




k-diagram: Rethinking Forecasting Uncertainty via Polar-based Visualization

Kouao Laurent Kouadio ^{1,2}

¹ School of Geosciences and Info-Physics, Central South University, Changsha, Hunan, 410083, China ² Hunan Key Laboratory of Nonferrous Resources and Geological Hazards Exploration, Changsha, Hunan, 410083, China

DOI: [10.21105/joss.08661](https://doi.org/10.21105/joss.08661)

Software

- [Review](#) 
- [Repository](#) 
- [Archive](#) 

Editor: Julia Romanowska 

Reviewers:

- [@ChinniAbbur](#)
- [@cbueth](#)

Submitted: 09 May 2025

Published: 12 December 2025

License

Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License ([CC BY 4.0](https://creativecommons.org/licenses/by/4.0/)).

Summary

k-diagram is an open-source Python package designed for the in-depth interpretation and diagnosis of probabilistic forecasts. Moving beyond traditional aggregate metrics, the package provides a suite of novel visualization tools, primarily based on polar coordinate systems, to dissect the complex spatiotemporal structure of forecast uncertainty and errors. By mapping statistical properties such as sharpness, reliability, and temporal stability onto intuitive geometric representations, k-diagram enables researchers and practitioners to identify model biases, understand forecast degradation over time, and communicate uncertainty characteristics more effectively. The package is designed to be a practical extension to the standard forecasting workflow, providing the visual evidence needed for more robust model evaluation and trustworthy, context-aware decision-making.

Statement of Need

The evaluation of probabilistic forecasts has a rigorous theoretical foundation based on concepts of calibration and sharpness, which are jointly assessed using proper scoring rules ([Gneiting et al., 2007](#)). However, these scores are typically aggregated into a single value, which can obscure vital information about a model's performance in heterogeneous spatiotemporal settings. This is a critical limitation, as modern machine learning models, such as Temporal Fusion Transformers ([Lim et al., 2021](#)), are increasingly applied to complex, high-dimensional problems where understanding the local and temporal behavior of uncertainty is paramount. A growing body of applied research highlights this challenge in fields as diverse as predictive policing ([Rummens & Hardyns, 2021](#)), energy forecasting ([Y. Liu et al., 2021](#)), chaotic market dynamics ([Caulk et al., 2022](#)), climatology ([Baillie & Chung, 2002](#)) and geohazard ([J. Liu et al., 2024](#)).

While an ecosystem of open-source forecasting tools exists, many focus on specific modeling or post-processing tasks, such as forecast reconciliation ([Biswas et al., 2025](#)), or provide verification suites for specific domains like climate science ([Brady & Spring, 2021](#)). General-purpose libraries like matplotlib ([Hunter, 2007](#)) and seaborn ([Waskom, 2021](#)) provide the building blocks but lack dedicated functions for the specialized diagnostics needed for high-dimensional uncertainty. Even established visualizations like fan charts, which have seen recent innovations ([Sokol, 2025](#)), are primarily designed for single time series and do not scale well to problems involving thousands of simultaneous forecasts ([Hong et al., 2025](#)). This creates a critical gap between the generation of complex probabilistic forecasts and the ability to interpret them effectively.

k-diagram addresses this gap by providing a scalable and intuitive toolkit designed specifically for the visual diagnosis of spatiotemporal probabilistic forecasts ([Kouadio, 2025](#); [J. Liu et al.,](#)

2024). Furthermore, in modern deployments, a single run may yield thousands to millions of (location, horizon) forecasts; k-diagram is designed to compress these large spatiotemporal distributions into stable, comparable views that preserve geography and lead-time structure. The package's novelty lies in its use of polar coordinates to map different dimensions of forecast performance, such as uncertainty magnitude, reliability, and temporal stability, onto angle and radius. This approach provides compact overviews that reveal patterns obscured in traditional Cartesian plots. By providing clear visual answers to key diagnostic questions (e.g., "Where is a forecast least certain?", "How is its uncertainty evolving?"), k-diagram serves as an essential tool for any researcher or practitioner seeking to move beyond aggregate metrics and gain a deeper, more actionable understanding of their forecasting models.

Functionality

k-diagram is implemented in Python (Python Software Foundation, 2008; Van Rossum & Drake, 2009), leveraging core scientific libraries including numpy (Harris et al., 2020), pandas (McKinney, 2010; Pandas development team, 2020), matplotlib (Hunter, 2007), scipy (Lu et al., 2020; Virtanen et al., 2020), and scikit-learn (Pedregosa et al., 2011). The core functionality is organized around a cohesive API designed to diagnose key aspects of forecast quality through four primary pillars of analysis (see Figure 1).

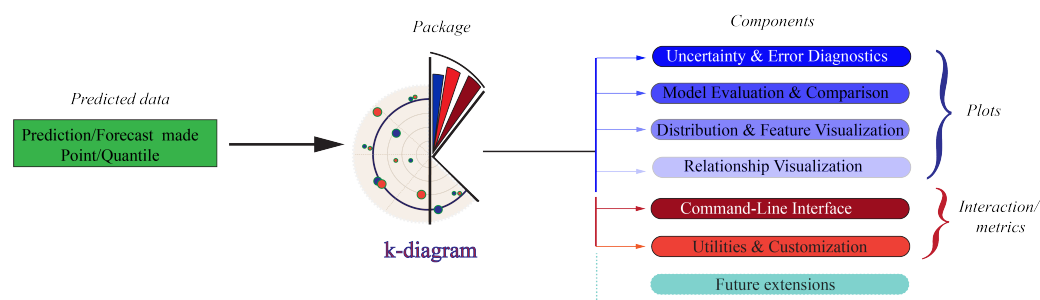


Figure 1: Structure of the k-diagram, illustrating the identification of global components.

The first and primary contribution of k-diagram is a suite of novel polar visualizations for **Uncertainty and Error Diagnostics** (Kouadio, 2025). The `kdiagram.plot.uncertainty`, `kdiagram.plot.errors`, and `kdiagram.plot.probabilistic` modules provide a comprehensive toolkit for dissecting forecast performance. This includes functions to evaluate prediction **interval performance** by visualizing coverage, the magnitude of anomalous failures, and the temporal consistency of interval widths. Furthermore, the package introduces innovative methods for visualizing **error distributions**, such as polar error bands to separate systemic bias from random error, and polar violins, a novel adaptation of the traditional violin plot (Hintze & Nelson, 1998), to compare the full error profiles of multiple models. For a complete assessment of the predictive distribution, a dedicated suite of **probabilistic diagnostics** offers Polar Probability Integral Transform Histograms for calibration, Sharpness Diagrams for precision, and Continuous Ranked Probability Score plots for overall skill, based on the foundational concepts of probabilistic forecasting (Gneiting et al., 2007).

Building on this, the second pillar focuses on established methods for **Model Evaluation and Comparison**. The package implements well-regarded evaluation techniques, adapting them for enhanced clarity. Functions in `kdiagram.plot.evaluation` generate **Taylor Diagrams**, based on the original formulation by (Taylor, 2001), to holistically compare models on correlation, standard deviation, and Root-Mean-Square Deviation. Complementing this, the `kdiagram.plot.comparison` module provides standard **Reliability Diagrams** for assessing probability calibration, a cornerstone of forecast verification (Jolliffe & Stephenson, 2012).

The third pillar of functionality addresses **Distribution and Feature Visualization**. `k-diagram` provides utilities for plotting 1D distributions and Kernel Density Estimates (KDEs) (Silverman, 1986) in both Cartesian and innovative polar ring formats. To facilitate model interpretability, the `kdiagram.plot.feature_based` module offers radar charts to visualize and compare feature importance profiles, creating unique model “fingerprints”.

Finally, the package provides tools for **Relationship Visualization**, where polar plots in `kdiagram.plot.relationship` offer a novel perspective on the correlation between true and predicted values by mapping them to angle and radius. All functionalities are supported by a set of helper utilities and a command-line interface (CLI) for generating key plots directly from data files.

Engineering - API and Extensibility

`k-diagram` is designed as a small stack of composable layers rather than a monolithic script. This architecture is directly reflected in its public API, which is data-first, predictable, and geared toward integration with the scientific Python ecosystem.

The primary API entry points are the `plot_*` functions. These functions are designed to be thin, predictable wrappers that accept data in common formats—either a tidy pandas `DataFrame` with explicit column selectors (e.g., `q_cols=('q10', 'q50', 'q90')`, `y_true='actual'`) or plain numpy arrays. In either case, inputs are validated early to provide clear error messages. A core feature of the API is its explicit “polar grammar.” Users have direct control over the geometry of their plots through parameters like `acov` (angular coverage), `zero_at` (orientation), and `clockwise` (direction). For periodic data, this grammar extends to human-readable labels via `theta_ticks` and `theta_ticklabels`, allowing axes to show “Monday” instead of “0.0 radians.” These controls are paired with shared display options like `show_grid` to ensure a consistent look and feel.

Crucially, every plotting function in `k-diagram` returns a standard Matplotlib Axes object. This is the package’s main contract: it does not hide the figure or return a custom object. This design choice ensures that users can immediately apply their existing Matplotlib knowledge to annotate, combine subplots, or save figures in a standard, compositional workflow. The proposed API is supported by an internal utility layer that handles the heavy lifting. The layer manages column detection, quantile pairing, angular mapping (`map_theta_to_span`), and color handling. This separation keeps the user-facing plot logic (the “what to draw”) independent from the rendering and validation logic (the “how to draw”). Cross-cutting concerns, like ensuring consistent polar grids (`setup_polar_axes`) or managing compatibility shims (`kdiagram.compat`), are centralized.

Furthermore, the layered design yields significant benefits in performance and testability. Data transformations are vectorized using NumPy and pandas, and no hidden global state is used. Each function depends only on its arguments, making its behavior pure and easy to test. Transforms and validators are unit-tested for correctness, while the rendering pipeline is exercised with headless smoke tests that assert on labels and returned object types rather than fragile pixel comparisons. The CLI simply wraps this same public API, guaranteeing identical figures from both scripts and interactive sessions.

Finally, the composable structure makes the package straightforward to extend. Adding a new diagnostic follows a consistent pattern: transform the input data, lay out the coordinate system using the shared helpers, and render with standard Matplotlib primitives. Because the shared helpers manage the complex parts of polar geometry and styling, new plot functions remain small and readable. This flexibility even allows a single data transform to back both a polar and a Cartesian renderer, which is exposed in some functions via the `kind="polar"|"cartesian"` toggle. The full [development page](#) describes these conventions in detail, allowing external contributors to add new diagnostics without guessing at internal idioms.

Illustrative Diagnostics and Application

The practical utility of k-diagram is best demonstrated through its key diagnostic plots, which are grounded in clear statistical concepts and reveal model behaviors often hidden by aggregate metrics (see Figure 2).

For instance, the **Coverage Evaluation** plot in Figure 2a provides a point-wise diagnostic of interval performance. For each observation y_i and its corresponding prediction interval $[L_i, U_i]$, a binary coverage value c_i is determined as:

$$c_i = 1(L_i \leq y_i \leq U_i)$$

where $1(\cdot)$ is the indicator function. The plot visualizes each c_i , allowing for a granular analysis of where interval failures occur. The average of these outcomes provides the overall empirical coverage, which at 81.1% is close to the nominal 80% for the Q10-Q90 interval shown.

The **Model Error Distributions** diagram (Figure 2b) offers a comparative view by visualizing the full error distribution for multiple models. It uses polar violins, where the shape of each violin is determined by the KDE of the model's errors (Silverman, 1986). The resulting diagram clearly distinguishes a "Good Model" (unbiased and narrow) from a "Biased Model" (shifted from the zero-error line) and an "Inconsistent Model" (wide and dispersed), revealing performance trade-offs that a single error score would miss.

Finally, the **Forecast Horizon Drift** diagram (Figure 2c) visualizes how uncertainty evolves over time. For each forecast horizon j , it calculates the mean interval width, \bar{w}_j , across all N spatial locations as:

$$\bar{w}_j = \frac{1}{N} \sum_{i=1}^N (U_{i,j} - L_{i,j})$$

The increasing height of the bars from 2023 to 2026 provides an immediate and intuitive confirmation that the model's uncertainty grows as it forecasts further into the future.

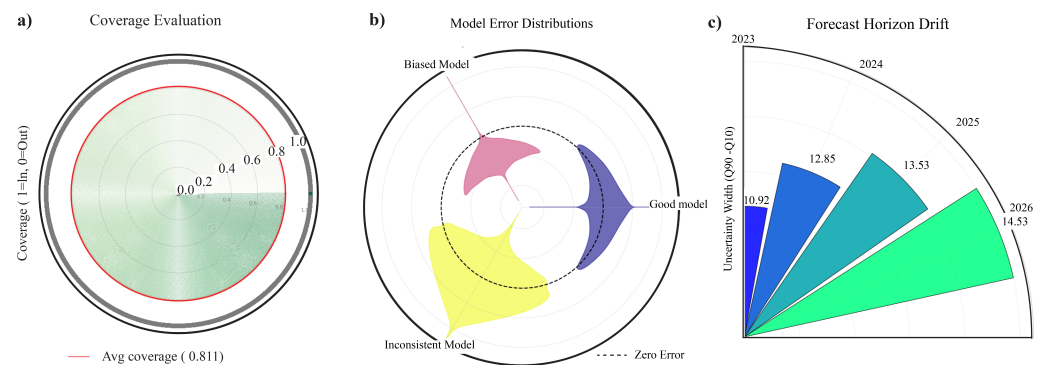


Figure 2: Model performance evaluation. (a) Coverage Evaluation: radial plot comparing empirical coverage against nominal quantile levels (average coverage = 0.811). (b) Model Error Distributions: The radial axis represents the error value, with the dashed circle indicating zero error. The width of each violin shows the density of errors, revealing that the "Good Model" is unbiased and consistent, the "Biased Model" consistently under-predicts, and the "Inconsistent Model" has high variance. (c) Forecast Horizon Drift: radial bar chart of uncertainty width (Q90-Q10) for forecast years 2023–2026, illustrating increasing prediction uncertainty.

These visualization methods were developed alongside research applying advanced deep learning models, such as physics-informed deep learning¹, to complex environmental forecasting challenges. Specifically, these polar diagnostics were utilized to analyze and interpret the uncertainty associated with land subsidence predictions using an Extreme Temporal Fusion Transformer model (Kouadio et al., 2025) in Nansha city, China.

Full usage examples and a gallery of all plot types are available in the official documentation's [gallery section](#). For a deeper understanding of the mathematical concepts and interpretation guides, please refer to the detailed [User Guide](#). The complete mathematical foundations and derivations for all visualizations are provided in the official Technical Report (Kouadio, 2025).

Availability and Community

The latest stable release of k-diagram is available on the Python Package Index and can be installed via `pip install k-diagram`. The package is distributed under the OSI-approved [Apache License 2.0](#). Comprehensive documentation, including a user guide, an example gallery, and a detailed API reference, is hosted at <https://k-diagram.readthedocs.io/>. I welcome contributions from the community; bug reports, feature requests, and development discussions occur on the [GitHub repository](#). Detailed [guidelines](#) for contributors can be found in the documentation.

Acknowledgements

I extend my sincere gratitude to the anonymous colleagues who provided invaluable feedback during the development of k-diagram, as well as those who rigorously tested its early iterations. Their insights and dedication were instrumental in refining the software. I also appreciate the constructive feedback from reviewers and early users, whose contributions have significantly enhanced the quality and usability of the project.

References

- Baillie, R. T., & Chung, S. K. (2002). Modeling and forecasting from trend-stationary long memory models with applications to climatology. *International Journal of Forecasting*, 18(2), 215–226. [https://doi.org/10.1016/S0169-2070\(01\)00154-6](https://doi.org/10.1016/S0169-2070(01)00154-6)
- Biswas, A., Nespoli, L., Azzimonti, D., Zambon, L., Rubattu, N., & Corani, G. (2025). BayesReconPy: A python package for forecast reconciliation. *Journal of Open Source Software*, 10(111), 8336. <https://doi.org/10.21105/joss.08336>
- Brady, R. X., & Spring, A. (2021). Climpred: Verification of weather and climate forecasts. *Journal of Open Source Software*, 6(59), 2781. <https://doi.org/10.21105/joss.02781>
- Caulk, R. A., Törnquist, E., Voppichler, M., Lawless, A. R., McMullan, R., Santos, W. C., Pogue, T. C., Vlugt, J. van der, Gehring, S. P., & Schmidt, P. (2022). FreqAI: Generalizing adaptive modeling for chaotic time-series market forecasts. *Journal of Open Source Software*, 7(80), 4864. <https://doi.org/10.21105/joss.04864>
- Gneiting, T., Balabdaoui, F., & Raftery, A. E. (2007). Probabilistic forecasts, calibration and sharpness. *Journal of the Royal Statistical Society. Series B: Statistical Methodology*, 69(2), 243–268. <https://doi.org/10.1111/j.1467-9868.2007.00587.x>
- Harris, C. R., Millman, K. J., Walt, S. J. van der, Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Memmesheimer, S., Backx, T., & others. (2020). Array programming with NumPy. *Nature*, 585(7825), 357–362. <https://doi.org/10.1038/s41586-020-2649-2>

¹For an overview of the concepts behind Physics-Informed Neural Networks, see: https://fusion-lab.readthedocs.io/en/latest/user_guide/models/pinn/index.html

- Hintze, J. L., & Nelson, R. D. (1998). Violin plots: A box plot-density trace synergism. *The American Statistician*, 52(2), 181–184. <https://doi.org/10.2307/2685478>
- Hong, S., Choi, Y., & Jeon, J. J. (2025). Interpretable water level forecaster with spatiotemporal causal attention mechanisms. *International Journal of Forecasting*, 41(3), 1037–1054. <https://doi.org/10.1016/j.ijforecast.2024.10.003>
- 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>
- Jolliffe, I. T., & Stephenson, D. B. (2012). *Forecast verification: A practitioner's guide in atmospheric science*. Wiley. <https://doi.org/10.1002/9781119960003>
- Kouadio, K. L. (2025). *k-diagram: Technical report — derivations and details* (Version 1.2.3). School of Geosciences; Info-physics, Central South University. <https://doi.org/10.5281/zenodo.17051182>
- Kouadio, K. L., Liu, Z., Liu, R., Bizimana, P. C., Yang, G., & Liu, W. (2025). *XTFT: A Next-Generation Temporal Fusion Transformer for Uncertainty-Rich Time Series Forecasting*. TechRxiv. <https://doi.org/10.22541/au.175390529.91420978/v1>
- Lim, B., Arık, S. O., Loeff, N., & Pfister, T. (2021). Temporal fusion transformers for interpretable multi-horizon time series forecasting. *International Journal of Forecasting*, 37(4), 1748–1764. <https://doi.org/10.1016/j.ijforecast.2021.03.012>
- Liu, J., Liu, W., Allechy, F. B., Zheng, Z., Liu, R., & Kouadio, K. L. (2024). Machine learning-based techniques for land subsidence simulation in an urban area. *Journal of Environmental Management*, 352(2024), 17. <https://doi.org/10.1016/j.jenvman.2024.120078>
- Liu, Y., Davanloo Tajbakhsh, S., & Conejo, A. J. (2021). Spatiotemporal wind forecasting by learning a hierarchically sparse inverse covariance matrix using wind directions. *International Journal of Forecasting*, 37(2), 812–824. <https://doi.org/10.1016/j.ijforecast.2020.09.009>
- Lu, Haw-minn, Kwong, Adrian, & Unpingco, José. (2020). Securing Your Collaborative Jupyter Notebooks in the Cloud using Container and Load Balancing Services. In Meghann Agarwal, Chris Calloway, Dillon Niederhut, & David Shupe (Eds.), *Proceedings of the 19th Python in Science Conference* (pp. 2–10). <https://doi.org/10.25080/Majora-342d178e-001>
- McKinney, W. (2010). Data Structures for Statistical Computing in Python. In S. van der Walt & J. Millman (Eds.), *Proceedings of the 9th python in science conference* (pp. 56–61). SciPy. <https://doi.org/10.25080/majora-92bf1922-00a>
- Pandas development team. (2020). *Pandas-dev/pandas: pandas* (latest). Zenodo. <https://doi.org/10.5281/zenodo.3509134>
- Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Olszewski, B., Blondel, M., Prejbal, F., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, É. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12, 2825–2830. <https://scikit-learn.org/stable/>
- Python Software Foundation. (2008). *Python language reference, version 3.x*. <https://www.python.org>
- Rummens, A., & Hardyns, W. (2021). The effect of spatiotemporal resolution on predictive policing model performance. *International Journal of Forecasting*, 37(1), 125–133. <https://doi.org/10.1016/j.ijforecast.2020.03.006>
- Silverman, B. W. (1986). *Density estimation for statistics and data analysis*. Chapman; Hall/CRC. <https://doi.org/10.1201/9781315140919>
- Sokol, A. (2025). Fan charts 2.0: Flexible forecast distributions with expert judgement. *International Journal of Forecasting*, 41(3), 1148–1164. <https://doi.org/10.1016/j.ijforecast.2024.11.009>

- Taylor, K. E. (2001). Summarizing multiple aspects of model performance in a single diagram. *Journal of Geophysical Research: Atmospheres*, 106(D7), 7183–7192. <https://doi.org/10.1029/2000JD900719>
- Van Rossum, G., & Drake, F. L. (2009). *Python 3 reference manual*. CreateSpace. ISBN: 1441412697
- Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., & others. (2020). SciPy 1.0: Fundamental algorithms for scientific computing in python. *Nature Methods*, 17(3), 261–272. <https://doi.org/10.1038/s41592-019-0686-2>
- Waskom, M. L. (2021). Seaborn: Statistical data visualization. *Journal of Open Source Software*, 6(60), 3021. <https://doi.org/10.21105/joss.03021>