicates to allow the software to combine the data for plotting; the data in each column doesn't have to be paired data, i.e. can have different number of entries. Column names should not contain special characters or sub/super-script.

names of conditions

data in column format

names of replicates
(same spreadsheet layout)

Example of a data file in an 'untidy' format (several Excel spreadsheets).

2. Open the terminal or Anaconda Prompt. Navigate to the folder containing the data file by typing & copy-pasting `cd drive:\full\path\to\your\data\folder` into the command line (replace the 'drive:\...' by your actual path).
3. Run `superviolin init` to generate the args.txt file in this folder.
4. Edit the args.txt file with the parameters specific to your data file. Optional arguments are detailed in section 6. The required parameters are:
 - i. **filename**, the name of your data file. Must be in CSV or Excel format.


- ii. **data_format**, whether the data follows the format of tidy data (table above) or 'untidy' data (figure above). NOTE: when using the untidy format, the package creates a tidy format dataframe from the Excel sheets, and the columns of this dataframe are named from the following 3 arguments.

For tidy data, additional inputs are required:


- iii. **condition**, the column in your dataset which specifies the experimental conditions.
 - iv. **value**, the variable to be plotted from your dataset.
 - v. **replicate**, the name of the replicate column in your dataset.
5. Save the args.txt file.
 6. Run `superviolin plot` to generate your Violin SuperPlot which will open in the "Figure 1" window. Please note that the Figure window needs to be closed to be able to access the command line / Anaconda prompt again.

Simply changing the filename in the args.txt file will allow you to apply those settings to other CSV or Excel files with the same structure, provided the new filename is in the same directory as the args.txt file.

4. Editing your Violin SuperPlot

Most of the editing of the Violin SuperPlot is done directly in the args.txt file. The basic Violin SuperPlot generated by the program can then be further adapted to edit the amount of whitespace around your plot. **Supplemental Figure S2** shows the Violin SuperPlot generated by the `superviolin demo` command. The whitespace around the plot is already optimized within the code, but the size of these spaces can be adjusted by clicking on the slider button  to open a menu of sliders corresponding to the whitespace to the top, bottom, left, and right of the plot.

5. Saving your Violin SuperPlot

Once satisfied with the plot, you have the option to save it either as a rendered image or a vector graphics image. Clicking the save icon  will open a dialog to choose the location for saving your figure and allow you to select the appropriate output format from the dropdown menu.

Rendered images: preferred are lossless formats (TIFF, PNG). The plot is saved at the resolution specified during creation of the plot (see section 6 below for optional arguments in the 'args.txt' file).

For maximal resolution and further editing in other programs, it is recommended to save the figure in SVG vector graphics format.

6. Optional arguments in args.txt used to customize the Violin SuperPlot

For example usages of optional arguments, see **Supplemental Figure S3**.

- i) **Xlabel** and **Ylabel** indicate the labels you wish to use for the x and y axes. These arguments can be left unchanged if you don't wish to have a label on either axis. Special characters and super/sub-script formatting can be specified using Python's string formatting: <https://pythonforundergradengineers.com/unicode-characters-in-python.html>
For example, the y-axis label "spreading area (μm^2)" in Supplemental Figure S2 is input into the args.txt file as "spreading area (μm^2)", where μ and the m^2 correspond to the Greek letter μ and the m^2 , respectively.
- ii) **Order**, the order of the experimental conditions to be displayed on the x axis. The values in this argument MUST be separated by a single comma and single space otherwise it will throw an error. There is no default order.
In the above example: 'Control, Drug'
- iii) **Middle_vals**, the central measure of each replicate. Displayed as circles over each replicate stripe. Default is mean. Accepted values are "mean", "median", or "robust". The "robust" mean ignores the upper and lower 2.5% of the data before taking the mean of the inner 95%.
- iv) **Centre_val**, the central tendency (mean or median) calculated from replicate means (or medians) and shown between the error bars on the plot. Default is mean.
- v) **Error_bars**, whether to use the standard error of the mean (SEM), the standard deviation (SD), or 95% confidence interval (CI) for plotting error bars. Default is SEM.

- vi) **Bw**, the bandwidth of the kernel density estimator, a decimal smoothing factor for stripe and violin outlines. The default value is “None”, which means an optimal value will be calculated using Scott’s Rule and the software will show this quantity in the Anaconda Prompt or Terminal window.
- vii) **Paired_data**, whether to run a paired samples t-test or a non-paired t-test for statistical comparison of replicates. Values are either “yes” or “no”. Only applicable if exactly two conditions are provided/present.
- viii) **Stats_on_plot**, choose whether to display t-test statistics on the plot. Only applicable for 2 experimental conditions. Either “yes” or “no”.
- ix) **Ylimits**, the minimum and maximum values to display on the y-axis. Should be in the form “low, high” e.g. “0.5, 1.5”. If this argument is to be ignored, leave the value as “None”.
- x) **Total_width**, the overall normalised width of each violin in decimal form. Should be between 0 and 1. Default is 0.8.
- xi) **Linewidth**, the width of lines (in points) used to plot summary statistics and the outline of each violin. Default is 1.
- xii) **Show_legend**, whether to show a legend or not. The legend links the names of each replicate to the corresponding colour-coded stripe in each Violin SuperPlot.
- xiii) **Cmap**, the colourmap specifying the colours used for the replicates. Set2 is the default colourmap, but users can specify a list of colours or a colourmap of their choosing from https://matplotlib.org/stable/gallery/color/named_colors.html or <https://matplotlib.org/stable/tutorials/colors/colormaps.html>
- xiv) **Dpi**, the dots per inch used for saving the plots. Plots are rendered at 300dpi for display purposes, but can be saved at much higher resolution, as Python cleverly manages an object for the figure which can be rendered and saved at different dpi values. The default value for saving is 600dpi, but can accommodate other values e.g. 300 or 1200.

Appendix A. Basic implementation of Violin SuperPlots in MATLAB

If you have questions regarding the use of these MATLAB functions, please contact the author ingmarschoen@rcsi.ie.

When using this software for your research, please cite our article.

Files

Two files are provided:

`superviolin.m` MATLAB function to create a basic Violin SuperPlot

`ExampleUsage.m` MATLAB script illustrating the usage of the `superviolin` function and its optional parameters.

The code and its calling function were tested only with MATLAB version R2020a on a Windows 10 computer. They require the 'Statistics and Machine Learning Toolbox' (Version: '11.7' was tested).

The code is not detailed here. Please refer to the in-line comments in the two files.

Implementation

What the `superviolin` function does: The raw data for different biological replicates is binned into histograms. A kernel density estimation is run on each of these binned data. The resulting smoothened histograms are stacked to end up with a symmetric outline. These transformed histograms are then depicted as filled polygons (patches) which make up the stripes of the compound violin. Replicate means (medians) are displayed as markers overlaid over the corresponding stripe of the compound violin. The mean and standard error of these replicate means are shown as a line with error bars. The function returns a plot object handle. This handle allows for customization of plot parameters even after generation of the figure.

Usage

The `superviolin` function is called with one required input argument, the data for a single condition, and a variable number of optional input arguments.

```
superviolin(condition1, varargin)
```

Data format

The data has to be provided as a 1-dimensional cell array (' $1 \times N$ cell'). Each cell represents one biological replicate. The number of replicates N will determine the number of stripes in the Violin SuperPlot.

Each replicate contains n individual measurements, which can be the number of cells or the number of technical replicates, depending on the experimental design. These data points have to be in the form of a data vector (an array of size $1 \times n$ or $n \times 1$). Data points can take any numerical format. Here an example:

```
condition1 =  
    1x4 cell array  
    {1x83 double} {1x73 double} {1x83 double} {1x75 double}
```

If your data has a different format, you need to convert it into a cell array. Please consult the Matlab documentation for the built-in functions `mat2cell`, `struct2cell`, and `table2cell` for help.

Optional input arguments

The appearance of the Violin SuperPlot can be adapted using optional input arguments provided as a comma-separated list of name-value pairs:

```
superviolin(condition1, Name, Value, Name, Value,...)
```

The choice and meaning of optional arguments are largely identical to the ones implemented in the Python package (see above). The following table gives an overview. Please note that the `Name` argument has to be provided in quotation marks.

Name	Value type	Description	Range (* = typical)	Default
'Parent'	axis handle	Handle to plot axis in current figure. If no input is provided, the current axis will be used.	[]	[]
'Xposition'	numeric	Centre position of the violin along the x axis. Useful to generate several Violin SuperPlots side by side by subsequent calls.	0., 1., 2., ... (*)	1

'Width'	numeric	Maximum width of the violin (along the x-axis).	0.5...2 (*)	0.8
'LUT'	lookup table	Lookup table for colour-coding of the biological replicates. Any built-in LUT from Matlab or user-defined LUTs can be used.	'parula', 'jet', 'hsv', ...	'lines'
'FaceAlpha'	numeric	Opacity of the face colour of violin stripes. 1=solid, 0=complete transparency.	0...1	0.7
'LineWidth'	numeric	Width of the dividing lines between stripes and the outlines of markers, given in points	0...1 (typ.)	0.5
'Centrals'	string	Which central tendency of replicates to use: mean, median, or robust mean. The robust mean removes outliers by taking the mean of the data in the [0.025 0.975] quantile range.	'mean', 'median', 'robustmean'	'mean'
'ErrorBars'	string	What to show as error bars: standard error of the mean (sem), standard deviation (std), or 95% confidence intervals (ci).	'sem', 'std', 'ci'	'sem'
'Bandwidth'	numeric	Bandwidth (relative to the data range) used for kernel density estimation. Smaller values give less smoothing. If no value is provided, the bandwidth is determined using Scott's rule which takes into account the number of data points: $bw = 0.5 n^{(-0.2)}$; the factor 0.5 has been added to avoid oversmoothing.	0.05...0.3	[]

For an example usage of these arguments, see the last section of the `ExampleUsage.m` file and **Supplemental Figure S4**.

Plotting two or more conditions

Plots for different conditions have to be generated by subsequent calls to the function using the respective data. They can be shown side-by-side in the same plot axis by varying the optional 'Xposition' argument (see above or `ExampleUsage.m`).

Statistical comparisons

Statistical tests are not implemented in these scripts; we assume that MATLAB users are familiar with the built-in functionalities of `multcompare` and alike.