

Label-aware cross-validation for overlapping-horizon prediction

Evgenii Lazarev^{1,*}

¹Independent Researcher

*elazarev@gmail.com

ORCID: 0009-0000-1398-7842

Abstract

Cross-validation is routinely used to estimate out-of-sample performance in statistical learning, but standard shuffled or blocked folds can be invalid when responses are measured over future intervals. A label such as the mean demand over the next twelve half-hours, the next-day rainfall amount, or the return over the next twenty bars overlaps the labels of nearby rows. If overlapping label intervals are split between training and test sets, the validation score partly measures information reuse rather than generalization. This study formalizes split-level conditions for leakage-aware validation in overlapping-label time-series and panel data, and evaluates them through `purgedcv`, an open-source Python implementation of purging, embargoing, walk-forward validation, group-purged folds, and combinatorial purged cross-validation. A controlled experiment with an unpredictable target shows that shuffled k-fold can report a mean out-of-sample R^2 of 0.918 while admitting complete train/test label overlap. A full-population benchmark on Low Carbon London smart-meter data shows a more nuanced case: the temporal leakage gap is small but measurable, whereas the larger issue is household-level generalization. The results show that validation choice is a scientific measurement decision, not only a software parameter.

Keywords: cross-validation; data leakage; time series; panel data; model validation; reproducible software

1 Introduction

Cross-validation is often treated as a neutral measurement device: choose a splitter, fit the same estimator on each training fold, and average the test scores. That view depends on an independence assumption that is not satisfied by many time-indexed prediction tasks. In a forecasting or backtesting problem, row i is usually not just an instantaneous observation. Its response may be defined by an evaluation interval that starts at the prediction time and ends after a future horizon. Nearby rows can therefore share part of the same future outcome window. Standard shuffled k-fold cross-validation can place one row in the test fold and another row whose label interval overlaps it in the training fold. The resulting score is then contaminated by information that would not be available when the model is used prospectively.

Data leakage is a well-known source of inflated performance estimates in machine learning [1]. It is especially damaging in scientific applications where a model is selected or reported after many iterations, since the optimistic score becomes part of the published evidence rather than merely a development mistake [2]. In financial machine learning, López de Prado [3] proposed purging and embargoing as practical guards against leakage from overlapping labels and serial dependence, and Combinatorial Purged Cross-Validation (CPCV) as a way to obtain multiple out-of-sample backtest paths. Bailey and Lopez de Prado's Probabilistic Sharpe Ratio and Deflated Sharpe Ratio address the related problem of selection bias after trying multiple strategies [4, 5].

The same validation problem appears outside finance. Household electricity demand, equipment degradation, rainfall, clinical monitoring, and air-quality forecasting all contain future-horizon labels or repeated entities. What matters is whether any training-label window overlaps a test-label window, whether a post-test serial-dependence buffer has been respected, and whether the deployment target involves unseen entities rather than future observations from already-seen entities.

This study makes three contributions. First, it states a directly checkable interval condition for overlapping-label validation. Second, it presents `purgedcv`, an open Python implementation of purging, embargoing, walk-forward validation, group-purged k-fold, and CPCV path reconstruction through the scikit-learn cross-validation interface, together with leakage diagnostics for auditing splits [6, 7]. Third, it reports reproducible experiments that show both dramatic and undramatic outcomes: a synthetic task

where leakage fabricates strong skill, and a real smart-meter benchmark where the larger issue is not temporal leakage but household-level generalization.

2 Results

2.1 Overlapping labels define a split-level invariant

Let a supervised learning data set contain observations

$$z_i = (x_i, y_i, p_i, e_i, g_i), \quad i = 1, \dots, n,$$

where x_i is the feature vector, y_i is the response, p_i is the prediction time, e_i is the evaluation time at which the response is fully known, and g_i is an optional group identifier such as a household, patient, engine, or season. The label interval for row i is

$$I_i = [p_i, e_i).$$

The interval is half-open: a label window ending exactly where another begins is not counted as overlapping. For a train/test split (A, B) , label-overlap leakage is present when

$$\exists i \in A, \exists j \in B \quad \text{such that} \quad I_i \cap I_j \neq \emptyset.$$

A leakage-aware split must remove such training rows before fitting the estimator. In practice, a second guard is often needed after a test block. If the process remains serially correlated after the test interval, training immediately after the test block can still reuse information tied to that test period. An embargo removes training rows whose prediction time lies inside a post-test buffer of fixed duration or fixed fraction of the sample.

For panel data there is a separate deployment question. If the intended use is prediction on new entities, a chronological split that mixes the same entity across training and test sets may answer the wrong question even when no label intervals overlap. In that case the split must also satisfy

$$\{g_i : i \in A\} \cap \{g_j : j \in B\} = \emptyset.$$

Thus validation has at least three distinct requirements: interval disjointness, post-test embargo, and group disjointness. They are not interchangeable. A fixed integer gap may remove leakage for a single constant horizon, but it does not express variable horizons, time-duration embargoes, CPCV test blocks, or entity-level generalization.

2.2 The software makes validation contracts executable

`purgedcv` implements the interval operations and splitters needed to make these requirements executable. The package is written in Python, is MIT licensed, and follows the `scikit-learn` splitter protocol, so the same objects can be passed to `cross_val_score`, `GridSearchCV`, and `Pipeline`. Runtime dependencies are intentionally small: `numpy`, `pandas`, `scikit-learn`, and `scipy` [6, 8–10]. Table 1 summarizes the public components exposed by the package.

The implementation separates interval arithmetic from splitter orchestration. For each fold, test label intervals are sorted and merged once, and candidate training intervals are tested against the merged set. This avoids duplicating boundary logic across splitters and is particularly important for CPCV, where the test set may contain several non-adjacent blocks. In such a fold, purging must apply to the union of test label intervals, not to the convex hull between the first and last test block.

The diagnostic functions are deliberately independent of the package’s own splitters. They accept training indices, test indices, prediction times, and evaluation times, and can audit a split produced by any library or by hand. This turns the validation contract into an assertion that can be placed in tests.

2.3 A controlled task exposes complete validation failure

The controlled task is designed so that no feature has genuine predictive content. Let ϵ_t be independent noise and define the response at row t as the mean of the next H future noise values. The only feature is a monotone clock. A model cannot forecast the future noise, but shuffled k-fold can exploit overlap

Table 1: Main public API in `purgedcv`.

Component	Function or class	Purpose
Primitive	<code>purge</code>	Drop training rows whose label intervals overlap test labels
Primitive	<code>apply_embargo</code>	Drop post-test training rows inside an embargo buffer
Splitter	<code>WalkForwardSplit</code>	Expanding or rolling chronological validation
Splitter	<code>PurgedKFold</code>	Contiguous folds with label-aware purging and embargo
Splitter	<code>PurgedGroupKFold</code>	Purged folds with disjoint held-out groups
Splitter	<code>CombinatorialPurgedCV</code>	CPCV folds with multiple test-block combinations
Paths	<code>reconstruct_paths</code>	Assemble CPCV folds into backtest paths
Metrics	<code>probabilistic_sharpe_ratio</code>	Probability that skill exceeds a benchmark
Metrics	<code>deflated_sharpe_ratio</code>	Sharpe-ratio inference corrected for selection bias
Metrics	<code>min_track_record_length</code>	Minimum observations needed to establish a Sharpe ratio
Diagnostics	<code>assert_*</code> functions	Check temporal, embargo, and group-leakage invariants

between adjacent future-horizon labels. Large positive R^2 is therefore evidence of validation leakage, not model skill.

Table 2 reports a Random Forest experiment with $n = 1500$, $H = 20$, five outer folds, and seed 0. The overlap column is the mean fraction of training rows whose label window overlaps any test label window, averaged across folds.

Table 2: Controlled leakage task. Positive R^2 is fabricated because the target is unpredictable by construction.

Library	Splitter	Mean R^2	Mean overlap	Folds
scikit-learn	<code>KFold(shuffle=True)</code>	0.918	1.000	5
scikit-learn	<code>KFold(shuffle=False)</code>	-1.017	0.025	5
scikit-learn	<code>TimeSeriesSplit</code>	-2.506	0.035	5
scikit-learn	<code>TimeSeriesSplit(gap=20)</code>	-1.430	0.000	5
purgedcv	<code>PurgedKFold</code>	-0.870	0.000	5
purgedcv	<code>WalkForwardSplit</code>	-1.899	0.000	5
purgedcv	<code>CombinatorialPurgedCV</code>	-1.471	0.000	15
tscv	<code>GapKFold(gap_before=20, gap_after=20)</code>	-1.217	0.000	5
timeseriescv	<code>CombPurgedKFoldCV</code>	-0.894	0.004	15
timeseriescv	<code>PurgedWalkForwardCV</code>	-1.543	0.000	4

The shuffled k-fold score of 0.918 is not a small optimism effect. It is a complete failure of the validation design. The blocked and chronological baselines remove most of the effect but still admit small amounts of overlap unless a suitable gap is supplied. A fixed `TimeSeriesSplit` gap can solve this particular constant-horizon toy problem, but it does not provide label-aware intervals, variable horizons, group-purged folds, diagnostics, or CPCV paths. The `purgedcv` splitters remove the overlap by construction and return negative R^2 , which is the expected outcome for an unpredictable target evaluated out of sample.

2.4 A real smart-meter benchmark separates temporal leakage from entity generalization

The second experiment uses the Low Carbon London smart-meter data set from UK Power Networks and the London Datastore [11]. The prediction task is half-hourly household electricity demand forecasting. Features include calendar and lagged-load information, and the target is a forward-horizon mean. The validation schemes compare pooled shuffled k-fold, blocked k-fold, walk-forward validation, and held-out-household validation.

The full-population benchmark scans 167,932,474 raw rows, identifies 4,284 eligible Standard-tariff households with at least one year of data, draws 20 seeded subsamples of 60 households, and evaluates each validation scheme with the same modeling harness. Table 3 reports mean WAPE and 95% t-intervals. WAPE is $\sum |\hat{y} - y| / \sum |y|$, reported in percent.

Table 3: Low Carbon London benchmark over 20 seeded subsamples of 60 households. Lower WAPE is better.

Metric	Mean	95% CI low	95% CI high
Naive shuffled k-fold WAPE	41.68	40.37	42.99
Blocked k-fold WAPE	42.43	41.07	43.80
WalkForwardSplit WAPE	42.36	41.01	43.71
GroupKFold household WAPE	44.92	43.38	46.45
Temporal gap, WAPE points	0.68	0.53	0.83
Temporal gap, relative percent	1.60	1.27	1.94
Household gap, WAPE points	2.56	2.08	3.03
Household gap, relative percent	6.03	4.93	7.12

By design, the result is less dramatic than the synthetic example. The temporal leakage gap between shuffled k-fold and walk-forward validation is measurable but small: 0.68 WAPE points, or 1.60% relative to walk-forward WAPE. The larger effect is the household gap. Scoring on unseen households is 2.56 WAPE points worse than the pooled temporal estimate, or 6.03% relative. This is the more important conclusion for deployment: if the model will be used for customers not seen during training, a purely temporal split answers a different question.

2.5 Cross-domain examples show that leakage-aware validation can also return negative results

The repository also contains notebooks that exercise the same validation logic on other public data sets. Table 4 summarizes the role of each example. Some are designed to expose a large leakage effect; others show that a leakage-aware split can correctly report a small or absent gap. The “0.83–0.91” range in the first row refers to the companion notebook’s two models, k-nearest neighbors and Random Forest, rather than to multiple random seeds; the Random Forest-only benchmark in Table 2 reports 0.918.

Table 4: Reproducible examples included with the package.

Example	Data source	Main validation lesson
Synthetic proof	leakage Generated	k-nearest neighbors and Random Forest report R^2 of 0.83–0.91 on noise
Air quality	UCI air-quality data	A clock feature plus overlapping labels fabricates R^2 near 0.99
Earthquakes	USGS catalogue	Magnitude history has no skill; purged splits reject the illusion
Smart meters	Low Carbon London	Household generalization dominates temporal leakage
Clinical mortality	PhysioNet Challenge 2012	Whole-patient group holds are needed for patient-level inference
Predictive maintenance	NASA C-MAPSS	Walk-forward validation matches run-to-failure deployment
Rainfall	NOAA GHCN-Daily	One-day-ahead labels need purge and embargo buffers
Electricity load	PJM hourly load	CPCV paths expose score dispersion across backtest paths
Model comparison	Binance public bars	DSR prevents selecting an apparent edge after multiple trials
Sports prediction	Premier League matches	Honest validation shows calibration drift rather than a headline gap

The examples deliberately include negative results. In the model-comparison notebook, several models are tried on the same public price data. Once the Deflated Sharpe Ratio corrects for the number of trials, no model clears a DSR threshold of 0.95. In the PJM electricity-load notebook, CPCV produces five paths whose DSR values range from 0.0011 to 0.7761 after correction for 20 trials. These are not failures of the software. They show that the validation pipeline is capable of reporting that no reliable edge survived.

3 Discussion

The experiments show that leakage-aware validation is not a single recipe. In the controlled task, randomization is catastrophic because every test label has overlapping training labels. In the smart-meter benchmark, the temporal effect is small but statistically visible, while the larger operational issue is whether the model is expected to generalize to new households. In other domains, the required split can be driven by patients, engines, seasons, stations, or market regimes. The validation object should encode that deployment question rather than being chosen only for convenience.

`purgedcv` therefore treats diagnostics as first-class objects. A user can construct a split with this package, with another package, or by hand, and then check the interval and group invariants directly. This matters for reproducibility. A reported model score is only as meaningful as the split that created it, and the split should be auditable from code rather than described informally in prose.

Several packages overlap with part of this problem. `scikit-learn` provides `TimeSeriesSplit`; its `gap` argument is a fixed integer count rather than a label-aware interval, and it does not provide group-purged folds, CPCV paths, or split-level diagnostics. `tscv` provides fixed-gap splits, which are useful when the required buffer is known and constant, but it does not represent variable label horizons or grouped deployment targets. `timeseriescv` implements purged and combinatorial time-series cross-validation, but it does not unify variable-horizon label intervals, group-purged folds, post-test embargoes, CPCV path reconstruction, and independent diagnostic assertions in a typed `scikit-learn`-compatible package [12]. `mlfinlab` is the best-known implementation associated with the financial machine-learning literature, but it is distributed as a commercial product and therefore cannot serve as a permissive dependency for open scientific software [13]. The companion benchmark also records two non-tabulated open alternatives: `mlfinpy` did not run on the modern `pandas` stack used here, and `RiskLabAI` failed because a plotting dependency was unavailable. Those failures are recorded with exact exception messages rather than imputed scores.

`purgedcv` is therefore not differentiated by claiming new purging mathematics. Its contribution is integration and auditability. Unlike fixed-gap splitters or single-purpose CPCV implementations, `purgedcv` unifies variable-horizon label intervals, group-purged folds, post-test embargoes, CPCV path reconstruction, and split-level diagnostics as assertions that can be run on third-party or hand-written splits. This combination is what lets the same validation contract be used in ordinary `scikit-learn` model selection, in notebook examples, and in automated tests.

There are limitations. Purging and embargoing remove a specific class of validation leakage; they do not solve all forms of leakage. Feature engineering can still use future data, target transformations can still be computed globally, preprocessing can still be fit outside the training fold, and entity leakage can still occur if the wrong group identifier is supplied. The package does not claim that every chronological split is optimal. In highly non-stationary settings, any historical validation estimate can be unstable. The role of the package is narrower: when labels are interval-valued, it makes the no-overlap condition explicit and executable.

Another limitation is maturity. The package is new, even though the underlying methods are established. The open repository contains tests, type checks, documentation, notebooks, and reproducible benchmarks, but wider external use will be needed to discover edge cases in unfamiliar data layouts. For this reason the software should be treated as validation infrastructure whose outputs remain the analyst’s responsibility, not as an automatic guarantee of scientific validity.

Overlapping-label prediction problems require more than a chronological split. The validation design must remove training labels that overlap test labels, respect any post-test dependence buffer, and match the entity structure of the deployment target. The practical contribution is not that the package makes models better, but that it makes their validation harder to fool.

4 Methods

4.1 Purging, embargoing, and group separation

The implementation follows the half-open interval convention $I_i = [p_i, e_i)$. A training row is purged from a fold if its label interval intersects any test label interval. Boundary-touching intervals are not purged: if $e_i = p_j$ or $e_j = p_i$, the intersection is empty under the half-open convention. This matches common time-series indexing practice and prevents unnecessary deletion when one forecast horizon ends exactly at the next prediction time.

Embargoing is applied after purging. It removes training rows whose prediction times fall inside a post-test buffer. The buffer can be expressed as a duration for timestamped data or as a fraction of the sample. In grouped validation, group-purged splitters additionally require disjoint train and test groups, so that deployment to unseen entities can be evaluated directly.

4.2 Combinatorial purged cross-validation

CPCV divides the ordered sample into N contiguous blocks and holds out k blocks at a time, producing $\binom{N}{k}$ test-block combinations. Each combination defines a fold whose training side is purged and embargoed against the union of its held-out test label intervals. Because the held-out blocks can be non-adjacent, purging is performed against merged test intervals rather than against the convex hull spanning all held-out blocks. The resulting fold predictions can be recombined into several out-of-sample paths, giving a distribution of validation trajectories rather than a single historical path.

4.3 Implementation and diagnostics

The package separates low-level interval operations from splitter orchestration. The same interval functions are used by `PurgedKFold`, `PurgedGroupKFold`, `WalkForwardSplit`, and `CombinatorialPurgedCV`. Diagnostic assertions accept train indices, test indices, prediction times, evaluation times, and optional group labels. They can therefore audit splits produced by `purgedcv`, by another library, or by custom code.

The package is maintained as a public open-source project with continuous integration, strict static typing, and an extensive test suite covering split invariants, numerical metrics, end-to-end reproducibility, notebook-derived fixtures, and packaging quality gates. The repository accepts issues and pull requests, and the core validation behavior is protected by tests rather than by example output alone.

4.4 Controlled leakage benchmark

The controlled benchmark uses $n = 1500$ observations and a horizon of $H = 20$. For each row t , the response is the mean of the next H independent noise values. The feature matrix contains only a monotone time index. Random Forest regression is evaluated with five outer folds for ordinary k-fold, time-series splits, fixed-gap baselines, and `purgedcv` splitters. The reported overlap metric is the mean fraction of training rows whose label window overlaps any test label window. Competitor rows are generated by the repository script `tools/competitor_benchmark.py`; unavailable competitors are recorded as NOT RUN with exact exception messages.

4.5 Low Carbon London benchmark

The Low Carbon London benchmark uses the public smart-meter corpus released by UK Power Networks and the London Datastore [11]. The benchmark script scans the raw CSV files, retains Standard-tariff households with at least one year of data, and samples 20 independent groups of 60 households. For each subsample, the same feature builder, estimator, and scoring code are used across pooled shuffled k-fold, blocked k-fold, walk-forward validation, and held-out-household validation. The primary score is WAPE, $\sum |\hat{y} - y| / \sum |y|$, reported in percent. Confidence intervals are ordinary 95% t-intervals across the 20 seeded subsamples.

4.6 Cross-domain notebooks and statistical metrics

The companion notebooks apply the same validation checks to public data from earthquake catalogues, air-quality monitoring, smart meters, clinical mortality prediction, engine run-to-failure simulation, rainfall records, electricity load, public market bars, and sports prediction. The financial-statistical metrics in `purgedcv` implement the Probabilistic Sharpe Ratio, Deflated Sharpe Ratio, and Minimum Track Record Length following Bailey, Lopez de Prado, and coauthors [4, 5]. These metrics are included because model selection after many trials can itself create optimistic evidence, even when a split is temporally valid.

4.7 Computational environment and reproducibility

The benchmark tables reported here were produced with `purgedcv` 0.0.6, Python 3.12.7, `numpy` 2.4.5, `pandas` 3.0.3, `scikit-learn` 1.8.0, and `scipy` 1.17.1 on macOS 26.3.1 (arm64). The current released version

of `purgedcv` at the time of this manuscript is 0.0.10; the change set from 0.0.6 to 0.0.10 is limited to input-validation hardening, CI tooling, and documentation, with no changes to the cross-validation algorithms or to the numerical paths exercised by the reported computations. The package supports Python 3.10 through 3.14; runtime dependency lower bounds are `numpy` 1.24, `pandas` 2.0, `scikit-learn` 1.3, and `scipy` 1.10.

A split-generation microbenchmark is tracked as `tools/microbench.py`. It uses 1,000,000 times-tamped rows, five folds, one feature, a constant 20-second label horizon, and no estimator fitting. In the recorded local run, `PurgedKFold` generated the five folds in 1.898 seconds best-of-three (mean 1.911 seconds), and wrote the environment details to `paper/microbench_summary.md`. The full Low Carbon London benchmark scanned 167,932,474 raw rows and ran 20 seeded 60-household subsamples in 53.8 minutes on the author’s local machine.

The main local reproduction commands are:

```
pip install -e ".[dev,examples]"
pip install tscv timeseriescv
pytest -q
python tools/microbench.py
python tools/competitor_benchmark.py --out-dir examples/data
python tools/lcl_full_benchmark.py --k 20 --n 60 --seed 0
```

The competitor command above reproduces the reported Table 2 rows when `tscv` and `timeseriescv` are installed. For a fast core-only smoke check, use `-core-only`. The Low Carbon London command expects the raw CSV files to be present locally. For faster checks, the repository includes end-to-end tests with synthetic fixtures that exercise the same parser, feature builder, and benchmark output format.

4.8 Use of generative AI tools

Large language model tools, including OpenAI Codex and ChatGPT in the GPT-5 family and Anthropic Claude in the Claude 4 family, were used as assistants for code review, refactoring suggestions, test scaffolding, documentation drafting, copy-editing, and pre-submission checks. These tools were not treated as authors. All design decisions, AI-assisted changes, and outputs were reviewed and validated by the author through unit, property, doctest, end-to-end, type-checking, linting, and benchmark tests; no claim was accepted without source or executable verification.

Data availability

The synthetic benchmark data are generated by scripts included in the public repository. The Low Carbon London raw smart-meter files are publicly available from UK Power Networks and the London Datastore; they are not redistributed in this repository because of their size. The repository contains the benchmark scripts, derived summaries, and notebook fixtures needed to reproduce the reported tables once the raw public files are available locally. The other examples use public data sources identified in the notebooks and reference list, including the U.S. Geological Survey catalogue [14], UCI air-quality data [15], NOAA GHCN-Daily [16], NASA C-MAPSS [17], PhysioNet [18, 19], PJM hourly load data [20], and Binance public market data via `pricehub` [21].

Code availability

The software is available from the project repository at <https://github.com/eslazarev/purged-cross-validation> and distributed on PyPI as `purgedcv` at <https://pypi.org/project/purgedcv/>. The source distribution contains the examples and benchmark tools; the wheel contains the importable `purgedcv` package. The software is MIT licensed and archived on Zenodo with DOI 10.5281/zenodo.20312695. The current released version at the time of this manuscript is 0.0.10.

References

- [1] Shachar Kaufman, Saharon Rosset, Claudia Perlich, and Ori Stitelman. Leakage in data mining: Formulation, detection, and avoidance. *ACM Transactions on Knowledge Discovery from Data*, 6(4): 1–21, 2012. 10.1145/2382577.2382579.

- [2] Matthew B. A. McDermott, Shirly Wang, Nikki Marinsek, Rajesh Ranganath, Luca Foschini, and Marzyeh Ghassemi. Reproducibility in machine learning for health research: Still a ways to go. *Science Translational Medicine*, 13(586), 2021. 10.1126/scitranslmed.abb1655.
- [3] Marcos López de Prado. *Advances in Financial Machine Learning*. Wiley, Hoboken, NJ, 2018. ISBN 9781119482086. Purging and embargoing: chapter 7. Combinatorial Purged Cross-Validation: chapter 12.
- [4] David H. Bailey and Marcos López de Prado. The sharpe ratio efficient frontier. *Journal of Risk*, 15(2):3–44, 2012.
- [5] David H. Bailey and Marcos López de Prado. The deflated sharpe ratio: Correcting for selection bias, backtest overfitting, and non-normality. *Journal of Portfolio Management*, 40(5):94–107, 2014.
- [6] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
- [7] Evgenii Lazarev. `purgedcv`: scikit-learn-compatible purged and combinatorial cross-validation for time-series and panel machine learning. Python package, <https://github.com/eslazarev/purged-cross-validation>, 2026. Software concept DOI; MIT license.
- [8] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández Del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, 2020. 10.1038/s41586-020-2649-2.
- [9] Wes McKinney. Data structures for statistical computing in Python. In *Proceedings of the 9th Python in Science Conference*, pages 56–61, 2010. 10.25080/Majora-92bf1922-00a.
- [10] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C. J. Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, and Paul van Mulbregt. SciPy 1.0: Fundamental algorithms for scientific computing in Python. *Nature Methods*, 17:261–272, 2020. 10.1038/s41592-019-0686-2.
- [11] UK Power Networks. Smartmeter energy consumption data in london households (low carbon london). London Datastore, 2014. Dataset `smartmeter-energy-use-data-in-london-households`; open terms.
- [12] timeseriescv contributors. `timeseriescv`: scikit-learn style cross-validation for time series. Python package, <https://pypi.org/project/timeseriescv/>, 2018. Version 0.2.
- [13] Hudson and Thames. `mlfinlab`: Financial machine learning package. Software product, <https://hudsonthames.org/mlfinlab/>, 2026. Accessed 2026-05-21.
- [14] U.S. Geological Survey. Earthquake catalog. USGS FDSN event web service, 2026. Public domain. <https://earthquake.usgs.gov/fdsnws/event/1/>; accessed 2026-05-21.
- [15] S. De Vito, E. Massera, M. Piga, L. Martinotto, and G. Di Francia. On field calibration of an electronic nose for benzene estimation in an urban pollution monitoring scenario. *Sensors and Actuators B: Chemical*, 129(2):750–757, 2008. 10.1016/j.snb.2007.09.060.
- [16] Matthew J. Menne, Imke Durre, Russell S. Vose, Byron E. Gleason, and Tamara G. Houston. An overview of the global historical climatology network-daily database. *Journal of Atmospheric and Oceanic Technology*, 29(7):897–910, 2012. 10.1175/JTECH-D-11-00103.1.

- [17] Abhinav Saxena, Kai Goebel, Don Simon, and Neil Eklund. Damage propagation modeling for aircraft engine run-to-failure simulation. In *International Conference on Prognostics and Health Management*, 2008. 10.1109/PHM.2008.4711414.
- [18] Ary L. Goldberger, Luis A. N. Amaral, Leon Glass, Jeffrey M. Hausdorff, Plamen Ch. Ivanov, Roger G. Mark, Joseph E. Mietus, George B. Moody, Chung-Kang Peng, and H. Eugene Stanley. Physiobank, physiokit, and physionet: Components of a new research resource for complex physiologic signals. *Circulation*, 101(23):e215–e220, 2000. 10.1161/01.CIR.101.23.e215.
- [19] Ikaro Silva, George Moody, Daniel J. Scott, Leo A. Celi, and Roger G. Mark. Predicting in-hospital mortality of icu patients: The physionet/computing in cardiology challenge 2012. In *Computing in Cardiology*, volume 39, pages 245–248, 2012.
- [20] PJM Interconnection. Hourly metered load data. Public historical load data, 2026. Mirror used by the example; public domain dedication (CC0); accessed 2026-05-21.
- [21] Evgenii Lazarev. pricehub: Unified ohlc market-data fetcher. Python package, <https://pypi.org/project/pricehub/>, 2026. Binance public spot market data; subject to the exchange API terms.

Acknowledgements

The author thanks the maintainers of the open data sets used in the reproducible examples and the maintainers of the scientific Python ecosystem. The purging, embargoing, CPCV, PSR, DSR, and MinTRL methods implemented in `purgedcv` are due to Lopez de Prado, Bailey, and colleagues; any implementation errors are the author's.

Author contributions

E.L. conceived the study, implemented the software, designed and ran the experiments, analyzed the results, wrote the manuscript, prepared the reproducibility materials, and approved the final version.

Additional information

Competing interests

The author declares no competing interests.

Funding

No external funding was received for this work.

Ethics declarations

This study did not involve new experiments with human participants, human tissue, live vertebrates, or higher invertebrates. The empirical examples use publicly released or synthetic data; the Low Carbon London data are used in the public anonymized form released by the original data providers.