The Journal of Open Source Software

Jupyter Scatter: Interactive Exploration of Large-Scale Datasets

Fritz Lekschas [1] and **Trevor Manz** [2]

1 Ozette Technologies, Seattle, WA, USA **2** Harvard Medical School, Boston, MA, USA

DOI: 10.21105/joss.07059

Software

- Review
- Repository
- Archive

Editor: Fabian-Robert Stöter

Reviewers:

- @xiaohk
- @paulbrodersen

Submitted: 10 June 2024
Published: 10 September 2024

Summary

Jupyter Scatter is a Python package for rendering scalable, interactive, and interlinked scatterplots to explore datasets in Jupyter Notebook/Lab, Colab, and VS Code (Figure 1). Thanks to its WebGL-based rendering engine (Lekschas, 2023), Jupyter Scatter can render and animate up to several million data points. The tool focuses on data-driven visual encodings and offers perceptually-effective point color and opacity settings by default. For interactive exploration, Jupyter Scatter features two-way zoom and point selections. Furthermore, it can compose multiple scatterplots and synchronize their views and selections, which is useful for comparing datasets. Finally, Jupyter Scatter's API integrates with Pandas DataFrames (McKinney, 2010) and Matplotlib (Hunter, 2007) and offers functional methods that group properties by type to ease accessibility and readability. Extensive documentation and how-tos can be found at https://jupyter-scatter.dev and the code is available at https://github.com/flekschas/jupyter-scatter.

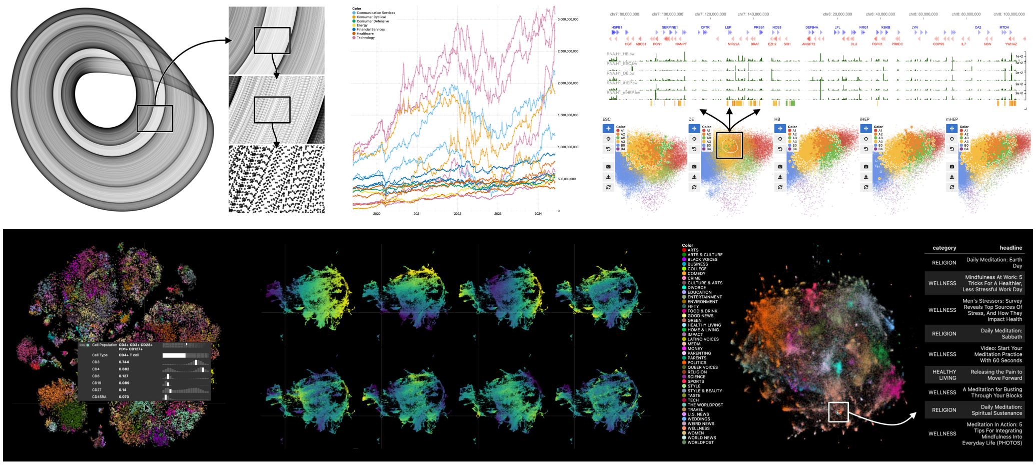

Figure 1: Examples of Jupyter Scatter. Top row left to right: A 10M point scatterplot of the Roessler Attractor. A connected scatterplot of the market capitalization over the last five years of the top ten S&P500 companies according to YCharts. Five linked embedding plots of epigenomic data (Dekker et al., 2023) that are connected to the HiGlass genome browser (Kerpedjiev et al., 2018). Bottom row left to right: A single-cell embedding plot of tumor data (Mair et al., 2022) that was clustered and annotated with FAUST (Greene et al., 2021, 2022). Several linked embedding plots of chromatin state datasets (Spracklin et al., 2023). An embedding plot of news headlines (Misra, 2022) that is linked to a widget for displaying selected articles.

Usage Scenario

Jupyter Scatter simplifies the visual exploration, analysis, and comparison of large-scale bivariate datasets. It renders up to twenty million points smoothly, supports fast point selections, integrates with Pandas DataFrame (McKinney, 2010), uses perceptually-effective default encodings, and offers a user-friendly API.

In the following, we demonstrate its usage for visualizing the GeoNames dataset (GeoNames, 2024), which contains data about 120k cities worldwide. For instance, to visualize cities by their longitude/latitude and color-code them by continent (Figure 2 Left), we create a `Scatter` instance as follows.

```python
import jscatter
import pandas as pd

geonames = pd.read_parquet('https://paper.jupyter-scatter.dev/geonames.pq')

scatter = jscatter.Scatter(
  data=geonames,
  x='Longitude',
  y='Latitude',
  color_by='Continent',
)
scatter.show()
```

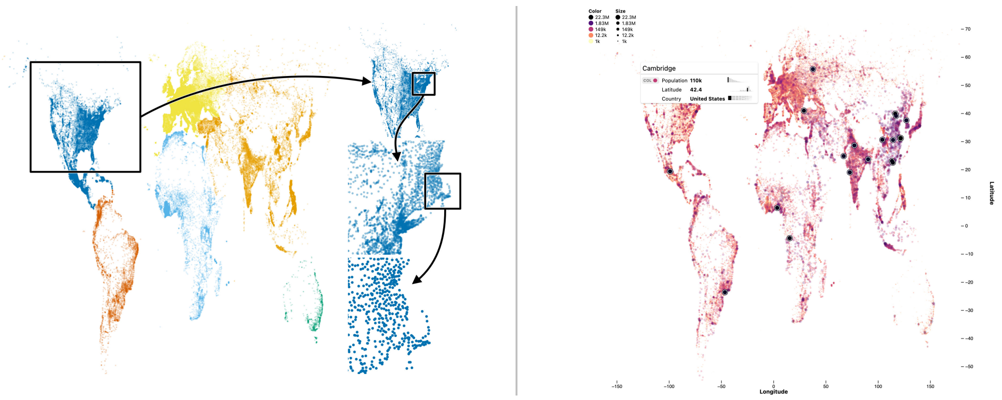

Figure 2: GeoNames Dataset of Cities Around the World.

Without specifying a color map, Jupyter Scatter uses the categorical colorblind-safe palette from Okabe & Ito (2002) for the `Continent` column, which has seven unique values. For columns with continuous data, it automatically selects Matplotlib's (Hunter, 2007) *Viridis* color palette. As shown in Figure 1 and Figure 2 Left, Jupyter Scatter dynamically adjusts the point opacity based on the point density within the field of view. This means points become more opaque when zooming into sparse areas and more transparent when zooming out into an area that contains many points. The dynamic opacity addresses over-plotting issues when zoomed out and visibility issues when zoomed in.

Jupyter Scatter offers many ways to customize the point color, size, and opacity encodings. To simplify configuration, it provides topic-specific methods for setting up the scatterplot, rather than requiring all properties to be set during the instantiation of `Scatter`. For instance, as shown in Figure 2 Right, the point opacity (`0.5`), size (asinh-normalized), and color (log-normalized population using Matplotlib's (Hunter, 2007) *Magma* color palette in reverse order) can be set using the following methods.

```python
from matplotlib.colors import AsinhNorm, LogNorm
scatter.opacity(0.5)
scatter.size(by='Population', map=(1, 8, 10), norm=AsinhNorm())
scatter.color(by='Population', map='magma', norm=LogNorm(), order='reverse')
```

To aid interpretation of individual points and point clusters, Jupyter Scatter includes legends, axis labels, and tooltips. These features are activated and customized via their respective methods.

```python
scatter.legend(True)
scatter.axes(True, labels=True)
scatter.tooltip(True, properties=['color', 'Latitude', 'Country'], preview='Name')
```

The tooltip can show a point's data distribution in context to the whole dataset and include a text, image, or audio-based media preview. For instance, the example (Figure 2 Right) shows the distribution of the visually encoded color property as well as the `Latitude` and `Country` columns. For numerical properties, the distribution is visualized as a bar chart, and for categorical properties the distribution is visualized as a treemap. As the media preview we're showing the city name.

Exploring a scatterplot often involves studying subsets of the points. To select points, one can either long press and lasso-select points interactively in the plot (Figure 3 Bottom Left) or query-select points (Figure 2 Right) as shown below. In this example, we select all cities with a population greater than ten million.

```python
scatter.selection(geonames.query('Population > 10_000_000').index)
```

The selected cities can be retrieved by calling `scatter.selection()` without any arguments. It returns the data record indices, which can then be used to get back the underlying data records.

```python
cities.iloc[scatter.selection()]
```

To automatically register changes to the point selection one can observe the `scatter.widget.selection` traitlet. The observability of the selection traitlet (and many other properties of `scatter.widget`) makes it easy to integrate Jupyter Scatter with other Jupyter Widgets.

For instance, Figure 3 (Left) shows a UMAP (McInnes et al., 2018) embedding of the Fashion MNIST dataset (Xiao et al., 2017) where points represent images and the point selection is linked to an image widget that loads the selected images.

```python
import ipywidgets
import jscatter

fashion_mnist = pd.read_parquet('https://paper.jupyter-scatter.dev/fashion-mnist-embeddi

images = ImagesWidget() # Custom widget for displaying Fashion MNIST images

scatter = jscatter.Scatter(
    data=fashion_mnist,
    x='umapX',
    y='umapY',
    color_by='class',
    background_color='black',
    axes=False,
)

ipywidgets.link((scatter.widget, 'selection'), (images, 'images'))

ipywidgets.AppLayout(center=scatter.show(), right_sidebar=images)
```

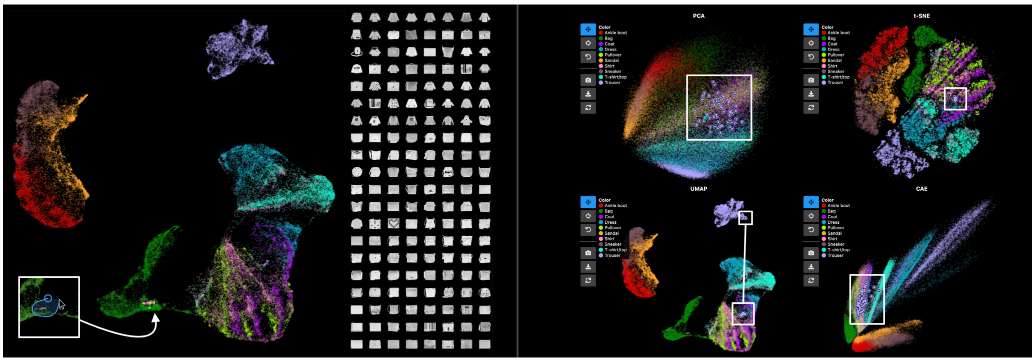

Figure 3: Fashion MNIST Embeddings. Left: Integration of Jupyter Scatter with an image widget through synchronized point selections. Right: Four scatterplots with synchronized point selection.

Comparing two or more related scatterplots can be useful in various scenarios. For example, with high-dimensional data, it might be necessary to compare different properties of the same data points. Another scenario involves embedding the high-dimensional dataset and comparing different embedding methods. For large-scale datasets, it might be useful to compare different subsets of the same dataset or entirely different datasets. Jupyter Scatter supports these comparisons with synchronized hover, view, and point selections via its `compose` method.

For instance, there are many ways to embed points into two dimensions, including linear and non-linear methods, and comparing point clusters between different embedding methods can be insightful. In the following, we compose a two-by-two grid of four embeddings of the Fashion MNIST dataset (Xiao et al., 2017) created with PCA (Pearson, 1901), UMAP (McInnes et al., 2018), t-SNE (Maaten & Hinton, 2008), and a convolutional autoencoder (Kingma & Welling, 2013). As illustrated in Figure 3 (Right), the point selection of the four scatterplots is synchronized.

```python
config = dict(
    data=fashion_mnist,
    color_by='class',
    legend=True,
    axes=False,
    zoom_on_selection=True,
)

pca = jscatter.Scatter(x='pcaX', y='pcaY', **config)
tsne = jscatter.Scatter(x='tsneX', y='tsneY', **config)
umap = jscatter.Scatter(x='umapX', y='umapY', **config)
cae = jscatter.Scatter(x='caeX', y='caeY', **config)

jscatter.compose(
    [(pca, "PCA"), (tsne, "t-SNE"), (umap, "UMAP"), (cae, "CAE")],
    sync_selection=True,
    sync_hover=True,
    rows=2,
)
```

Note, by setting `zoom_on_selection` to `True` and synchronizing selections, selecting points in one scatter will automatically select and zoom in on those points in all scatters.

Statement of Need

Jupyter Scatter is primarily a tool for data scientists to visually explore and compare bivariate datasets. Its ability for two-way point selections and synchronized plots, enable interactive exploration and comparison in ways that is not possible with existing widgets (e.g., multiple linked scatterplots) or requires considerable effort to set up (e.g., two-way communication of point selections).

Further, due to its usage of traitlets (IPython development team, 2024), Jupyter Scatter integrates easily with other widgets, which enables visualization researchers and practitioners to build domain-specific applications on top of Jupyter Scatter. For instance, the *Comparative Embedding Visualization* widget (Manz, Lekschas, et al., 2024) uses Jupyter Scatter to display four synchronized scatterplots for guided comparison of embedding visualizations. Andrés Colubri's research group is actively working on a new version of their *Single Cell Interactive Viewer* which will be based on Jupyter Scatter.

Implementation

Jupyter Scatter has two main components: a Python program running in the Jupyter kernel and a front-end program for interactive visualization. The Python program includes a widget and an API layer. The widget defines the view model for drawing scatterplots, while the API layer simplifies defining the view model state, integrating with Pandas DataFrames (McKinney, 2010) and Matplotlib (Hunter, 2007). The front-end program is built on top of regl-scatterplot (Lekschas, 2023), a high-performance rendering library based on WebGL, ensuring efficient GPU-accelerated rendering.

All components are integrated using anywidget (Manz, Abdennur, et al., 2024) to create a cross-platform Jupyter widget compatible with various environments, including Jupyter, JupyterLab, Google Colab, VS Code, and dashboarding frameworks like Shiny for Python, Solara, and Panel. The Python program uses anywidget and ipywidgets (Jupyter widgets community, 2015) to communicate with the front end, using binary data support to efficiently send in-memory data to the GPU, avoiding the overhead of JSON serialization. This approach enables the transfer of millions of data points from the Python kernel to the front end with minimal latency. Bidirectional communication ensures the visualization state is shared between the front-end and kernel, allowing updates to scatterplot properties and access to states like selections. Coordination is managed using anywidget APIs, enabling connections to other ipywidgets like sliders, dropdowns, and buttons for custom interactive data exploration widgets.

Related Work

There are many Python packages for rendering scatterplots in notebook-like environments. General-purpose visualization libraries like Matplotlib (Hunter, 2007), Bokeh (Bokeh development team, 2018), or Altair (VanderPlas et al., 2018) offer great customizability but do not scale to millions of points. They also don't offer bespoke features for exploring scatterplots and require manual configuration.

More bespoke dataset-centric plotting libraries like Seaborn (Waskom, 2021) or pyobsplot (Barnier, 2024) require less configuration and make it easier to create visually-pleasing scatterplots but they still fall short in terms of scalability.

Plotly combines great customizability with interactivity and can render scatterplots of up to a million points. However, drawing many more points is challenging and the library also focuses more on generality than dedicated features for scatterplot exploration and comparison. Plotly's WebGL rendering mode is also bound to the number of WebGL contexts your browser supports (typically between 8 to 16) meaning that it can't render more than 8 to 16 plots when using the WebGL render mode. Jupyter Scatter does not have this limitation as it uses a single WebGL renderer for all instantiated widgets, which is sufficient as static figures don't need constant re-rendering and one will ever only interact with a single or few plots at a time. Being able to render more than 8 to 16 plots can be essential in notebook environments as these are often used for exploratory data analysis.

Datashader (Anaconda developers and community contributors, 2024) specializes in the static rendering of large-scale datasets and offers unparalleled scalability that greatly exceeds that of Jupyter Scatter. One can also fine-tune how data is aggregated and rasterized. However, this comes at the cost of limited interactivity. While it's possible to interactively zoom into a rasterized image produced by Datashader, the image is just drawn at scale instead of being re-rendered at different fields of view. Re-rendering can be important though to better identify patterns in subsets of large scatterplots through optimized point size and opacity.

Although Jupyter Scatter is not tied to any specific application area and works with any bivariate data, one common use case is to plot 2D embeddings. In this context, Embedding Projector (Smilkov et al., 2016), WizMap (Wang et al., 2023), and DataMapPlot [datamapplot] are alternatives to Jupyter Scatter that run in Jupyter Notebook/Lab and can scale to millions of points. The Embedding Projector can visualize 2D and 3D scatter plots but is tightly coupled with TensorFlow's TensorBoard. WizMap and DataMapPlot work with any bivariate data and offer additional specialized features like displaying labels and cluster outlines/contours. However, unlike Jupyter Scatter, WizMap offers only a fixed visual encoding optimized for embeddings, meaning that point color, size, and opacity cannot be adjusted. Additionally, both WizMap and DataMapPlot output static HTML only, which means they do not integrate into the Jupyter Widget ecosystem. For example, while Jupyter Scatter does not have built-in search functionality, it can be easily implemented using the existing Jupyter Text Widget in combination with Jupyter Scatter's two-way point selections[1].

Finally, except for Plotly, none of the tools offer readily available interactive *two-way* point selection that exposes the selected points for reading and writing in both the Python and JavaScript kernels. This is a key feature of Jupyter Scatter to enable follow-up analysis of subsets of the data. Also, no other library offers direct support for synchronized exploration of multiple scatterplots for comparison.

Acknowledgements

We acknowledge and appreciate contributions from Pablo Garcia-Nieto, Sehi L'Yi, Kurt McKee, and Dan Rosén. We also thank Nezar Abdennur for his feedback on the initial API design.

References

Anaconda developers and community contributors. (2024). *Datashader: Accurately render even the largest data*. https://github.com/holoviz/datashader

Barnier, J. (2024). *Observable Plot in Jupyter notebooks and Quarto documents*. https://github.com/juba/pyobsplot

Bokeh development team. (2018). *Bokeh: Python library for interactive visualization*. https://bokeh.pydata.org/en/latest/

Dekker, J., Alber, F., Aufmkolk, S., Beliveau, B. J., Bruneau, B. G., Belmont, A. S., Bintu, L., Boettiger, A., Calandrelli, R., Disteche, C. M., & others. (2023). Spatial and temporal organization of the genome: Current state and future aims of the 4D nucleome project. *Molecular Cell*. https://doi.org/10.1016/j.molcel.2023.06.018

[1] https://github.com/flekschas/jupyter-scatter-tutorial/blob/main/notebooks/5-Search.ipynb

GeoNames. (2024). *GeoNames*. https://www.geonames.org

Greene, E., Finak, G., D'Amico, L. A., Bhardwaj, N., Church, C. D., Morishima, C., Ramchurren, N., Taube, J. M., Nghiem, P. T., Cheever, M. A., & others. (2021). New interpretable machine-learning method for single-cell data reveals correlates of clinical response to cancer immunotherapy. *Patterns*, *2*(12). https://doi.org/10.1016/j.patter.2021.100372

Greene, E., Finak, G., Lekschas, F., Smith, M., D'Amico, L. A., Bhardwaj, N., Church, C. D., Morishima, C., Ramchurren, N., Taube, J. M., Nghiem, P. T., Cheever, M. A., Fling, S. P., & Gottardo, R. (2022). *Data Transformations for Effective Visualization of Single-Cell Embeddings* (Version 1.0.0). https://doi.org/10.5281/zenodo.7522322

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

IPython development team. (2024). *Traitlets: A lightweight Traits like module*. https://github.com/ipython/traitlets

Jupyter widgets community. (2015). *Ipywidgets: Interactive widgets for the jupyter notebook*. https://github.com/jupyter-widgets/ipywidgets

Kerpedjiev, P., Abdennur, N., Lekschas, F., McCallum, C., Dinkla, K., Strobelt, H., Luber, J. M., Ouellette, S. B., Azhir, A., Kumar, N., & others. (2018). HiGlass: Web-based visual exploration and analysis of genome interaction maps. *Genome Biology*, *19*, 1–12. https://doi.org/10.1186/s13059-018-1486-1

Kingma, D. P., & Welling, M. (2013). *Auto-encoding variational Bayes*. https://doi.org/10.48550/ARXIV.1312.6114

Lekschas, F. (2023). Regl-Scatterplot: A Scalable Interactive JavaScript-based Scatter Plot Library. *Journal of Open Source Software*, *8*(84), 5275. https://doi.org/10.21105/joss.05275

Maaten, L. van der, & Hinton, G. (2008). Visualizing data using t-SNE. *Journal of Machine Learning Research*, *9*(86), 2579–2605. http://jmlr.org/papers/v9/vandermaaten08a.html

Mair, F., Erickson, J. R., Frutoso, M., Konecny, A. J., Greene, E., Voillet, V., Maurice, N. J., Rongvaux, A., Dixon, D., Barber, B., & others. (2022). Extricating human tumour immune alterations from tissue inflammation. *Nature*, *605*(7911), 728–735. https://doi.org/10.1038/s41586-022-04718-w

Manz, T., Abdennur, N., & Gehlenborg, N. (2024). *Anywidget: Reusable widgets for interactive analysis and visualization in computational notebooks*. https://doi.org/10.31219/osf.io/tw9sg

Manz, T., Lekschas, F., Greene, E., Finak, G., & Gehlenborg, N. (2024). *A general framework for comparing embedding visualizations across class-label hierarchies*. https://doi.org/10.31219/osf.io/puxnf

McInnes, L., Healy, J., & Melville, J. (2018). *UMAP: Uniform manifold approximation and projection for dimension reduction*. https://doi.org/10.48550/ARXIV.1802.03426

McKinney, Wes. (2010). Data Structures for Statistical Computing in Python. *Proceedings of the 9th Python in Science Conference*, 56–61. https://doi.org/10.25080/Majora-92bf1922-00a

Misra, R. (2022). *News category dataset*. https://doi.org/10.48550/arXiv.2209.11429

Okabe, M., & Ito, K. (2002). *How to make figures and presentations that are friendly to color blind people*. https://jfly.uni-koeln.de/color/

Pearson, K. (1901). On lines and planes of closest fit to systems of points is space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, *2*(11), 559–572. https://doi.org/10.1080/14786440109462720

Smilkov, D., Thorat, N., Nicholson, C., Reif, E., Viégas, F. B., & Wattenberg, M. (2016). *Embedding projector: Interactive visualization and interpretation of embeddings*. https://doi.org/10.48550/arXiv.1611.05469

Spracklin, G., Abdennur, N., Imakaev, M., Chowdhury, N., Pradhan, S., Mirny, L. A., & Dekker, J. (2023). Diverse silent chromatin states modulate genome compartmentalization and loop extrusion barriers. *Nature Structural & Molecular Biology*, *30*(1), 38–51. https://doi.org/10.1038/s41594-022-00892-7

VanderPlas, J., Granger, B., Heer, J., Moritz, D., Wongsuphasawat, K., Satyanarayan, A., Lees, E., Timofeev, I., Welsh, B., & Sievert, S. (2018). Altair: Interactive statistical visualizations for Python. *Journal of Open Source Software*, *3*(32), 1057. https://doi.org/10.21105/joss.01057

Wang, Z. J., Hohman, F., & Chau, D. H. (2023). WizMap: Scalable Interactive Visualization for Exploring Large Machine Learning Embeddings. *arXiv 2306.09328*. https://doi.org/10.48550/arXiv.2306.09328

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

Xiao, H., Rasul, K., & Vollgraf, R. (2017). *Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms*. https://doi.org/10.48550/arXiv.1708.07747