The Journal of Open Source Software

ArviZ: a modular and flexible library for exploratory analysis of Bayesian models

Osvaldo A Martin[1*¶], Oriol Abril-Pla[2*¶], Jordan Deklerk[3], Seth D. Axen[4], Colin Carroll[2], Ari Hartikainen[2], and Aki Vehtari[1,5]

1 Aalto University, Espoo, Finland **2** arviz-devs **3** DICK's Sporting Goods, Coraopolis, Pennsylvania **4** University of Tübingen **5** ELLIS Institute Finland ¶ Corresponding author * These authors contributed equally.

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor: Matt Graham

Reviewers:

- @smutch
- @vankesteren

Submitted: 13 January 2026
Published: unpublished

Summary

When working with Bayesian models, a range of related tasks must be addressed beyond inference itself. These include diagnosing the quality of Markov chain Monte Carlo (MCMC) samples, model criticism, model comparison, etc. We collectively refer to these activities as exploratory analysis of Bayesian models.

In this work, we present a redesigned version of ArviZ, a Python package for exploratory analysis of Bayesian models (EABM). The redesign emphasizes greater user control and modularity. This redesign delivers a more flexible and efficient toolkit for exploratory analysis of Bayesian models. With its renewed focus on modularity and usability, ArviZ is well-positioned to remain an essential tool for Bayesian modelers in both research and applied settings.

Statement of need

Probabilistic programming has emerged as a powerful paradigm for statistical modeling, accompanied by a growing ecosystem of tools for model specification and inference. Effective modeling requires robust support for uncertainty visualization, sampling diagnostics, model comparison, and model checking (Gelman et al., 2020; Guo et al., 2024; Martin, 2024). ArviZ addresses this gap by providing a unified, backend-agnostic library to perform these tasks. The original ArviZ paper (Kumar et al., 2019) described the landscape of probabilistic programming tools at the time and the need for a unified, backend-agnostic library for exploratory analysis — a need that has only grown as the ecosystem has expanded.

The methods implemented in ArviZ are grounded in well-established statistical principles and provide robust, interpretable diagnostics and visualizations (Dimitriadis et al., 2021; Gelman et al., 2019; Kallioinen et al., 2023; Paananen et al., 2021; Padilla et al., 2021; Säilynoja et al., 2022, 2025; Vehtari et al., 2017, 2021). Modern Bayesian practice is a rapidly advancing field in which new methodological developments continually extend the range and complexity of models that can be fit in practice. For instance, the methods to compute key ArviZ features such as `ess`, `rhat`, `loo` or `compare` have been improved between 2019 and now, and new implementations needed significant development effort to adapt to because it wasn't possible to change a part of ArviZ without also adapting everything that interacted with it. The redesign addresses these challenges by modularizing the codebase, allowing individual components to be updated or replaced without affecting the entire system. This modularity not only facilitates maintenance and updates but also encourages community contributions, as developers can focus on specific components without needing to understand the entire codebase.

State of the field

In the Python Bayesian ecosystem, ArviZ occupies a niche comparable to tools in the R/Stan community such as posterior (Gelman et al., 2013; Vehtari et al., 2021), loo (*Loo*, 2025; Vehtari et al., 2017), bayesplot (Gabry et al., 2019; Gabry & Mahr, 2025), priorsense (Kallioinen et al., 2023), and ggdist (Kay, 2024) sharing similar goals while reflecting different language ecosystems and workflows.

Research impact statement

ArviZ (Kumar et al., 2019) is a Python package for exploratory analysis of Bayesian models that has been widely used in academia and industry since its introduction in 2019, with over 700 citations and 75 million downloads. Its goal is to integrate seamlessly with established probabilistic programming languages and statistical interfaces, such as PyMC (Abril-Pla et al., 2023), Stan (via the cmdstanpy interface) (Carpenter et al., 2017), Pyro, NumPyro (Bingham et al., 2019; Phan et al., 2019), emcee (Foreman-Mackey et al., 2019), and Bambi (Capretto et al., 2022), among others.

The maturity of ArviZ has also led to other initiatives, including ArviZ.jl (Axen & Widmann, 2025) for Julia, PreliZ (Icazatti et al., 2023) for prior elicitation and the development of educational resources (Martin et al., 2025).

Software design

The previous ArviZ design divided the package into three submodules, which are now available as three independent installable packages. This redesign emphasizes greater user control and modularity. The new architecture enables users to customize the installation and use of specific components. Key design changes include:

General functionality, data processing, and data input/output (I/O) have been streamlined and enhanced for greater versatility. Previously, ArviZ used the custom `InferenceData` class to organize and store the high-dimensional outputs of Bayesian inference in a structured, labeled format, enabling efficient analysis, metadata persistence, and serialization. These have been replaced with the `DataTree` class from xarray (Hoyer & Hamman, 2017), which, like the original `InferenceData`, supports grouping but is more flexible, enabling richer nesting and automatic support for all xarray I/O formats. Additionally, converters allow more flexibility in dimensionality, naming, and indexing of their generated outputs.

Statistical functions are now accessible through two distinct interfaces:

- A low-level array interface with only `numpy` (Harris et al., 2020) and `scipy` (Virtanen et al., 2020) as dependencies, intended for advanced users and developers of third-party libraries.
- A higher-level xarray interface designed for end users, which simplifies usage by automating common tasks and handling metadata.

Plotting functions have also been redesigned to support modularity at multiple levels:

- At a high level, ArviZ offers a collection of "batteries-included" plots. These are built-in plotting functions providing sensible defaults for common tasks like MCMC sampling diagnostics, predictive checks, and model comparison.
- At an intermediate level, the application programming interface enables easier customization of batteries-included plots and simplifies the creation of new plots. This is achieved through the `PlotCollection` class, which enables developers and advanced users to focus solely on the plotting logic, delegating any faceting or aesthetic mappings to `PlotCollection`.

- At a lower level, we have improved the separation between computational and plotting logic, reducing code duplication and enhancing modular design. These changes also facilitate support for multiple plotting backends, improving extensibility and maintainability. Currently, ArviZ supports three plotting backends: matplotlib (Hunter, 2007), Bokeh (Bokeh Development Team, 2018), and plotly (Plotly Technologies Inc., 2015).

Thanks to this new design, the cost of adding "batteries-included" plots has reduced in more than half even though ArviZ now supports one extra backend. Consequently, redesigned ArviZ already has 37 "batteries-included", 10 more than the 0.x versions.

Examples

For the first example, we use the low-level array interface to compute the effective sample sizes for some fake data. We construct an array resembling data from MCMC sampling with 4 chains and 1000 draws for two posterior variables. When using the array interface we need to specify which axes represent the chains and which the draws.

```
import numpy as np
from arviz_stats.base import array_stats

rng = np.random.default_rng()
samples = rng.normal(size=(4, 1000, 2))  # (chain, draw, variable)
array_stats.ess(samples, chain_axis=0, draw_axis=1)
```

The array interface is lightweight and intended for advanced users and library developers. For most users, we instead recommend the xarray interface, as it is more user-friendly and automates many tasks. When converting the NumPy array to a `DataTree`, ArviZ assigns `chain` and `draw` as named dimensions based on the assumed dimension order, so this information is already encoded in the resulting object and does not need to be specified explicitly when calling other functions.

```
import arviz as az
dt_samples = az.convert_to_datatree(samples)
az.ess(dt_samples)
```

The only required argument for battery-included plots, like `plot_dist`, is the input data, typically a `DataTree` (`dt`). In this example we also apply optional customizations.

```
az.style.use('arviz-variat')
dt = az.load_arviz_data("centered_eight")
pc = az.plot_dist(
    dt,
    kind="dot",
    visuals={"dist":{"marker": "C6"},
             "point_estimate_text":False},
    aes={"color": ["school"]}
);
pc.add_legend("school", loc="outside right upper")
```

The Journal of Open Source Software

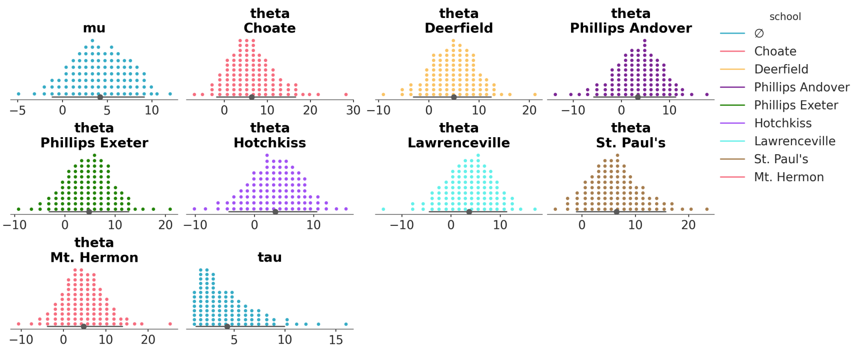

Figure 1: plot_dist with color mapped to school dimension.

To create Figure 1 we change the default kind argument in `plot_dist` from “kde” to “dot” to produce quantile dot plots (Kay et al., 2016), and map the school dimension to color so that each school is shown in a different hue. Variables that do not have a school dimension (such as mu and tau) are automatically assigned a neutral color. We also disable the point-estimate text and set a custom marker style for the dots, and finally add a legend for the school.

For more examples and a more comprehensive overview, see the ArviZ documentation and the EABM guide (Martin et al., 2025). These resources include a wide range of examples designed for all types of users, from casual users to advanced analysts and developers looking to use ArviZ in their projects or libraries.

AI usage disclosure

Generative AI tools were used during software development and documentation in a limited capacity, primarily to assist with rewording and minor code suggestions. All AI-assisted contributions were reviewed and edited by the authors. Core design decisions, feature development, and scientific or technical judgment were carried out by the authors, and all code and claims were tested and manually verified to ensure correctness.

Acknowledgements

We thank our fiscal sponsor, NumFOCUS, a nonprofit 501(c)(3) public charity, for their operational and financial support. We also thank all the contributors to `arviz`, `arviz-base`, `arviz-stats`, and `arviz-plots` repositories, including code contributors, documentation writers, issue reporters, and users who have provided feedback and suggestions.

This research was supported by:

- The Research Council of Finland Flagship Program “Finnish Center for Artificial Intelligence” (FCAI)
- Research Council of Finland grant 340721
- Essential Open Source Software Round 4 grant by the Chan Zuckerberg Initiative (CZI)
- Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany’s Excellence Strategy – EXC number 2064/1 – Project number 390727645

References

Abril-Pla, O., Andreani, V., Carroll, C., Dong, L., Fonnesbeck, C. J., Kochurov, M., Kumar, R., Lao, J., Luhmann, C. C., Martin, O. A., Osthege, M., Vieira, R., Wiecki, T., & Zinkov, R. (2023). PyMC: A modern, and comprehensive probabilistic programming framework in Python. *PeerJ Computer Science*, *9*, e1516. https://doi.org/10.7717/peerj-cs.1516

Axen, S. D., & Widmann, D. (2025). *arviz-devs/ArviZ.jl: v0.14.0* (Version v0.14.0). Zenodo. https://doi.org/10.5281/zenodo.17194186

Bingham, E., Chen, J. P., Jankowiak, M., Obermeyer, F., Pradhan, N., Karaletsos, T., Singh, R., Szerlip, P. A., Horsfall, P., & Goodman, N. D. (2019). Pyro: Deep universal probabilistic programming. *Journal of Machine Learning Research*, *20*, 28:1–28:6. http://jmlr.org/papers/v20/18-403.html

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

Capretto, T., Piho, C., Kumar, R., Westfall, J., Yarkoni, T., & Martin, O. A. (2022). Bambi: A simple interface for fitting Bayesian linear models in Python. *Journal of Statistical Software*, *103*(15), 1–29.

Carpenter, B., Gelman, A., Hoffman, M. D., Lee, D., Goodrich, B., Betancourt, M., Brubaker, M., Guo, J., Li, P., & Riddell, A. (2017). Stan: A probabilistic programming language. *Journal of Statistical Software*, *76*(1), 1–32. https://doi.org/10.18637/jss.v076.i01

Dimitriadis, T., Gneiting, T., & Jordan, A. I. (2021). Stable reliability diagrams for probabilistic classifiers. *Proceedings of the National Academy of Sciences*, *118*(8), e2016191118. https://doi.org/10.1073/pnas.2016191118

Foreman-Mackey, D., Farr, W. M., Sinha, M., Archibald, A. M., Hogg, D. W., Sanders, J. S., Zuntz, J., Williams, P. K. g., Nelson, A. R. j., Val-Borro, M. de, Erhardt, T., Pashchenko, I., & Pla, O. A. (2019). emcee v3: A Python ensemble sampling toolkit for affine-invariant MCMC. *Journal of Open Source Software*, *4*(43), 1864. https://doi.org/10.21105/joss.01864

Gabry, J., & Mahr, T. (2025). *Bayesplot: Plotting for Bayesian models*. https://doi.org/10.32614/cran.package.bayesplot

Gabry, J., Simpson, D., Vehtari, A., Betancourt, M., & Gelman, A. (2019). Visualization in Bayesian workflow. *Journal of the Royal Statistical Society Series A*, *182*, 389–402. https://doi.org/10.1111/rssa.12378

Gelman, A., Carlin, J. B., Stern, H. S., Dunson, D. B., Vehtari, A., & Rubin, D. B. (2013). *Bayesian Data Analysis, Third Edition* (3 edition). Chapman; Hall/CRC. ISBN: 978-1-4398-4095-5

Gelman, A., Goodrich, B., Gabry, J., & Vehtari, A. (2019). R-squared for Bayesian regression models. *The American Statistician*, *73*(3), 307–309. https://doi.org/10.1080/00031305.2018.1549100

Gelman, A., Vehtari, A., Simpson, D., Margossian, C. C., Carpenter, B., Yao, Y., Kennedy, L., Gabry, J., Bürkner, P.-C., & Modrák, M. (2020). *Bayesian workflow*. https://doi.org/10.48550/arXiv.2011.01808

Guo, Z., Kale, A., Kay, M., & Hullman, J. (2024). *VMC: A grammar for visualizing statistical model checks*. https://doi.org/10.48550/arXiv.2408.16702

Harris, C. R., Millman, K. J., Walt, S. J. van der, Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., Kerkwijk, M. H. van, Brett, M., Haldane, A., Río, J. F. del, Wiebe, M., Peterson, P., … Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2

Hoyer, S., & Hamman, J. (2017). xarray: N-D labeled arrays and datasets in Python. *Journal of Open Research Software*, *5*(1). https://doi.org/10.5334/jors.148

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

Icazatti, A., Abril-Pla, O., Klami, A., & Martin, O. A. (2023). PreliZ: A tool-box for prior elicitation. *Journal of Open Source Software*, *8*(89), 5499. https://doi.org/10.21105/joss.05499

Kallioinen, N., Paananen, T., Bürkner, P. C., & Vehtari, A. (2023). Detecting and diagnosing prior and likelihood sensitivity with power-scaling. *Statistics and Computing*, *34*. https://doi.org/10.1007/s11222-023-10366-5

Kay, M. (2024). ggdist: Visualizations of distributions and uncertainty in the grammar of graphics. *IEEE Transactions on Visualization and Computer Graphics*, *30*(1), 414–424. https://doi.org/10.1109/TVCG.2023.3327195

Kay, M., Kola, T., Hullman, J. R., & Munson, S. A. (2016). When (ish) is my bus? User-centered visualizations of uncertainty in everyday, mobile predictive systems. *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, 5092–5103. https://doi.org/10.1145/2858036.2858558

Kumar, R., Carroll, C., Hartikainen, A., & Martin, O. (2019). ArviZ a unified library for exploratory analysis of Bayesian models in Python. *Journal of Open Source Software*, *4*(33), 1143. https://doi.org/10.21105/joss.01143

Loo: Efficient leave-one-out cross-validation and WAIC for bayesian models. (2025). https://doi.org/10.32614/cran.package.loo

Martin, O. A. (2024). *Bayesian Analysis with Python: A Practical Guide to probabilistic modeling, 3rd Edition*. Packt Publishing. ISBN: 978-1-80512-716-1

Martin, O. A., Abril-Pla, O., & Deklerk, J. (2025). *Exploratory analysis of Bayesian models* (Version v0.3.0) [Computer software]. Zenodo. https://doi.org/10.5281/zenodo.15127548

Paananen, T., Piironen, J., Bürkner, P. C., & Vehtari, A. (2021). Implicitly adaptive importance sampling. *Statistics and Computing*, *31*(16). https://doi.org/10.1007/s11222-020-09982-2

Padilla, L., Kay, M., & Hullman, J. (2021). Uncertainty visualization. *Wiley StatsRef: Statistics Reference Online*. https://doi.org/10.1002/9781118445112.stat08296

Phan, D., Pradhan, N., & Jankowiak, M. (2019). Composable effects for flexible and accelerated probabilistic programming in NumPyro. *arXiv Preprint arXiv:1912.11554*.

Plotly Technologies Inc. (2015). *Collaborative data science*. Plotly Technologies Inc. https://plot.ly

Säilynoja, T., Bürkner, P. C., & Vehtari, A. (2022). Graphical test for discrete uniformity and its applications in goodness-of-fit evaluation and multiple sample comparison. *Statistics and Computing*, *32*, 1573–1375. https://doi.org/10.1007/s11222-022-10090-6

Säilynoja, T., Johnson, A. R., Martin, O. A., & Vehtari, A. (2025). Recommendations for visual predictive checks in Bayesian workflow. *arXiv:2503.01509*. https://doi.org/10.48550/arXiv.2503.01509

Vehtari, A., Gelman, A., & Gabry, J. (2017). Practical Bayesian model evaluation using leave-one-out cross-validation and WAIC. *Statistics and Computing*, *27*, 1413–1432. https://doi.org/10.1007/s11222-016-9696-4

Vehtari, A., Gelman, A., Simpson, D., Carpenter, B., & Bürkner, P. C. (2021). Rank-normalization, folding, and localization: An improved $\widehat{R}$ for assessing convergence of MCMC. *Bayes Anal*, *16*, 667–718. https://doi.org/10.1214/20-BA1221

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., … SciPy 1.0 Contributors. (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, *17*, 261–272. https://doi.org/10.1038/s41592-019-0686-2