forecasting

MDPI

Technical Note

4Cast Lab: Open-Source App for Machine Learning and Statistical Analysis

Periklis Gogas

Department of Economics, Democritus University of Thrace, 69100 Komotini, Greece; pgkogkas@econ.duth.gr

Highlights

What are the main findings?

- On 59 M4 monthly series and the US unemployment rate, 4Cast Lab forecasts as accurately as the same learners in MLForecast, and close to StatsForecast's AutoARIMA once the target is defined as a change; in levels, it is clearly less accurate, and it says so.
- When the target is a random walk, reporting on the best of ten models on the Test Set declares a false forecasting success in 32% of simulated data sets and a high R^2 in over 90%; the benchmark-first verdict does so in 0 to 1%.

What are the implications of the main findings?

- Time-ordered splits, model selection inside the training period, a naive benchmark with a formal test and a correction for the number of models tried can be the default of forecasting software, at no cost in accuracy.
- Reporting "no model beats the benchmark" as a normal outcome protects analysts, and the readers of their work, from the most common false claims of forecasting success.

Abstract

Forecasting with machine learning often goes wrong due to the protocol and not the algorithms for reasons including leakage, model selection on the test sample, no naive benchmark, and no allowance for the number of models tried. 4Cast Lab is a free, open-source, single-file application that makes a benchmark-first protocol the default: time-ordered splits; selection by cross-validation inside the training period; one use of the Test Set; a naive benchmark with Diebold–Mariano and Clark–West tests; and the Holm, Reality Check, SPA and Model Confidence Set procedures. We document the protocol, the leakage controls and the fixed rules of the AutoML verdict, and report three kinds of evidence. A four-stage benchmark against certified values and reference software shows agreement to at least six significant digits and pre-specified equivalence tests for the random learners. A simulation run through the application shows that, for random-walk targets, common reporting rules declare a false success in 21 to 95% of cases and the benchmark-first verdict in 0 to 1%, at a measured cost in power. On 59 M4 monthly series and the US unemployment rate, the application matches MLForecast for like learners and approaches StatsForecast once the target is differenced.

Download: https://github.com/PerryGogas/4Cast-Lab/releases/latest

Use online or install as a web app: https://perrygogas.github.io/4Cast-Lab/

Keywords: forecasting software; forecast evaluation; naive benchmark; data leakage; model selection; data snooping; machine learning; AutoML; reproducibility; open-source software

Check for updates

Academic Editor: Sonia Leva

Received: 14 October 2026
Revised: 21 November 2026
Accepted: 23 November 2026
Published: 25 November 2026

Copyright: © 2026 by the author. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

JEL Classification: C45; C52; C53; C87; A22

1. Introduction

Machine learning is by now a routine tool in forecasting. It is also a tool that is misused quite often, and the literature on forecasting practice keeps returning to the same list of problems. Information from the future finds its way into the training data. Models are chosen on the very sample that is then used to report how accurate they are. Accuracy is reported with no naive benchmark next to it, and the best of many models is presented as if it had been the only one tried [1–3]. These are not failures of the learning algorithms. They have to do with the protocol that surrounds the algorithms, and in a general-purpose library that protocol is the user's own responsibility.

In this paper we present 4Cast Lab, a free and open-source application in which a defensible protocol is the default setting. We refer to it as *benchmark first*. The data are split in time order. Each algorithm is tuned, and the model is selected, by cross-validation within the training period, so that the Test Set is used only once. The selected model is then compared with a naive benchmark, itself chosen on the training rows, by means of a formal test of predictive accuracy, and the comparison is adjusted for the number of models that were tried. The outcome is reported to the user in plain words. This includes the case, not at all rare in practice, where no model does better than the benchmark.

The contribution of the paper is threefold, and the revised manuscript is organized accordingly.

1. A protocol and evidence that it matters. We state the benchmark-first protocol precisely (Sections 2.3–2.5) and measure, in a simulation run through the application itself, how often common alternatives declare a forecasting success when the truth is a random walk (Section 3.2).
2. A validated open implementation. All learners, estimators and tests are implemented in the application and were benchmarked against certified values and reference software in four stages, with pass rules fixed in advance (Sections 2.9 and 3.1; Supplementary Material S1).
3. A comparison with established tools on real data. On a random sample of M4 monthly series and on the US unemployment rate, we compare the application with StatsForecast and MLForecast under identical splits, horizons and benchmarks, and we set out what each tool reports to its user (Sections 3.3–3.5).

This is a technical note about a piece of software. We do not claim that 4Cast Lab produces better forecasts than the specialized libraries, and in fact Section 3.3 shows that it does not. What we do claim is more limited. An analyst who simply follows the defaults gets forecasts that are as accurate as those of the same learners in a standard library and gets with them the checks that show whether these forecasts are worth using at all. The description refers to version 6.1.1. Compared with version 5.0, which was described in the original submission, the application now also includes panel-data econometrics, difference in differences, dynamic panel GMM, structural VAR, count and limited-dependent models, a Patterns tab (clustering and principal components), public data sources, and Newton-step gradient boosting. These are listed briefly in Section 2.8 and are not discussed further.

2. Materials and Methods

2.1. Design and Architecture

4Cast Lab is a single HTML file written in JavaScript. All learning algorithms, estimators and tests are implemented in the file; Chart.js (version 4.5.1) draws the figures and SheetJS (version 0.18.5) reads and writes Excel files. The same file runs in three ways: opened from a folder in any modern browser; as a portable Windows executable that needs no installation or administrator rights; and as an installable web app from https://perrygogas.github.io/4Cast-Lab/ (accessed on 22 November 2026) that keeps working offline.

Why JavaScript and not Python in the browser. Since the application already includes a Python environment (Pyodide, Section 2.8), one may ask why the numerical core was not simply handed over to scikit-learn and statsmodels running in WebAssembly. There were four reasons for not doing so. The first is start-up: the single file opens within a few seconds and needs no network, while the Python runtime with its scientific packages is a download of tens of megabytes that has to be repeated, or cached, on each machine. The second is responsiveness: a training run returns control to the interface after every cross-validation fold, so the user can follow it and stop it, something that is not possible in the same way with a long Python call. Third, the construction of the folds, the purging, the per-fold preprocessing and the tests of Section 2.5 have to be exactly the same for all learners, and this is much easier to guarantee when all learners live in one code base. Finally, there is a teaching argument, as every algorithm can be read in a single file. The price is that the implementations are new code, and new code has to be validated; this validation is reported in Sections 2.9 and 3.1. Python remains in the application as a checking layer, since every run can be exported as a script that repeats the analysis with the standard packages and prints the two sets of numbers side by side.

Three views share the same engine. The Core view shows the essential settings; the Expert view shows all options, the Estimation mode and the Python Workbench; the AutoML view replaces the settings with a four-step wizard (Section 2.6). The settings are grouped as the steps of an analysis (1: Data, 2: Target, 3: Features, 4: Validation, 5: Model, 6: Decision). Every run records its settings and random seed and can be saved as a session file that reproduces the results.

2.2. Privacy and Network Access

The original manuscript stated that no data leave the machine. As the first reviewer pointed out, this was too absolute, and we now state the position more carefully. All computations are carried out by the browser engine on the user's own computer, and the user's data file is never transmitted by the application. Table 1 lists all features that open a network connection, together with what is transmitted in each case and the default setting. If the update check is switched off and the optional features are not used, no connection is opened at all; the application runs normally on a computer that is not connected to the internet.

Table 1. Network access in 4Cast Lab 6.0. Every feature not listed works offline.

Feature	Connects to	What Is Sent	What Is Received	Default
Update check	GitHub API (releases of the project)	A request for the latest release; no data, no identifier	Version number and release notes	On, at most once a day; can be turned off in Preferences
Python Workbench	jsDelivr CDN (Pyodide); PyPI when the user adds a package	Requests for the runtime and packages	Python runtime and packages, then cached	Off until the Python tab is first opened

Table 1. *Cont.*

Feature	Connects to	What Is Sent	What Is Received	Default
AI interpretation	The provider chosen by the user (OpenAI, Anthropic, Google, xAI or Azure OpenAI)	The results on screen (settings, metrics table, predictor names) and the user's API key; never the data file	A written interpretation	Off; runs only when the user enters a key and presses the button
Public data sources	World Bank, OECD, Eurostat, ECB, FDIC, or a file address typed by the user	The codes of the requested series	The requested series	Off; runs only when the user asks for a download
Help links	GitHub, YouTube, feedback form	Nothing (links open in the browser)		Only when clicked

The API key of the AI interpretation is kept in memory for the session only; it is not written to sessions, reports, preferences or files. The application contains no analytics, advertising or telemetry code.

2.3. Forecasting Workflow and Learners

A run in Forecasting mode follows the six settings groups. Data are imported from CSV or Excel files; an import report lists column types, missing values, date coverage, gaps and duplicates. The target can be a number, two classes or three to twenty classes. For forecasting, the target is shifted to $y(t + h)$, and lags, differences and growth rates are built within each unit of a panel. Eight algorithms are available, singly or all at once (*Train All Models & Compare*): linear and logistic regression; k-nearest neighbors; decision tree; random forest [4]; gradient boosting [5]; support vector machines with linear, RBF and polynomial kernels [6,7]; a feed-forward neural network; and stacking. With the three kernels, *Train All* compares ten configurations. Hyperparameters are tuned by cross-validation on the training set only, with a quick grid, full grids or random search [8], and optional nested cross-validation. One metric chosen by the user serves both tuning and selection (RMSE by default for numbers, AUC for classes).

Regression is reported with RMSE, MAE, MAPE, sMAPE, R^2 and MASE [9], with conformal prediction intervals [10]; classification with ROC-AUC, PR-AUC, accuracy, balanced accuracy, sensitivity, specificity, F1, MCC, the Brier score and log loss. Every metric is shown separately for the cross-validation folds, the training rows and the Test Set. Permutation importance [11], Shapley values [12], and partial-dependence and ICE curves describe the fitted model.

2.4. Leakage Control, Step by Step

By leakage, we mean the use, during training, of information that would not have been available at the time the forecast is made [13]. In the application it is controlled separately at every step of the pipeline, always by the same rule: whatever has to be estimated is estimated on the training rows only, and within cross-validation it is estimated anew inside each fold, using the fitting rows of that fold and nothing else. Validation folds always come after their fitting rows in time [14]. Table 2 goes through the steps one by one. Since version 6.1.1, the rule holds without exception in both the manual workflow and AutoML.

Table 2. Where each step of the pipeline is fitted.

Step	Fitted on	Inside Cross-Validation
Target shift $y(t + h)$, lags, differences, growth rates	No estimation; each row uses values dated t or earlier, within its panel unit	Same rows
Split	Time order: the Test Set is the last block of dates (panels: split by date, all units together)	Expanding or sliding window: every validation fold comes after its fitting rows

Table 2. *Cont.*

Step	Fitted on	Inside Cross-Validation
Purge	A chosen number of rows is removed at the end of the training period	The same gap before every validation fold
Imputation (mean or median) and outlier bounds	Training rows; applied unchanged to the Test Set	Refitted on the fitting rows of each fold, in the manual workflow and in AutoML
Scaling (z-score or min-max)	Training rows; applied unchanged to the Test Set	Refitted on the fitting rows of each fold
One-hot encoding	Applied to all rows before the split; nothing is estimated	Same
Variable selection (LASSO, elastic net, RFECV, stability selection, Boruta, filters)	Training rows	Repeated inside every fold on the fitting rows of that fold, so the selection never sees the validation rows of the fold or the Test Set (manual workflow and AutoML). If the user changes the ticked predictors by hand afterwards, they count as chosen by the user and the card says that nothing is repeated
Hyperparameter tuning and model selection	Cross-validation on the training rows	Nested cross-validation (optional) repeats the search inside each outer fold
Probability calibration and cut-off rules	Out-of-fold predictions of the training rows	Each fold takes its cut-off from the other folds (walk-forward: from earlier folds)
Conformal intervals	Out-of-fold residuals of the training rows	Not applicable
Benchmark choice (random walk or training mean)	Training rows	Not applicable
Test Set	Nothing is fitted; used once to evaluate the selected model	Not applicable

A warning is shown when a random split is chosen for dated data, when a predictor alone reproduces the target (possible target leakage), and when Test Set predictors lie outside their training range.

Fold-specific preprocessing and selection. In the version described in the first revision of this paper (6.0), three data-dependent steps of the manual workflow were not repeated inside the folds. The values used for imputation and the outlier bounds were computed once on the whole training period, and the result of the variable selection card was obtained once on all training rows and then used in every fold; only AutoML repeated them fold by fold. As the second reviewer pointed out, in that design the rows of a validation fold have some influence on how the data are prepared and on which predictors are kept, so that cross-validation scores can be too favorable and the selection of the model can be affected. The Test Set was never involved, but the validation could not be called leakage-safe without qualification. This has been corrected in the software. From version 6.1.1 onwards, imputation values, outlier bounds and the selection of predictors are all recomputed inside each fold from its fitting rows, and the same holds for the inner folds of nested cross-validation. The final model is still fitted on the whole training period, with the steps estimated on that period, and it is evaluated once on the Test Set as before. Two consequences should be noted. First, for analyses that use any of these three steps, the cross-validation scores of version 6.1.1 may differ from those of earlier versions; this is the intended effect, and it is stated in the release notes. Second, none of the results reported in this paper is affected. The simulated data sets, the M4 series and the unemployment series have no missing values; no outlier treatment was applied; and all predictors were used without selection, so the three steps were never active. We confirmed this by running a sample of the data sets again on version 6.1.1, which reproduced the earlier results exactly (Section 3.2). The size of the correction was measured on test data made for the purpose (addendum to Supplementary Material S1). Imputation and the clipping of outliers moved the cross-

validation scores by 0.03 or less. Variable selection is a different matter: with 80 rows and 120 predictors of pure noise, selecting once on all training rows gave a cross-validated AUC of 0.77 for a model that has no predictive ability at all, whereas selecting inside each fold gave 0.42, close to the 0.50 expected by chance [15].

Multi-step horizons. Forecasts for more than one step ahead are direct. For a horizon h, the target of row t is $y(t + h)$ and all predictors are dated t or earlier. Each row is therefore a complete pair, and no Test Set target is ever used in training. Overlapping horizons raise two further issues. The last rows of the training period have targets that are dated up to $h - 1$ periods after the first forecast origin of the Test Set. These are not Test Set targets, yet in real time they would not have been observed. They can be removed with the *Purge* setting [16]: when the purge is set to $h - 1$ rows, all models are trained only on targets that were known at the first Test Set origin, and the same gap is left before each validation fold. It should be noted that the application does not set this purge automatically. In the comparisons reported below we set it to $h - 1$ ourselves. The second issue is handled by the *Rolling-Origin Evaluation*, where the real-time information set holds by construction, because at each origin a direct model is fitted only on rows whose target is already known. For $h > 1$, the forecast errors overlap, so they are serially correlated up to lag $h - 1$ even when the forecast is optimal. The tests of Section 2.5 take this into account in their variance.

2.5. Benchmark, Selection and Tests

On time-ordered data, every model is compared with the stronger of the random walk and the training mean, chosen on training rows only; for classes, the benchmark is the base rate. Model selection is kept separate from evaluation: *Train All* gives each algorithm its own cross-validated search and selects the model with the best mean cross-validation score, and the Test Set is used once, to evaluate the selected model.

Inferential status of the Test Set comparisons. On the Test Set, the selected model is compared with the benchmark and also with each of the other models. These two kinds of comparison do not have the same status. Since the model was selected using the training rows alone, the Test Set observations are independent of the selection, and comparing the selected model with the benchmark amounts to a valid test of a single hypothesis that was specified in advance. The comparisons with the remaining models are conditional on the selection, and there are several of them. We present them only as a description of how close the choice was. They are not a second round of selection, and the application never changes the selected model because of them. Three safeguards are in place. (i) The pairwise *p*-values are adjusted by the Holm procedure [17] for the number of comparisons, and it is the adjusted values that enter the verdict shown to the user. (ii) The same pairwise tests are also computed on the out-of-fold predictions from cross-validation and are displayed next to the Test Set results, so an analyst who would rather compare models within the training period, keeping the Test Set for a final confirmation, has both sets of results on one card. (iii) Whether *any* model beats the benchmark, once the whole search is taken into account, is examined with White's Reality Check [18] and Hansen's test of superior predictive ability [19], while the Model Confidence Set [20] gives the models that cannot be distinguished from the best one. The settings of all the tests are collected in Table 3.

Table 3. Settings of the forecast-comparison tests in 4Cast Lab 6.0.

Test	Null Hypothesis	Loss and Statistic	Variance, Bootstrap and Defaults
Diebold–Mariano [21]	Equal expected loss of two forecasts	Squared error (RMSE, R^2), absolute error (MAE) or the loss of the selection metric; mean loss differential over its standard error	Long-run variance with rectangular weights up to lag h − 1; small-sample correction of Harvey, Leybourne and Newbold [22]; t distribution with n − 1 degrees of freedom; two-sided
Clark–West [23]	Equal population accuracy of the benchmark and a model that nests it	Squared-error differential adjusted for the noise of the larger model	Newey–West (Bartlett) variance with h − 1 lags; standard normal; one-sided
Pesaran–Timmermann [24]	No ability to predict the direction of change	Share of correctly predicted directions	Standard normal
DeLong [25]; McNemar [26]	Equal AUC; equal error rates (classes)	AUC difference; discordant pairs	DeLong covariance; exact binomial below 25 discordant pairs, otherwise chi-square with continuity correction
Paired bootstrap interval	(interval for the difference in the metric)	Difference between the selected model and each other model	B = 1000 resamples of Test Set rows (i.i.d. by default, moving blocks as an option); simultaneous level by Bonferroni; seeded
Holm adjustment [17]	Family of all pairwise comparisons with the selected model		Step-down adjusted *p*-values; verdict at 5%
Reality Check [18]	No model has lower expected loss than the benchmark	Largest mean loss differential times √n, not studentized	Stationary bootstrap [27]; mean block length by the automatic rule of Politis and White [28] (at least 1; an i.i.d. bootstrap for unordered data); B = 1000; re-centered on all models
SPA test [19]	Same as the Reality Check	Largest studentized mean loss differential	Same bootstrap draws; each differential studentized with its full-sample long-run variance (Hansen's definition); consistent *p*-value reported with its lower and upper bounds
Model Confidence Set [20]	Equal expected loss within the remaining set	Range statistic; elimination of the worst model	Same bootstrap; confidence level 90% by default (75, 80, 90 or 95%); MCS *p*-value for every model

h: forecast horizon; n: number of Test Set rows. All bootstrap procedures use the seed of the run when *Replicable sampling* is on (the default, seed 42), so the *p*-values are reproducible; B and the block length can be changed in the Expert view.

2.6. Overfitting Check, Rolling Origins and AutoML

An Overfitting Check card collects eight checks of the selected model on the selection metric: the gap between the training and cross-validation scores; the spread across folds; plain against nested cross-validation; the Test Set score against the cross-validation score in standard errors (the winner's curse); the model against the benchmark; a validation curve; a permutation test in which the target is shuffled and the model refitted [29]; and a learning curve. The Rolling-Origin Evaluation [30] makes every date of the Test Set period a forecast origin and reports RMSE, MAE, MAPE and Theil's U by horizon against the random walk, the seasonal random walk and the training mean, with a Diebold–Mariano test at each horizon.

The AutoML view was designed for users who need a forecasting model without having to make the choices that lie behind it. A wizard asks for the data, the target, the horizon and the training effort. AutoML then prepares the data, selects predictors within every fit, trains and selects the models using the same engine as *Train All*, and produces a summary page that carries a verdict. Non-specialists are likely to act on this verdict, so its rules are fixed, and we give them in full in Table 4. The rules are conservative on purpose. For a model to be called *useful* it has to beat the benchmark on the Test Set with a lead that is statistically significant at the 5% level and large enough to be of practical interest, and the overfitting checks must not point the other way.

Table 4. Rules of the AutoML verdict (version 6.0).

Verdict	Numeric Target	Classes
Useful model	Lower Test Set RMSE than the benchmark, with $p < 0.05$ (Diebold–Mariano), and an error at least 10% below the benchmark's, and no strong sign of overfitting	Better than the base rate with $p < 0.05$ and Test Set AUC of at least 0.65, and no strong sign of overfitting
Weak model	Lower RMSE than the benchmark with $p \leq 0.10$, but $p \geq 0.05$ or an improvement below 10%; or the only strong overfitting sign is a Test Set score clearly below the cross-validation score	Better than the base rate with $p \leq 0.10$ and AUC of at least 0.55, but $p \geq 0.05$ or AUC below 0.65; or the same overfitting exception
No usable model	Not better than the benchmark on the Test Set; or better with $p > 0.10$ ("the lead could be chance"); or a strong sign of overfitting	Not better than the base rate; or $p > 0.10$; or AUC below 0.55; or a strong sign of overfitting

Benchmark: the stronger of the random walk and the training mean (numbers), the base rate or class shares (classes). A strong sign of overfitting is a strong verdict on the benchmark check, the Test Set against cross-validation check or the permutation test. The reasons for a verdict other than *useful* are written under it. At full effort, the summary also reports the Reality Check, the SPA test and the Model Confidence Set over all models tried.

Both reviewers asked how the verdict deals with a number of difficult cases, so we take them in turn. If a model beats the benchmark by a small margin and the difference is not statistically significant, the verdict is *no usable model* when $p > 0.10$ and *weak model* when p lies between 0.05 and 0.10; the p-value is given in the text in both cases. Uncertainty is conveyed through a 90% conformal range around each forecast, and the coverage of this range on the Test Set is reported. Instability across validation periods is picked up by the fold-spread check and by the comparison of the Test Set score with the cross-validation score. It remains true that the verdict is based on a single Test Set and cannot be more reliable than that sample. The summary reports how many rows were used, and we come back to this limitation in Section 4.3.

2.7. Causality

Prediction and causation are different questions, and the Causality tab keeps them apart under a warning that is always visible. Its cards use the data, predictors and model of the last run.

ML Granger causality. With dated data, each predictor is dropped in turn while the lags of the target itself are kept in both models. The model with the predictor and the model without it are then compared on the Test Set and on the cross-validation folds using the Clark–West and Diebold–Mariano tests. What is tested is Granger causality [31] in the out-of-sample sense of Ashley, Granger and Schmalensee [32], that is, predictive precedence and not causation. The reviewer raised three questions on the design. The first concerns *compensation*. A flexible model can make up for a dropped predictor, at least in part, through other predictors that are correlated with it. We consider this to be the intended behavior, because the question asked by the test is whether the predictor contains information over and above the others, and this is precisely how Granger causality is defined when the information set is multivariate. The second concerns *re-tuning*. The default is to tune the model without the predictor again, on the same grid and the same folds. In this way the restricted model is the best of its class, and the test is not tilted towards finding an effect. Fixed hyperparameters are available as a faster alternative, and the report records which option was used. The third is *multiple testing and p-hacking*. The *p*-values are Holm-adjusted over all the predictors tested. For a verdict of *yes*, the Holm-adjusted Test Set result must be significant and the cross-validation score must improve as well. All ticked predictors are tested together every time, so it is not possible to report one favorable predictor in isolation. As for the use of the model of the last run, the reason is that this model had been selected by cross-validation before any causal question was put to it. Of course, a user may still repeat the analysis with different settings until something significant turns up. The run history and the table of settings that is written into each report keep a record of what was done. This makes such a search visible to a reader, but the software has no way of preventing it.

Double ML. The effect of a treatment variable, D, on the target is estimated by double/debiased machine learning [33]. *Identification* requires unconfoundedness, meaning that all common causes of D and of the target are included among the controls, and in the interactive model it also requires overlap. Unconfoundedness is not testable, and the card says so. For *estimation*, the application uses the partially linear model and, when the treatment is binary, the interactive model for the average effect (ATE) and for the effect on the treated (ATT). Neyman-orthogonal scores are used throughout, with cross-fitting in 5 folds repeated 5 times (for time series, the folds are blocks of consecutive periods). The estimates are aggregated by taking the median over the repetitions, as is done in the DoubleML package [34]. *Overlap* is checked by reporting the range of the estimated propensity scores together with the share that falls below 0.05 or above 0.95. For the *sensitivity* to an omitted confounder, we follow Chernozhukov et al. [35]. The robustness value [36] is the share of the residual variation of both treatment and outcome that a hidden confounder would need to explain in order to bring the effect down to zero. Such a number is hard to judge without a scale. For this reason, each observed control is left out in turn and its own explanatory strength is computed, and the card then reports whether a hidden confounder as strong as the strongest observed control, or twice as strong, would be enough to remove the effect. The refutation checks are completed by a placebo treatment and by estimation on random subsamples.

2.8. Estimation Mode, Python Workbench and Other Components

Estimation mode fits models to the whole sample for explanation and inference: OLS with classical, robust [37], HAC [38] and clustered standard errors; logit, probit, Poisson, negative binomial, Tobit and quantile regression; IV/2SLS; pooled, fixed-effects and random-effects panel models and dynamic panel GMM; difference in differences, including the estimator of Callaway and Sant'Anna; ARIMA, ARDL with the bounds test [39], error-correction models, VAR, structural VAR, Johansen cointegration [40] and VECM; and GARCH [41]. A Patterns tab offers k-means, hierarchical clustering and principal components. The Python Workbench runs Python in the browser through Pyodide; the data, the split, the predictions and the settings of the last run are available as Python objects, and *Load script of this run* opens the exported script, which repeats the analysis with scikit-learn or statsmodels and prints Python's numbers next to the application's.

2.9. Numerical Validation: Design

In the original manuscript, the random learners were said to have been "checked for equal accuracy" against scikit-learn. The reviewers found this too weak and they were right. We repeated the validation in four stages, on versions 5.9 to 6.1.1, and this time the rules were fixed before any test was run. The complete report is provided as Supplementary Material S1; the design is summarized below.

- **Procedure.** The application was driven in a headless browser through its real controls (upload, settings, and run buttons), and results were read from its result objects with all decimals. The reference software received exactly the same rows, folds, scaled predictors and hyperparameters.
- **Measure.** Agreement is the log relative error, $\mathrm{LRE} = -\log10(|a - r|/|r|)$, the number of significant digits that the application's value, a, shares with the reference, r.
- **Rule 1, results with one correct answer** (estimators, tests, metrics, and deterministic learners): at least 6 significant digits.
- **Rule 2, random learners: equivalence.** Each learner is fitted 20 times in the application and 20 times in scikit-learn on the same training rows and scored on the same Test Set rows. The equivalence margins are ± 0.02 for accuracy, AUC, R^2, log loss and the Brier score, and $\pm 5\%$ of the scikit-learn mean for RMSE and MAE. Two one-sided tests [42] at the 5% level: equivalence is declared if the 90% confidence interval (Welch) of the difference in means lies inside the margin. Margins and rule were fixed after a pilot run with one set of seeds; the reported run used new seeds.
- **Rule 3, learners with ties.** Decision trees and boosting have several equally good solutions when two splits tie, and scikit-learn chooses among them at random. For these, the test is that every split of the application's trees attains the optimum found by an independent exhaustive search and that leaf values are correct.

Software-engineering evidence. The test harness consists of the benchmark of Section 3.1 and of a regression suite. Before each release, the suite runs the estimators, the ten learners (for two classes, several classes and regression), the metrics, cross-validation and the forecast tests again on fixed data sets, and each stored value is compared with that of the previous release. A difference is accepted only if it is announced in the changelog. We cannot report a line-coverage figure. The tests exercise the application through its interface and there are no unit tests of individual functions, which we acknowledge as a limitation. The behavior of the application in the edge cases mentioned by the reviewer is given in Table 5.

Table 5. Behavior in edge cases.

Case	Behavior
Missing values	Rows with a missing target are never imputed and are left out. Predictors: incomplete rows dropped (default) or mean/median imputation fitted on the training rows (inside cross-validation, on the fitting rows of each fold). A missing-values map shows where they are and how many rows the default removes
Constant columns	Reported in the import report, left out of scaling and of the model; descriptive statistics show a standard deviation of 0 and no skewness or kurtosis
Perfect separation of classes	Logistic regression carries a ridge of 10^{-6} on the standardized slopes, so a finite solution exists; the exported script uses the same objective
Class imbalance	A Target Classes card gives the class shares and warns below 20%; options: class weights, undersampling, oversampling, SMOTE; cut-off rules for the decision; balanced accuracy and PR-AUC reported
Validation fold with one class	AUC and related metrics of that fold are not computed and are left out of the mean, with a warning
Three to twenty classes	Native multi-class KNN, trees, forest, neural network and multinomial logit; one vs. rest for boosting and SVM; more than twenty classes refused with a message
Panel data	Unit column detected or chosen; lags, differences, the target shift and the random walk computed within units; split by date
Dates	ISO dates, Excel serials, year-only columns, year with month or quarter columns; irregular spacing and duplicates reported; a repeated year without a unit column is not treated as a time index
Tied tree splits	The first predictor in column order is taken (scikit-learn chooses at random); documented in the help and in every report
Neighboring values one binary digit apart	The cut is placed so that no side is empty (corrected in build 6.0.1 after the benchmark found one such case in 9915 rows)
A learner returns no prediction	Double ML stops with a message naming the model and row; metrics report the number of valid rows
Large samples with kernel methods	A warning before training when the Gram matrix would make the exact SVM solver slow

2.10. Design of the Simulation and of the Comparison with Other Tools

Simulation (Section 3.2). Four data-generating processes with 300 monthly observations, a target y and five predictors were simulated, with 100 replications each. A: y is a random walk and the predictors are independent persistent AR(1) series ($\rho = 0.95$) unrelated to y. B: y is a random walk with drift and the predictors are independent random walks, the setting of spurious regression. C: the change in y depends on last month's value of the first predictor, with a population R^2 of 5% for the change. D: the same with an R^2 of 30%. In A and B, the random walk is the best possible forecast and every declared success is false. Each data set was loaded into the application and run with the defaults: target $y(t+1)$, the five predictors and the current value of y as predictors, time-series split 80/20, five-fold expanding-window cross-validation, quick tuning, *Train All* (10 models). From the application's output, we recorded what four protocols would conclude; they are defined in Section 3.2.

Comparison (Section 3.3). Sixty series were drawn at random (seed 20261112) from the 30,570 monthly series of the M4 competition [43] with 150 to 500 observations, training

and test parts joined; to these we added the US unemployment rate (FRED series UNRATE, January 1948 to September 2025, retrieved on 12 November 2026). For each series and each horizon h = 1, 3, 6 and 12 months, the application was run with target $y(t + h)$; predictors $y(t)$, ..., $y(t - 11)$; time-series split 80/20; a purge of $h - 1$ rows; five-fold expanding-window cross-validation; and quick tuning. StatsForecast 2.1.1 [44] (AutoARIMA [45], AutoETS, AutoTheta [46], naive and seasonal naive, season length 12) and MLForecast 1.1.0 [47] (linear regression, random forest and LightGBM 4.7.0 [48] on lags 1 to 12, in levels and with first differencing of the target) were given the same information: each model was fitted once on the observations up to the first Test Set origin and then produced the h-step forecast from every Test Set origin without refitting (MLForecast recursively, its default; the application with a direct model for each horizon). All tools were therefore scored on identical origins and targets. Accuracy is measured by the MASE [9] (scaled by the in-sample seasonal naive error, as in M4), the sMAPE and the RMSE relative to the random walk. Two identities confirm the alignment: the naive forecast of StatsForecast equals the application's random-walk benchmark in every case, and MLForecast's linear regression equals the application's linear regression at h = 1. The application was run a second time with the target defined as the change over the next h months, $y(t + h) - y(t)$, and the twelve most recent monthly changes as predictors; its forecasts were converted back to levels and scored in the same way.

3. Results

3.1. Numerical Validation

The four stages of the benchmark are summarized in Table 6. All the tables, the data sets, the versions of the reference software and the conventions followed can be found in Supplementary Material S1. Where a result has one correct answer, the application agrees with the certified value or with the reference software to at least six significant digits, and in most cases to between 10 and 15. There is one exception, the Wampler 5 polynomial of NIST, where agreement is 5.9 digits; the reference package itself reaches 5.6 on this problem.

Table 6. Numerical validation of 4Cast Lab (versions 5.9 to 6.1.1): overview.

Stage	What Was Tested	Size	Reference	Result
1. Econometrics	OLS, IV, logit, probit, Poisson, negative binomial, Tobit, quantile regression, panel models, dynamic panel GMM, difference in differences, ARIMA, ARDL, ECM, VAR, VECM, structural VAR, GARCH, unit-root and residual tests	About 1930 values; 19 estimators and their tests	NIST certified values [49]; statsmodels [50]; linearmodels; arch; pydynpd; csdid	At least 6 significant digits everywhere, most 10 to 15; 4 issues found and corrected
2. Machine learning	All metrics, scaling, splits, cross-validation, the 10 learners for two classes, three classes and regression	370 metric values; 24,404 scaled values; 4110 tree splits; 2 × 20 fits per random learner	scikit-learn [51]; exhaustive search for tree splits	Metrics 14.7 digits or more; deterministic learners equal to scikit-learn; every tree split optimal; random learners in Table 7; 2 issues corrected

Table 6. *Cont.*

Stage	What Was Tested	Size	Reference	Result
3. Tests, causality, patterns	Diebold–Mariano, Clark–West, DeLong, McNemar, Holm, Reality Check, SPA, Model Confidence Set; Double ML and its sensitivity analysis; ML Granger causality; clustering; principal components	About 190 test cases; Double ML on 9915 households	dieboldmariano; statsmodels; arch; doubleml [34]; published formulas written out independently	8.9 to 15 digits; with the same bootstrap draws the Reality Check, SPA and MCS *p*-values are identical to the references; Double ML equal to doubleml to 13 digits; 3 issues corrected
4. Exported Python scripts	The script exported for each run, executed in the application's browser Python	37 scripts	The application's own results	All scripts run; exact methods reproduced to a relative difference of 10^{-10} or less; 4 issues in the scripts corrected

Digits: log relative error (Section 2.9). The benchmark found thirteen issues, nine in the application and four in the exported scripts. All of them are corrected in the current release and are listed with their effects in Section S7 of Supplementary Material S1, together with a fourteenth issue that was found by the simulation of Section 3.2.

Table 7. Random learners against scikit-learn: Test Set results over 20 fits each (selected rows; all 45 comparisons are in Section S4.4 of Supplementary Material S1).

Data	Learner	Metric	4Cast Lab: Mean (SD)	Scikit-Learn: Mean (SD)	Difference (90% CI)	Margin	Equivalent
Breast Cancer	Random forest	AUC	0.9960 (0.0009)	0.9956 (0.0008)	0.0004 (−0.0000 to 0.0009)	±0.02	Yes
Breast Cancer	Neural network	AUC	0.9995 (0.0006)	0.9943 (0.0046)	0.0052 (0.0034 to 0.0070)	±0.02	Yes
Breast Cancer	Neural network	Accuracy	0.9865 (0.0104)	0.9730 (0.0162)	0.0135 (0.0062 to 0.0208)	±0.02	No (higher)
Breast Cancer	Stacking	AUC	0.9922 (0.0010)	0.9922 (0.0010)	0.0000 (−0.0005 to 0.0006)	±0.02	Yes
Iris	Random forest	Accuracy	0.9367 (0.0103)	0.9333 (0.0000)	0.0033 (−0.0006 to 0.0073)	±0.02	Yes
Iris	Neural network	Accuracy	0.9867 (0.0199)	0.9900 (0.0190)	−0.0033 (−0.0137 to 0.0071)	±0.02	Yes
Diabetes	Random forest	RMSE	59.96 (0.55)	60.08 (0.69)	−0.12 (−0.45 to 0.22)	±3.00	Yes
Diabetes	Neural network	RMSE	58.38 (1.17)	59.03 (1.58)	−0.65 (−1.40 to 0.09)	±2.95	Yes
Diabetes	Neural network	R^2	0.4130 (0.0236)	0.3996 (0.0322)	0.0134 (−0.0017 to 0.0285)	±0.02	No (interval too wide)
Diabetes	Stacking	RMSE	55.15 (0.12)	55.16 (0.08)	−0.01 (−0.07 to 0.04)	±2.76	Yes

Application seeds 2001 to 2020, scikit-learn random states 100 to 119, same training and Test Set rows. Equivalent: the 90% Welch confidence interval of the difference in means lies inside the margin (two one-sided tests at 5%). RMSE margins are 5% of the scikit-learn mean.

Random learners. The repeated-run evidence requested by the reviewers is given in Table 7. There were 15 pre-specified comparisons for each learner (three data sets with three or four metrics each). The random forest is equivalent to that of scikit-learn in all 15, the neural network in 12 and stacking in 11. In none of the comparisons where equivalence could not be declared was the application significantly worse. Either the confidence interval crosses the margin on the side that favors the application, as happens with the neural network on the Breast Cancer data, or the interval is wide because the Test Set is small. Accordingly, in what follows the random forest is described as *equivalent* within the stated margins, whereas for the neural network and for stacking we only say that they are *comparable* to the scikit-learn models with the same settings.

Trees and boosting. In the application, the decision tree and gradient boosting involve no randomness. All 41 splits of the decision trees and all 4069 splits of the boosted trees reach the optimum found by an exhaustive search, and the leaf values agree to 13 digits or more. For classes, gradient boosting was changed in version 6.0. It used mean-residual leaves before, and the benchmark showed a measurable difference in the log loss on the Breast Cancer data (0.195 against 0.108); it now uses the Newton step, as scikit-learn does. When ties are broken in the same way, the training probabilities are equal to those of scikit-learn at each of the 100 stages. For two classes and for regression, the Test Set metrics fall within the range that scikit-learn gives over 20 random states. With three classes, the Test Set results may differ, depending on which of several equally good splits is chosen. The application always takes the first predictor in column order. This is a difference by design, and it is documented in the help and in every report. The support vector machines use the SMO solver at the default tolerance of LIBSVM. They agree with the exact optimum to about three digits, which is also what scikit-learn achieves at that tolerance, and all 948 Test Set decisions have the same sign as the exact optimum.

3.2. Does the Protocol Reduce False Claims of Forecasting Success? A Simulation

The second reviewer asked whether there is any evidence that the benchmark-first workflow reduces erroneous model selection. To answer this we used the application itself. The 400 simulated data sets described in Section 2.10 (100 for each design) were loaded into 4Cast Lab one by one and run with *Train All* at the default settings, in which case the Test Set has 60 months. For every run, we then recorded what would be concluded under each of the following five reporting rules.

- **High R^2.** The model selected by cross-validation has a Test Set R^2 of at least 0.5. This is the R^2 level that many applied papers report as evidence of a good forecasting model.
- **Best on the Test Set.** The ten models are compared on the Test Set, and the best one has a lower RMSE than the random walk. This is model selection on the test sample.
- **Selected by cross-validation, no test.** The model selected on the training rows has a lower Test Set RMSE than the random walk, with no test of the difference.
- **Benchmark first.** The model selected on the training rows has a lower Test Set RMSE than the benchmark and the Diebold–Mariano test rejects equal accuracy at 5%. This is the application's verdict.
- **SPA test.** Hansen's test over all ten models rejects at 5% the hypothesis that no model beats the benchmark.

The results are in Table 8 and Figure 1. A high R^2 turns out to be no evidence at all. When the target is a random walk, the selected model has a Test Set R^2 above 0.5 in more than 90% of the data sets, with a median of 0.84. The reason is well known: the level of a persistent series is easy to track, but it cannot be forecast better than by its last value. Comparing the models on the Test Set and reporting the best one leads to a false success in 32% of the random-walk data sets. Even when the model is selected properly, on the

training rows, it still beats the random walk by chance in 21 to 25% of them if no test is applied. Under the benchmark-first verdict a false success is declared in 0 to 1% of the random-walk data sets. The nominal level of the test is 5%, so the test is conservative here, which is what one expects when the models nest the benchmark [23].

Table 8. Simulation: share of replications in which a forecasting success is declared (%).

Rule	A: Random Walk, Unrelated Predictors	B: Random Walk, Random-Walk Predictors	C: Weak Signal	D: Strong Signal
Test Set R^2 of at least 0.5	92	95	100	99
Best model on the Test Set beats the random walk	32	32	49	87
Model selected by CV beats the random walk, no test	25	21	41	81
Benchmark first (selected by CV, Diebold–Mariano $p < 0.05$)	0	1	5	36
The same with the Holm-adjusted p-value of the comparison card	0	1	3	31
Clark–West test of the selected model, $p < 0.05$	7	13	23	85
White's Reality Check, $p < 0.05$	0	0	0	0
SPA test, $p < 0.05$	1	5	9	43
Median Test Set R^2 of the selected model	0.84	0.84	0.92	0.97
Median RMSE of the selected model relative to the random walk	1.04	1.05	1.01	0.92

100 replications per design; the Monte Carlo standard error of a share is at most 5 percentage points. In every replication, the application chose the random walk, not the training mean, as the benchmark. C and D: population R^2 of 5% and 30% for the monthly change.

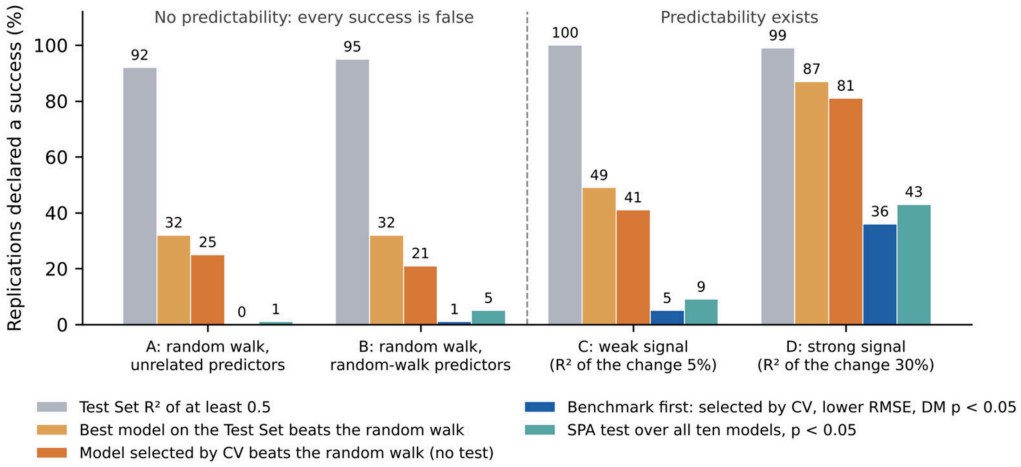


Figure 1. Share of simulated data sets in which each reporting rule declares a forecasting success (100 replications per design, 300 observations, Test Set of 60). In designs A and B, the target is a random walk, and no model can beat the benchmark.

This protection comes at a cost, which the simulation also allows us to measure. With 60 observations in the Test Set, the verdict detects a real but weak signal (design C) in just 5% of the cases, and a strong signal (design D) in 36%. So, when a user reads *no usable model*, this should be understood as *not shown on this sample*; the wording of the verdict ("the lead could be chance") was chosen with this in mind. The application also reports the Clark–West test next to the Diebold–Mariano test, as it was designed for models that nest

the benchmark. It is much more powerful (85% in design D). On the other hand, it rejects too often when the predictors are random walks themselves (13% in design B), and for this reason it is shown only as supporting evidence and plays no part in the verdict. The Reality Check did not reject in any replication, which agrees with what is known about its loss of power when poor models are included in the comparison [19]. Among the rules that keep false successes under control, the SPA test has the correct size and the highest power.

An error found by the simulation. The simulation also revealed an error in the application. Up to build 6.0.2, the SPA p-value was wrong in one particular situation. When none of the models has a lower Test Set loss than the benchmark, the statistic is equal to zero and the p-value should be 1. The application, however, counted the bootstrap draws that were strictly above the statistic, and so it reported a value close to 0. In the two random-walk designs, the SPA line of the output rejected in 22 and 32% of the replications as a result. The earlier validation had not detected this, since the p-values had been compared with the reference implementations only in cases where at least one model was ahead of the benchmark. The error has been corrected in build 6.0.3, where $p = 1$ if no model is ahead, and the SPA figures in Table 8 are those of the corrected test. A sample of the simulated data sets was run again on build 6.0.3 and on version 6.1.1; all results were identical to those of the earlier build apart from the corrected SPA p-values. The Reality Check, the Model Confidence Set and the verdict itself were not affected. We mention the episode because, in our view, it shows that a simulation under a known truth is a necessary part of validating forecasting software, in addition to comparisons with reference values.

3.3. Comparison with StatsForecast and MLForecast on M4 Monthly Series

The comparison of the application with StatsForecast and MLForecast on the M4 sample, following the design of Section 2.10, is given in Table 9 and Figure 2. One of the 60 series had to be left out for all methods (see the note to Table 9). The remaining 59 series have a median Test Set of 62 months and consist of 17 macroeconomic, 13 financial, 11 microeconomic, 11 demographic and seven industry series.

Table 9. Forecast accuracy on 59 M4 monthly series, by horizon h (months). Lower is better.

Method	MASE h = 1	h = 3	h = 6	h = 12	Rel. RMSE h = 1	h = 3	h = 6	h = 12
Benchmarks								
Random walk	0.83	1.30	1.32	1.37	1.00	1.00	1.00	1.00
Seasonal naive	1.24	1.25	1.28	1.37	1.44	1.24	1.09	1.00
StatsForecast								
AutoARIMA	0.51	0.73	0.94	1.27	0.77	0.79	0.85	0.95
AutoETS	0.52	0.75	0.96	1.27	0.81	0.83	0.89	0.98
AutoTheta	0.55	0.77	0.97	1.28	0.79	0.81	0.88	0.96
MLForecast, target in levels								
Linear regression	0.59	0.85	1.12	1.53	0.88	0.87	0.90	1.00
Random forest	1.49	1.68	1.87	2.31	1.29	1.37	1.23	1.42
LightGBM	1.83	2.01	2.21	2.68	1.52	1.46	1.39	1.55
MLForecast, target differenced								
Linear regression	0.53	0.74	0.94	1.22	0.80	0.79	0.82	0.91
Random forest	0.58	0.84	1.08	1.49	0.87	0.86	0.95	1.09
LightGBM	0.61	0.90	1.11	1.47	0.90	0.90	0.97	1.07

Table 9. *Cont.*

Method	MASE h = 1	h = 3	h = 6	h = 12	Rel. RMSE h = 1	h = 3	h = 6	h = 12
4Cast Lab, target in levels								
Linear regression	0.59	0.82	1.19	1.53	0.88	0.83	0.89	0.99
Random forest	1.50	1.87	2.14	2.71	1.31	1.24	1.21	1.79
Gradient boosting	1.55	1.93	2.15	2.73	1.46	1.36	1.28	1.80
Model selected by CV (of 10)	0.62	0.85	1.39	2.21	0.87	0.86	0.90	1.20
4Cast Lab, target defined as the change over h months								
Linear regression	0.53	0.75	0.97	1.28	0.80	0.80	0.83	0.90
Random forest	0.57	0.85	1.04	1.30	0.88	0.87	0.89	0.99
Gradient boosting	0.60	0.87	1.07	1.39	0.90	0.89	0.93	1.07
Model selected by CV (of 10)	0.53	0.76	0.92	1.31	0.80	0.83	0.87	0.94

MASE: mean over the series of the mean absolute error scaled by the in-sample seasonal naive error. Rel. RMSE: median over the series of the RMSE divided by the RMSE of the random walk. All methods are scored on the same forecast origins. Series M38997, in which two thirds of the monthly changes are exactly zero, was left out for all methods because the application returned no naive benchmark for its differenced target at h = 1; this edge case is recorded as an open issue.

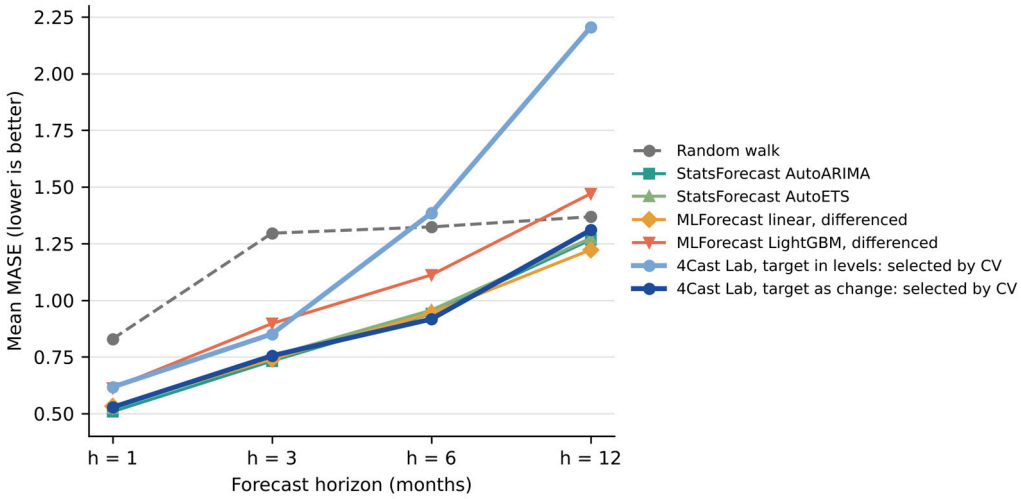


Figure 2. Mean MASE by forecast horizon on 59 M4 monthly series, selected methods of Table 9.

Several things can be read from Table 9. To begin with, when the comparison is like for like, the application gives the same results as the library. With the target in levels and the same twelve lags, the linear regression of the application and that of MLForecast have the same accuracy at h = 1 (MASE: 0.59; the forecasts are in fact identical). The tree ensembles of the two tools also fail in the same manner (MASE: 1.50 and 1.55 against 1.49 and 1.83), as trees are not able to extrapolate a trending series. At the longer horizons, the two tools follow different strategies, direct in the application and recursive in MLForecast, but the results remain close. With the target in levels, however, the application is clearly less accurate than the statistical models. The model selected by cross-validation has an MASE of 0.62 at h = 1, while AutoARIMA has an MASE of 0.51, and at h = 12 it does worse than the random walk (2.21 against 1.37). What happens is that cross-validation sometimes selects a flexible learner, which then meets values outside the range it was trained on. We do not wish to play this down: with the default settings and trending univariate series, 4Cast Lab cannot replace a specialized forecasting library. Most of the gap disappears once the target is defined as a change. If the target is the change over the next h months, a variable the

user can build in the Variable Construction card, the MASE of the selected model becomes 0.53, 0.76, 0.92 and 1.31 at the four horizons, compared with 0.51, 0.73, 0.94 and 1.27 for AutoARIMA. The selected model then beats the random walk in 88, 81, 69 and 61% of the series (for AutoARIMA, the figures are 83, 83, 81 and 58%). The results of MLForecast with and without differencing are explained in the same way. The application, though, does not make this choice on behalf of the user. StatsForecast applies differencing and models seasonality automatically, whereas 4Cast Lab leaves the transformation to the user and only shows unit-root tests on the Data tab. Automatic differencing of the target is the most useful addition suggested by this exercise, and we plan to include it.

What the user is told. The libraries return forecasts and, if requested, cross-validated errors. The application gives a verdict in addition, and Table 10 reports what this verdict was for the same 59 series. With the target in levels and h = 1, the model selected by cross-validation had a lower Test Set RMSE than the benchmark in 69% of the series. The difference was statistically significant in only 44%. In 25% of the series, the lead could not be distinguished from chance, and in a further 5% the selected model was significantly worse than the benchmark. At h = 12, the selected model beat the benchmark in 36% of the series, significantly so in 8%. In each case, the user is told in words which of these situations applies. We believe this is what *benchmark first* offers in practice. The forecasts do not become more accurate because of it, but the user is much less likely to draw the wrong conclusion about how accurate they are.

Table 10. What 4Cast Lab reported on the 59 M4 series (% of series).

Outcome	h = 1	h = 3	h = 6	h = 12
Target in levels				
Benchmark chosen on the training rows is the random walk (otherwise the training mean)	90	90	88	90
Best of the ten models on the Test Set has a lower RMSE than the benchmark	78	83	85	68
Model selected by CV has a lower RMSE than the benchmark	69	69	61	36
and the difference is significant (Diebold–Mariano, 5%)	44	47	39	8
and it remains significant after the Holm adjustment	39	37	27	5
Model selected by CV has a lower RMSE, not significant	25	22	22	27
Model selected by CV is significantly worse than the benchmark	5	7	8	17
SPA test over the ten models rejects at 5%	47	56	51	29
Reality Check over the ten models rejects at 5%	20	17	14	3
Mean number of models in the 90% Model Confidence Set (of 10)	4.6	4.6	4.3	3.9
Target defined as the change over h months				
Model selected by CV has a lower RMSE than the benchmark	86	83	73	41
and the difference is significant (Diebold–Mariano, 5%)	51	47	39	15

SPA: corrected *p*-value (Section 3.2). For the change target, the benchmark chosen on the training rows was the training mean of the change, that is, a random walk with drift, in all but one or two series.

3.4. A Real Example with a Structural Break: The US Unemployment Rate

In the original manuscript, the worked examples were based on the simulated teaching data sets. We have replaced them with a public series that has the features the reviewers asked for, namely, persistence, non-stationarity and a break. The monthly unemployment rate of the US [52] is available from January 1948 to September 2025 (933 observations). With the time-ordered split, the Test Set contains the forecast origins from April 2010 to August 2025, which is 185 months at h = 1. The models are thus tuned and selected on data up to early 2010, and they are evaluated on a period that includes the jump from 4.4% to 14.8% in April 2020. The Test Set RMSE of the application and of the two libraries is reported in Table 11, and the output of the application at h = 1 is shown in Figures 3 and 4.

Table 11. US unemployment rate: Test Set RMSE (percentage points) by horizon.

Method	h = 1	h = 3	h = 6	h = 12
Random walk (the benchmark chosen by 4Cast Lab)	0.824	1.375	1.711	2.000
Seasonal naive	1.984	1.989	1.989	2.000
StatsForecast AutoARIMA	0.968	1.870	2.742	2.805
StatsForecast AutoETS	0.862	1.573	2.123	2.592
StatsForecast AutoTheta	0.825	1.375	1.708	2.004
MLForecast linear regression	0.908	1.671	2.274	2.378
MLForecast random forest	0.817	1.277	1.535	1.800
MLForecast LightGBM	0.829	1.238	1.501	1.841
MLForecast linear regression, differenced	0.912	1.700	2.392	2.683
MLForecast random forest, differenced	0.816	1.371	1.690	1.920
MLForecast LightGBM, differenced	0.823	1.376	1.720	1.988
4Cast Lab linear regression	0.908	1.670	2.312	2.323
4Cast Lab random forest	0.816	1.252	1.518	1.678
4Cast Lab gradient boosting	0.835	1.230	1.469	1.614
4Cast Lab, model selected by CV	0.901	1.654	2.312	2.323
Which model	SVR, linear	SVR, linear	Linear regression	Linear regression
Diebold–Mariano *p*-value against the benchmark	0.158	0.293	0.290	0.350
SPA *p*-value over the ten models	0.766	0.317	0.245	0.116
Reality Check *p*-value	0.898	0.714	0.732	0.657
Models in the 90% Model Confidence Set	10 of 10	10 of 10	10 of 10	10 of 10

Test Set: 185 forecast origins at h = 1 (April 2010 to August 2025), 182 at h = 12. Target in levels, predictors y(t) to y(t − 11), purge of h − 1 months.

At h = 1, none of the methods improves on the random walk by more than 1%, while AutoETS and AutoARIMA are 5 and 17% worse; at h = 12, AutoARIMA is 40% worse. The Test Set errors are dominated by the break, which a model fitted on earlier data has no way of anticipating. Cross-validation on the training period selected linear models at all four horizons, and on the Test Set these turned out to be 9 to 35% worse than the random walk. The application reports the result as *tied with the benchmark* at every horizon (Figure 3). The Diebold–Mariano *p*-values lie between 0.16 and 0.35, neither the SPA test nor the Reality Check rejects, and all ten models stay in the Model Confidence Set. The user is therefore given no model to recommend, and with these data we think this is the right conclusion. There is one more observation to make. For h = 3 to 12, the tree ensembles did better than the random walk on the Test Set, by as much as 19% in the case of gradient boosting at h = 12, and this holds for the application and for MLForecast alike. Someone who had compared the ten models on the Test Set would probably report this as a finding. The application does not, for two reasons: these models were not the ones selected on the training rows, and the SPA test, which allows for the search over the ten models, has a *p*-value of 0.12 at h = 12. That tree ensembles may forecast unemployment better at long horizons is a reasonable hypothesis to examine with new data. It is not something that this sample establishes.

3.5. What Each Tool Reports to Its User

In Table 12, we place 4Cast Lab next to the four tools that the reviewers mentioned. Only StatsForecast and MLForecast were actually run for this paper. PyCaret [53] and AutoGluon-TimeSeries [54] were not, and what the table says about them is based on their documentation. The table should not be read as a ranking. The libraries scale far better and include model families that are absent from 4Cast Lab, and any of the checks listed can be programmed on top of them. The difference lies in what the analyst obtains without having to write that code.

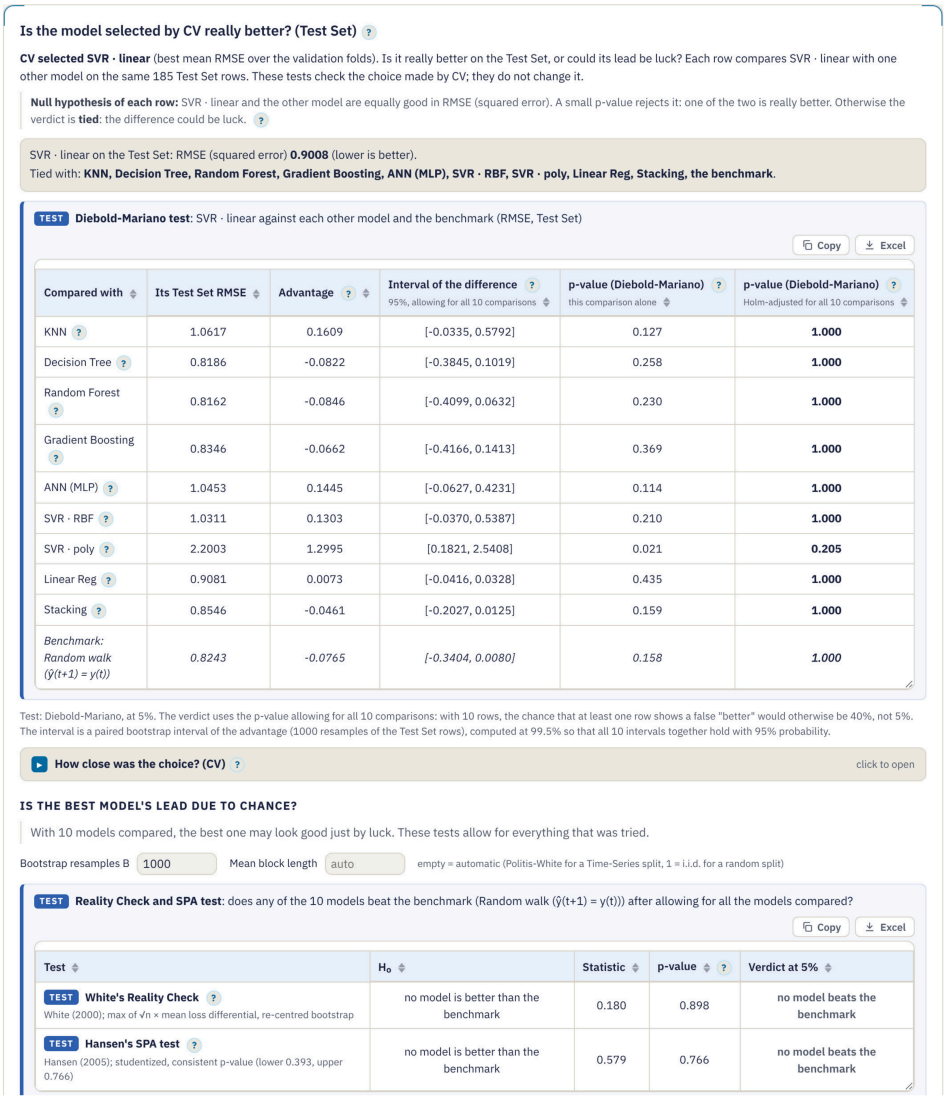


Figure 3. 4Cast Lab 6.0, Model tab, US unemployment rate at h = 1: the card *Is the model selected by CV really better?* with the Diebold–Mariano comparisons of the selected model with every other model and with the benchmark, and the Reality Check and SPA tests over all ten models.

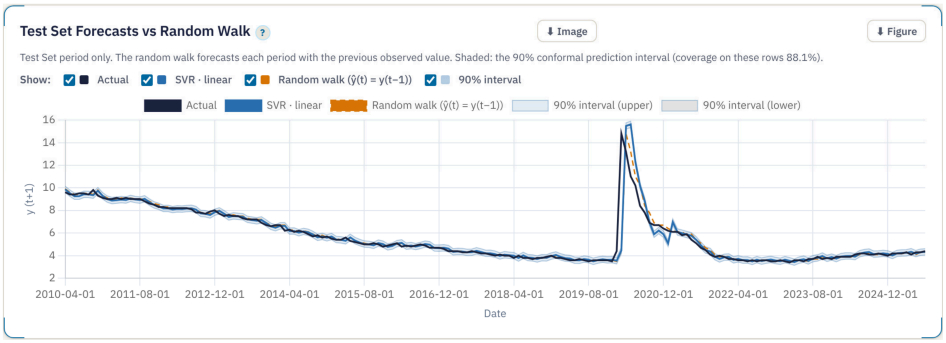


Figure 4. 4Cast Lab 6.0, US unemployment rate at h = 1: Test Set forecasts of the selected model and of the random walk, with the 90% conformal interval (coverage on the Test Set: 88.1%). The figure is a screenshot of the application: the dark line is the actual series, the blue line the forecast of the selected model and the orange dashed line the random walk; the light blue band is the 90% interval. The buttons with arrows above the chart export the image.

Table 12. 4Cast Lab and four established forecasting tools: what is provided by default.

	4Cast Lab 6.0	StatsForecast	MLForecast	PyCaret (Time Series)	AutoGluon-TimeSeries
Interface	Graphical, in the browser; no code	Python library	Python library	Python library, low-code	Python library
Model families	Ten machine learning configurations on lagged predictors; econometric models in a separate mode	Statistical models (ARIMA, ETS, Theta and others)	Any scikit-learn-type regressor on lag features	Statistical and machine learning models	Statistical, machine learning, deep learning and pretrained models, with ensembling
Time-ordered validation	Yes; expanding or sliding window, purge	Yes	Yes	Yes	Yes
Automatic differencing and seasonality	No; built by the user	Yes	By target transformations	Seasonal period detected automatically; differencing within some models	Yes, within the models
Naive benchmark	Reported with every result; chosen on the training rows	Available as models; added by the user	Added by the user	Naive models in the model library that are compared	A seasonal naive baseline among the models trained by default
Formal test of the selected model against the benchmark	Diebold–Mariano and Clark–West	Not built in	Not built in	Not built in	Not built in
Allowance for the number of models compared	Holm, Reality Check, SPA, Model Confidence Set	Not built in	Not built in	Not built in	Not built in
Overfitting checks and a verdict in words	Yes (Section 2.6)	No	No	No	No
Prediction intervals	Conformal	Model-based and conformal	Conformal	Yes; coverage set by the user	Quantile forecasts
Scale	One series or a panel; up to tens of thousands of rows	Millions of series	Millions of series	Single machine	Large collections of series

"Not built in" means that we found no such function in the online documentation consulted in November 2026 (StatsForecast and MLForecast at nixtlaverse.nixtla.io, PyCaret 3.0.4, AutoGluon 1.6.1); the tests can be computed by the user from the forecasts that every tool returns.

4. Discussion

4.1. What the Evidence Shows

The first reviewer wanted to know whether the implementation can be trusted, and the second what the paper contributes beyond a description of software. We think the evidence of Section 3 gives an answer to both questions. It also shows where the claims of the paper stop.

On the first question, the implementation appears to be sound. In about 2500 comparisons with certified values and reference software, the results that have one correct answer agree to at least six digits. The random forest is equivalent to that of scikit-learn within

margins that were fixed in advance, and the neural network and stacking are comparable. Thirteen defects were found in the process. The simulation of Section 3.2 then found a fourteenth, which the comparison with reference values had missed. We regard this list of defects as part of the result, since it indicates what the tests were able to detect.

On the second question, the contribution is the benchmark-first protocol as a software default, together with evidence on what it achieves. In the cases where there is nothing to forecast, the reporting rules that the protocol is meant to replace declare a success in 21 to 95% of the simulated data sets, against 0 to 1% for the protocol (Table 8). With real series, about a quarter of the selected models had a lower error than the benchmark, although the difference was not statistically significant (Table 10). In a table that shows only errors, such cases would be presented as successes. It should be clear that the protocol does not improve the forecasts themselves. What changes is what is reported about them, and this is done for users who would not have programmed the tests on their own.

This protection has a cost. When the Test Set has about 60 observations, even a strong signal is detected by the verdict in only about one third of the cases. The design prefers the error of saying *not shown* to the error of saying *shown*. For software that will be used by non-specialists we believe this is the better side to err on. Users should nevertheless be aware of it, and for this reason the summary reports the size of the Test Set and the *p*-value along with every verdict.

As for the forecasting engine, it is adequate but it is not competitive. Given the same learner and the same inputs, the application reproduces MLForecast. Compared with the automatic statistical models, it is clearly weaker when a trending target is modeled in levels, and it comes close to them when the target is defined as a change. If the only aim is the most accurate forecast for a large number of univariate series, the specialized libraries are the better choice. 4Cast Lab has a different user in mind. This is someone with a target, a set of candidate predictors and the question whether anything in these data beats the naive forecast, who needs an answer that can be defended.

4.2. Uses

In research and professional work, a model built in the application can be described, checked and reproduced with the help of the Expert view, nested cross-validation, the pairwise and data-snooping tests, the overfitting check, the reports and the exported Python script. Users from outside statistics and economics can turn to the AutoML view, which returns a tested model, a verdict based on the fixed rules of Table 4 and forecasts for new data. We also use the application in our own courses, where the Core view and the built-in examples make it convenient to demonstrate the protocol. We do not make any claim regarding learning outcomes. The application has not been evaluated in a controlled teaching study, and the statements about teaching that appeared in the original manuscript have been removed. It should also be said that the built-in example data sets are simulated. Their purpose is to let any user repeat an analysis whose properties are known, and they provide no evidence about forecasting performance. For this reason, the worked examples of the original manuscript, among them a Test Set AUC of 1.000 obtained on 51 simulated rows, have given way to the real-data results of Sections 3.3–3.5.

4.3. Limitations

The validation is not complete, and what it leaves out is listed in Section S9 of Supplementary Material S1. The tuning grids, nested cross-validation, probability calibration, the interpretability tools, variable selection and AutoML were all exercised, but no external reference exists for them. The tests are run through the interface, and no measure of line

coverage is available. In some of the pre-specified comparisons, the neural network and stacking are comparable, but not equivalent, to their scikit-learn counterparts (Table 7).

The protocol has limitations of its own. (i) The verdict depends on a single Test Set. When the Test Set is short, the Diebold–Mariano test has low power, and in such cases *no usable model* will often mean that the evidence is not sufficient; Section 3.2 gives an idea of the magnitudes. A longer Test Set and the Rolling-Origin Evaluation help here, and the summary always reports the number of rows used. (ii) The Reality Check and the SPA test are computed on the Test Set for models that were tuned on the training rows. They protect against the search over the ten models compared in a run. They offer no protection against a search over data sets, targets or settings that a user may carry out across runs. (iii) The purge for overlapping multi-step targets has to be set by the user and is not an automatic default. Fold-specific preprocessing and selection (Section 2.4) remove the dependence between fitting and validation rows that arises from estimated quantities, but not the one that arises from overlapping targets, for which the purge is needed. (iv) Forecasts are produced by direct models on lagged predictors. Forecasting mode has no automatic differencing, no seasonal adjustment and no exponential smoothing, and the cost of this on trending and seasonal series was seen in Section 3.3. (v) As the application runs in a browser engine, it is suited to data sets of thousands to tens of thousands of rows, and kernel methods are slower still. (vi) The causal estimates can be given a causal reading only under unconfoundedness, an assumption that no software is able to test.

5. Conclusions

With 4Cast Lab, a benchmark-first forecasting protocol can be applied without any programming. The protocol consists of time-ordered validation, selection within the training period, a single use of the Test Set, a naive benchmark accompanied by a formal test, and an adjustment for the number of models tried. In this revision, we have added the evidence that was missing from the original manuscript. The numerical core of the application agrees with certified values and with reference software, under rules that were fixed in advance. In the simulation, the protocol removes nearly all false claims of forecasting success, and the cost of this in terms of power has been measured and reported. On public data, the forecasts are the same as those of the corresponding learners in MLForecast, and they come close to those of the automatic statistical models when the target is differenced. The verdicts of the application distinguish the leads over the benchmark that have statistical support from those that do not. Following the second round of review, imputation, outlier bounds and variable selection are now carried out within each fold in the manual workflow as well, so that no data-dependent step of the pipeline uses the validation rows. In future versions, we intend to add automatic differencing of the target, an automatic purge for multi-step targets, and unit tests of individual functions.

Supplementary Materials: The following supporting information can be downloaded at https://www.mdpi.com/article/10.3390/forecast8060115/s1: Supplementary Material S1: 4Cast Lab Numerical Validation Report (design, data, reference software and all results of the four-stage benchmark), with an addendum on version 6.1.1 (preprocessing and variable selection inside the folds). Supplementary Material S2: replication files for Sections 3.2–3.5 (the simulated data sets, the M4 sample and the unemployment series as loaded into the application; the scripts that drive the application and the reference libraries; and the result files).

Funding: This research received no external funding.

Data Availability Statement: 4Cast Lab is free software under the MIT license. The source code, the releases, the User Manual and the Quick Start Guide are available at https://github.com/PerryGogas/4Cast-Lab (accessed on 22 November 2026); the application can be used and installed

from https://perrygogas.github.io/4Cast-Lab/ (accessed on 22 November 2026); each release is archived on Zenodo (https://doi.org/10.5281/zenodo.22818865, which always resolves to the latest version) [55]. The M4 data are available from https://github.com/Mcompetitions/M4-methods (accessed on 22 November 2026) and the unemployment rate from FRED [52]. The data sets of the numerical validation are public and are listed in Supplementary Material S1. All other data and scripts are in Supplementary Material S2.

Acknowledgments: The author thanks Alexandros Grigorian and Panagiotis E. Goumenidis, from the E|M|E|F Lab, Democritus University of Thrace, and Athanasia Dimitriadou, Senior Lecturer at the University of Lancashire, for testing successive versions of the application and for reporting errors and corrections. Other students of the E|M|E|F Lab tested the application on their own data sets. During the preparation of this work, the author used Claude (Anthropic) to help draft and edit the text; to write and run the validation, simulation and comparison scripts; and to identify and correct software defects. The author reviewed and edited all content, verified the results, and takes full responsibility for the publication.

Conflicts of Interest: The author is the developer of 4Cast Lab, which is distributed free of charge under an open-source license. The author declares no other conflicts of interest.

References

1. Hewamalage, H.; Ackermann, K.; Bergmeir, C. Forecast evaluation for data scientists: Common pitfalls and best practices. *Data Min. Knowl. Discov.* **2023**, *37*, 788–832. [CrossRef] [PubMed]
2. Makridakis, S.; Spiliotis, E.; Assimakopoulos, V. Statistical and machine learning forecasting methods: Concerns and ways forward. *PLoS ONE* **2018**, *13*, e0194889. [CrossRef] [PubMed]
3. Kapoor, S.; Narayanan, A. Leakage and the reproducibility crisis in machine-learning-based science. *Patterns* **2023**, *4*, 100804. [CrossRef] [PubMed]
4. Breiman, L. Random forests. *Mach. Learn.* **2001**, *45*, 5–32. [CrossRef]
5. Friedman, J.H. Greedy function approximation: A gradient boosting machine. *Ann. Stat.* **2001**, *29*, 1189–1232. [CrossRef]
6. Cortes, C.; Vapnik, V. Support-vector networks. *Mach. Learn.* **1995**, *20*, 273–297. [CrossRef]
7. Chang, C.-C.; Lin, C.-J. LIBSVM: A library for support vector machines. *ACM Trans. Intell. Syst. Technol.* **2011**, *2*, 27.
8. Bergstra, J.; Bengio, Y. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.* **2012**, *13*, 281–305.
9. Hyndman, R.J.; Koehler, A.B. Another look at measures of forecast accuracy. *Int. J. Forecast.* **2006**, *22*, 679–688. [CrossRef]
10. Vovk, V.; Gammerman, A.; Shafer, G. *Algorithmic Learning in a Random World*; Springer: New York, NY, USA, 2005.
11. Fisher, A.; Rudin, C.; Dominici, F. All models are wrong, but many are useful: Learning a variable's importance by studying an entire class of prediction models simultaneously. *J. Mach. Learn. Res.* **2019**, *20*, 177. [PubMed]
12. Štrumbelj, E.; Kononenko, I. An efficient explanation of individual classifications using game theory. *J. Mach. Learn. Res.* **2010**, *11*, 1–18.
13. Kaufman, S.; Rosset, S.; Perlich, C.; Stitelman, O. Leakage in data mining: Formulation, detection, and avoidance. *ACM Trans. Knowl. Discov. Data* **2012**, *6*, 15.
14. Bergmeir, C.; Benítez, J.M. On the use of cross-validation for time series predictor evaluation. *Inf. Sci.* **2012**, *191*, 192–213. [CrossRef]
15. Ambroise, C.; McLachlan, G.J. Selection bias in gene extraction on the basis of microarray gene-expression data. *Proc. Natl. Acad. Sci. USA* **2002**, *99*, 6562–6566. [CrossRef] [PubMed]
16. López de Prado, M. *Advances in Financial Machine Learning*; Wiley: Hoboken, NJ, USA, 2018.
17. Holm, S. A simple sequentially rejective multiple test procedure. *Scand. J. Stat.* **1979**, *6*, 65–70.
18. White, H. A reality check for data snooping. *Econometrica* **2000**, *68*, 1097–1126. [CrossRef]
19. Hansen, P.R. A test for superior predictive ability. *J. Bus. Econ. Stat.* **2005**, *23*, 365–380. [CrossRef]
20. Hansen, P.R.; Lunde, A.; Nason, J.M. The model confidence set. *Econometrica* **2011**, *79*, 453–497. [CrossRef]
21. Diebold, F.X.; Mariano, R.S. Comparing predictive accuracy. *J. Bus. Econ. Stat.* **1995**, *13*, 253–263. [CrossRef]
22. Harvey, D.; Leybourne, S.; Newbold, P. Testing the equality of prediction mean squared errors. *Int. J. Forecast.* **1997**, *13*, 281–291. [CrossRef]
23. Clark, T.E.; West, K.D. Approximately normal tests for equal predictive accuracy in nested models. *J. Econom.* **2007**, *138*, 291–311. [CrossRef]
24. Pesaran, M.H.; Timmermann, A. A simple nonparametric test of predictive performance. *J. Bus. Econ. Stat.* **1992**, *10*, 461–465. [CrossRef]

25. DeLong, E.R.; DeLong, D.M.; Clarke-Pearson, D.L. Comparing the areas under two or more correlated receiver operating characteristic curves: A nonparametric approach. *Biometrics* **1988**, *44*, 837–845. [CrossRef] [PubMed]
26. McNemar, Q. Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika* **1947**, *12*, 153–157. [CrossRef] [PubMed]
27. Politis, D.N.; Romano, J.P. The stationary bootstrap. *J. Am. Stat. Assoc.* **1994**, *89*, 1303–1313. [CrossRef]
28. Politis, D.N.; White, H. Automatic block-length selection for the dependent bootstrap. *Econom. Rev.* **2004**, *23*, 53–70. [CrossRef]
29. Ojala, M.; Garriga, G.C. Permutation tests for studying classifier performance. *J. Mach. Learn. Res.* **2010**, *11*, 1833–1863.
30. Tashman, L.J. Out-of-sample tests of forecasting accuracy: An analysis and review. *Int. J. Forecast.* **2000**, *16*, 437–450. [CrossRef]
31. Granger, C.W.J. Investigating causal relations by econometric models and cross-spectral methods. *Econometrica* **1969**, *37*, 424–438. [CrossRef]
32. Ashley, R.; Granger, C.W.J.; Schmalensee, R. Advertising and aggregate consumption: An analysis of causality. *Econometrica* **1980**, *48*, 1149–1167. [CrossRef]
33. Chernozhukov, V.; Chetverikov, D.; Demirer, M.; Duflo, E.; Hansen, C.; Newey, W.; Robins, J. Double/debiased machine learning for treatment and structural parameters. *Econom. J.* **2018**, *21*, C1–C68. [CrossRef]
34. Bach, P.; Chernozhukov, V.; Kurz, M.S.; Spindler, M. DoubleML: An object-oriented implementation of double machine learning in Python. *J. Mach. Learn. Res.* **2022**, *23*, 53.
35. Chernozhukov, V.; Cinelli, C.; Newey, W.; Sharma, A.; Syrgkanis, V. *Long Story Short: Omitted Variable Bias in Causal Machine Learning*; NBER Working Paper 30302; National Bureau of Economic Research: Cambridge, MA, USA, 2022.
36. Cinelli, C.; Hazlett, C. Making sense of sensitivity: Extending omitted variable bias. *J. R. Stat. Soc. Ser. B* **2020**, *82*, 39–67. [CrossRef]
37. White, H. A heteroskedasticity-consistent covariance matrix estimator and a direct test for heteroskedasticity. *Econometrica* **1980**, *48*, 817–838. [CrossRef]
38. Newey, W.K.; West, K.D. A simple, positive semi-definite, heteroskedasticity and autocorrelation consistent covariance matrix. *Econometrica* **1987**, *55*, 703–708. [CrossRef]
39. Pesaran, M.H.; Shin, Y.; Smith, R.J. Bounds testing approaches to the analysis of level relationships. *J. Appl. Econom.* **2001**, *16*, 289–326. [CrossRef]
40. Johansen, S. Estimation and hypothesis testing of cointegration vectors in Gaussian vector autoregressive models. *Econometrica* **1991**, *59*, 1551–1580. [CrossRef]
41. Bollerslev, T. Generalized autoregressive conditional heteroskedasticity. *J. Econom.* **1986**, *31*, 307–327. [CrossRef]
42. Schuirmann, D.J. A comparison of the two one-sided tests procedure and the power approach for assessing the equivalence of average bioavailability. *J. Pharmacokinet. Biopharm.* **1987**, *15*, 657–680. [CrossRef] [PubMed]
43. Makridakis, S.; Spiliotis, E.; Assimakopoulos, V. The M4 Competition: 100,000 time series and 61 forecasting methods. *Int. J. Forecast.* **2020**, *36*, 54–74. [CrossRef]
44. Garza, A.; Mergenthaler Canseco, M.; Challú, C.; Olivares, K.G. StatsForecast: Lightning Fast Forecasting with Statistical and Econometric Models, Version 2.1.1. Available online: https://github.com/Nixtla/statsforecast (accessed on 13 November 2026).
45. Hyndman, R.J.; Khandakar, Y. Automatic time series forecasting: The forecast package for R. *J. Stat. Softw.* **2008**, *27*, 1–22. [CrossRef]
46. Assimakopoulos, V.; Nikolopoulos, K. The theta model: A decomposition approach to forecasting. *Int. J. Forecast.* **2000**, *16*, 521–530. [CrossRef]
47. Nixtla. MLForecast: Scalable Machine Learning for Time Series Forecasting, Version 1.1.0. Available online: https://github.com/Nixtla/mlforecast (accessed on 13 November 2026).
48. Ke, G.; Meng, Q.; Finley, T.; Wang, T.; Chen, W.; Ma, W.; Ye, Q.; Liu, T.-Y. LightGBM: A highly efficient gradient boosting decision tree. *Adv. Neural Inf. Process. Syst.* **2017**, *30*, 3146–3154.
49. Longley, J.W. An appraisal of least squares programs for the electronic computer from the point of view of the user. *J. Am. Stat. Assoc.* **1967**, *62*, 819–841. [CrossRef]
50. Seabold, S.; Perktold, J. Statsmodels: Econometric and statistical modeling with Python. In Proceedings of the 9th Python in Science Conference, Austin, TX, USA, 28 June–3 July 2010; pp. 92–96.
51. Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; et al. Scikit-learn: Machine learning in Python. *J. Mach. Learn. Res.* **2011**, *12*, 2825–2830.
52. U.S. Bureau of Labor Statistics. Unemployment Rate [UNRATE]; FRED, Federal Reserve Bank of St. Louis. Available online: https://fred.stlouisfed.org/series/UNRATE (accessed on 12 November 2026).
53. Ali, M. PyCaret: An Open Source, Low-Code Machine Learning Library in Python. Available online: https://www.pycaret.org (accessed on 22 November 2026).

54. Shchur, O.; Turkmen, A.C.; Erickson, N.; Shen, H.; Shirkov, A.; Hu, T.; Wang, B. AutoGluon–TimeSeries: AutoML for probabilistic time series forecasting. In Proceedings of the International Conference on Automated Machine Learning, Potsdam, Germany, 12–15 September 2023; pp. 1–21.
55. Gogas, P. *4Cast Lab: Open-Source App for Machine Learning and Statistical Analyses*, Version 6.1.1; E|M|E|F Lab, Democritus University of Thrace: Komotini, Greece, 2026. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.