WWW.IJRPR.COM

International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

Adaptive AutoML for Structured Tabular Prediction: Design, Implementation, and Reproducible Evaluation of a Lightweight No-Code Baseline Platform

Kaushik P[1], Jai Kaarthik.B[2], Dr. G. Paavai Anand[3]

[1–2] Undergraduate Students; [3] Faculty Guide
Department of Computer Science and Engineering
SRM Institute of Science and Technology, Vadapalani, Chennai, India
[1] RA2411003040047 [2] RA2411003040035
[1] kp2230@srmist.edu.in [2] jb3259@srmist.edu.in

ABSTRACT :

Automated machine learning (AutoML) aims to reduce the repetitive effort required to transform a structured dataset into an evaluated predictive model. However, full AutoML systems often involve extensive search, nontrivial compute costs, and methodological complexity that may be unsuitable for introductory or lightweight applications. This paper presents Adaptive AutoML, a Python and Streamlit platform that integrates CSV/Excel ingestion, data profiling, task and target configuration, automated type-dependent preprocessing, comparison of three fixed candidate estimators per task, and downloadable prediction outputs. The implemented classification candidates are logistic regression, random forests, and extremely randomized trees; the regression candidates are ridge regression, random forests, and extremely randomized trees. Training-only fitting of preprocessing steps reduces a common source of leakage. We audit the supplied implementation, execute all three provided unit tests, and conduct 105 estimator fits across seven public or deterministic synthetic datasets using five random train/test splits. Additional two-stage evaluations reserve untouched external test subsets to examine the distinction between model selection and generalization assessment. On the Wine benchmark, Extra Trees attains 0.9889 mean accuracy across five splits; on synthetic Friedman-1 regression, Extra Trees attains a mean root-mean-square error of 1.9711. Results vary substantially by dataset, and no model family dominates all tasks. Importantly, the current interface selects its best model using the same held-out data whose scores it displays; consequently, the displayed winning score is not an independent post-selection performance estimate. We position the contribution as an accessible, auditable baseline workflow rather than a novel optimization algorithm, and identify concrete steps required for stronger validity and deployment.

Keywords: automated machine learning; tabular prediction; Streamlit; scikit-learn; model selection; leakage-aware preprocessing; reproducibility

1. Introduction

Supervised prediction projects frequently repeat the same operational sequence: import a dataset, inspect target and feature columns, address missing values, encode categorical variables, train several plausible estimators, compare task-appropriate metrics, and export predictions. Reimplementing these operations for every dataset creates a practical barrier for learners and small teams, even when existing Python libraries make each individual step straightforward. AutoML research responds to a broader challenge of automatically selecting algorithms, processing pipelines, and hyperparameters (Hutter et al., 2019; Feurer et al., 2015; Olson & Moore, 2016). Yet there is an important distinction between exhaustive pipeline optimization and the narrower automation of a transparent baseline workflow.

Adaptive AutoML is an implemented web application for the latter problem. It combines the interface capabilities of Streamlit with numerical and machine-learning functionality from pandas, NumPy, and scikit-learn (McKinney, 2010; Pedregosa et al., 2011; Streamlit, 2026). Users specify a prediction task and target column; the system applies a fixed preprocessing template, trains three compatible algorithms, ranks their performance, and makes the chosen fitted pipeline available for predictions. This offers a concrete engineering artifact that is easier to inspect than a black-box search system while remaining useful as a baseline for classification and regression.

The research issue is therefore twofold. First, can a small and reusable application execute a coherent end-to-end prediction workflow across differently structured datasets without writing new training code for each dataset? Second, how should the output be interpreted when the same split is used both to choose a winning model and to report its performance? The second issue matters because selecting the minimum observed error over several algorithms can introduce optimism into the selected score, even if the preprocessing itself is correctly confined to training rows (Varma & Simon, 2006; Raschka, 2018).

1.1 Research questions

RQ1: Does the supplied implementation support ingesting, preprocessing, training, evaluating, and predicting across binary, multiclass, and numerical targets? RQ2: How does the relative performance of its fixed baseline estimators change across real and synthetic tabular datasets? RQ3: What threats to valid generalization assessment remain after training-only preprocessing, and how can they be tested independently?

1.2 Contributions and boundaries

This study makes three bounded contributions. It documents the implemented architecture and its exact estimator and preprocessing configurations; it reports reproducible test and benchmark results rather than treating slide illustrations as measured evidence; and it subjects the selection procedure to an independent-holdout audit. It does not claim a new learning algorithm, a mathematically novel search strategy, improvements over established optimizers such as auto-sklearn or TPOT, or validated usability gains. Such claims would require different experiments and comparators.

2. Related Work and Theoretical Background

2.1 Automated machine learning and baseline automation

AutoML generally encompasses choices about transformations, learning algorithms, model configurations, and evaluation protocols. The survey edited by Hutter, Kotthoff, and Vanschoren explains how automation can span the full machine-learning pipeline (Hutter et al., 2019). Systems such as auto-sklearn search broad configuration spaces and use meta-learning, Bayesian optimization, and ensembling (Feurer et al., 2015). TPOT employs genetic programming to search pipeline designs (Olson & Moore, 2016). In contrast, Adaptive AutoML maintains a predetermined list of three estimators for the chosen task. Its adaptivity consists primarily of responding to user-specified task type and observed column types rather than learning which pipelines or hyperparameters to explore.
This narrower approach has benefits and limitations. A fixed candidate set is simple to audit, predictable in broad resource requirements, and suitable for demonstrating the mechanics of tabular prediction. It also cannot guarantee competitive performance on a novel dataset: important families such as gradient-boosted trees and support-vector machines are absent, and there is no budget-aware or data-dependent hyperparameter search. Random search and more elaborate configuration methods illustrate directions for systematic future optimization (Bergstra & Bengio, 2012).

2.2 Linear and ensemble estimators

For classification, logistic regression provides a regularized discriminative baseline with a parametric decision function; random forests average or vote across bootstrap-trained decision trees; and Extra Trees inject additional randomness into split selection (Breiman, 2001; Geurts et al., 2006). For regression, ridge regression fits a linear relation while penalizing large coefficients, and the tree ensembles represent nonlinear interactions without specifying a particular functional form (Breiman, 2001; Geurts et al., 2006; Hoerl & Kennard, 1970). The three-family comparison is therefore useful pedagogically: linear and nonlinear estimators respond differently to data dimensionality, noise, interactions, and sample size.

2.3 Pipeline-based preprocessing and model assessment

A common failure in model evaluation occurs when statistics computed from evaluation rows influence fitted transformations. The scikit-learn Pipeline interface addresses this specific threat by allowing data transformation and model fitting to be combined while learning preprocessing parameters exclusively from training rows (Pedregosa et al., 2011; scikit-learn developers, 2026a). It does not automatically prevent every leakage mechanism. Features that encode the target, duplicated entities split across training and testing, temporal dependence, or preprocessing decisions made using target labels outside the split may still invalidate conclusions. Selection on test outcomes is a separate statistical issue: independent final evaluation or nested resampling is needed when claiming generalization performance after comparing candidate estimators (Varma & Simon, 2006; Raschka, 2018).

3. Problem Formulation and Scope

Let \(D = \{(x_i, y_i)\}\) for \(i = 1,\ldots,n\) denote a labeled tabular dataset, where the feature vector may contain numerical and categorical attributes, and \(y\) represents either a discrete class label or a continuous numerical target. The user selects the task type \(T\) and the corresponding target column based on the prediction objective. Depending on the selected task, the platform constructs a candidate set \(M_T\) consisting of three compatible scikit-learn estimators and applies an appropriate preprocessing pipeline fitted exclusively on the training partition. The dataset is divided into training and testing subsets to support model development and performance comparison. Each candidate estimator is trained using the processed training data and subsequently evaluated on the held-out testing partition using task-specific evaluation metrics. For classification, the model with the highest accuracy is selected, whereas regression selects the model with the lowest root-mean-square error (RMSE). The selection process can be expressed mathematically as follows:

m* = arg max_m Accuracy(D_test, f_m) [classification]

m* = arg min_m RMSE(D_test, f_m) [regression]

The resulting workflow provides users with an inspected dataset, a comparative ranking of candidate models, evaluation metrics, and predictions for held-out or newly supplied feature records. These expressions represent the model-selection procedure implemented in the application rather than a statistically recommended final-test evaluation strategy. Since the selected model depends on the observed outcomes of the held-out partition, its displayed performance may be optimistic due to repeated candidate comparison. Therefore, an additional untouched test partition is necessary to independently assess the generalization performance of the selected estimator. The supplementary evaluation in this study examines this distinction without implying that the original application implements the additional evaluation procedure. The system is designed primarily for independent-row, supervised learning tasks involving structured tabular datasets. Its current scope does not include time-series forecasting, image or text processing, spatial or group-based data splitting, neural architecture search, or validation for high-stakes deployment environments.

Additionally, the platform establishes a consistent evaluation environment by applying the same data partitioning strategy across all candidate estimators within each experimental run. This allows the models to be compared under identical training and testing conditions. The preprocessing pipeline handles missing numerical and categorical values through separate imputation strategies, ensuring compatibility with mixed-type datasets. Numerical features undergo standardization, while categorical attributes are transformed using one-hot encoding. These transformations are incorporated into the estimator pipeline to maintain consistency between training and prediction stages. The platform also supports reproducibility through configurable random seeds, allowing experiments to be repeated under controlled conditions. However, the candidate selection process remains restricted to predefined algorithms and does not explore alternative model architectures automatically. Consequently, the quality of the selected estimator depends on the suitability of the

available candidates for the underlying dataset. The framework prioritizes accessibility, computational simplicity, and transparency over exhaustive model optimization. This scope provides a foundation for evaluating lightweight AutoML workflows while identifying areas that require further methodological and engineering improvements.

4. System Design and Implementation

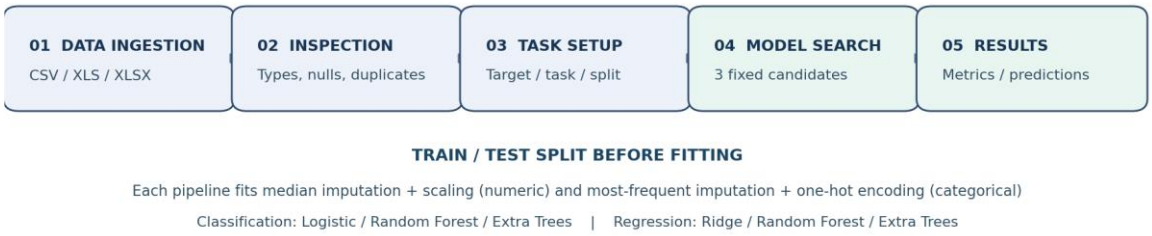


Figure 1. Implemented data-to-prediction pathway. All fitted preprocessing parameters are learned from the training partition.

4.1 Data ingestion and quality inspection

The Streamlit front end accepts CSV, XLSX, and XLS uploads via a file-uploader control and delegates file reading to pandas. The interface displays counts of rows, columns, missing cells, and duplicate rows; a preview of the first 20 observations; each column's data type; missing-value frequency and percentage; and the number of distinct non-null values. Fully empty columns are removed before target selection. Duplicate column names and data with fewer than two remaining columns are rejected. Rows lacking target values are dropped at training time. A minimum of eight labeled records is enforced, but this threshold alone does not establish statistical adequacy.

4.2 Task selection and conditional preprocessing

The user manually selects Classification or Regression, a target column, the proportion assigned to testing (15–40%, default 20%), and a random seed (default 42). The system does not infer or validate whether a numeric target should be modeled as categorical; classification converts target values to strings, while regression requires a numeric target. Numeric and Boolean features are passed through median imputation followed by standard scaling; other features undergo most-frequent imputation followed by one-hot encoding with unknown categories ignored. A ColumnTransformer and Pipeline keep these operations coupled to the estimator (Pedregosa et al., 2011; scikit-learn developers, 2026a).

4.3 Candidate estimators and model comparison

The classifier options are LogisticRegression(max_iter=1500), RandomForestClassifier(n_estimators=160, class_weight="balanced", random_state=seed, n_jobs=-1), and ExtraTreesClassifier with the same ensemble size, class weight, random state, and parallel setting. Regression candidates are Ridge() and 160-tree RandomForestRegressor/ExtraTreesRegressor with the same random-state and parallel settings. Aside from these settings, scikit-learn defaults apply; no search over hyperparameters occurs. Training errors are recorded per candidate, and the workflow fails if every candidate fails

Task	Algorithm	Explicit implementation parameters
Classification	Logistic regression	max_iter = 1500
Classification	Random forest	n_estimators = 160; class_weight = balanced; n_jobs = -1; random_state = seed
Classification	Extra Trees	n_estimators = 160; class_weight = balanced; n_jobs = -1; random_state = seed
Regression	Ridge regression	Library defaults
Regression	Random forest	n_estimators = 160; n_jobs = -1; random_state = seed
Regression	Extra Trees	n_estimators = 160; n_jobs = -1; random_state = seed

Table 1. Exact candidate-model configurations in the supplied automl_core.py.

4.4 User-facing results and prediction interface

Following training, the app displays all successful candidate metrics, identifies a selected model, and generates its predictions for the saved test rows. Class probabilities are appended when the selected classifier supports predict_proba. A dynamic table editor allows entry of new feature records, which are passed through the fitted pipeline without asking the user to manually repeat preprocessing. Outputs can be downloaded as CSV. Models are held in the current Streamlit session, rather than stored as portable, versioned artifacts.

4.5 Implementation-to-evidence traceability

The core preprocessing and estimator selection logic appears in automl_core.py, while app.py contains ingestion controls, inspection panels, training interactions, metrics, prediction editors, and CSV export. The project README describes this as baseline AutoML and lists cross-validation, tuning, model explainability, and model persistence as unimplemented. The presentation additionally contains a clinical-screening demonstration chart. Because neither its underlying data nor a reproducible generation procedure was supplied, the chart is not used as empirical evidence in this paper.

5. Mathematical and Evaluation Framework

5.1 Numeric and categorical transformations

For a numerical feature x_j, missing values are replaced with the median computed on the training set. The resulting feature is standardized using training-only mean μ_j and standard deviation σ_j. If σ_j is zero, the library's handling of constant features applies. For categorical attributes, missing entries are replaced using the training-set most-frequent category, and a one-hot representation is learned; unseen categories at prediction time are ignored rather than causing errors. This yields a consistent numeric representation for all six candidate estimators.

$$z_{ij} = (x_{ij} - \mu_{j,\text{train}}) / \sigma_{j,\text{train}} \quad (1)$$

5.2 Classification metrics

Classification reports accuracy, balanced accuracy, weighted precision, weighted recall, and weighted F1. For K classes, let TP_k, FP_k, and FN_k denote per-class counts and n_k the number of true observations in class k. Weighted metrics average classwise values according to empirical class support; balanced accuracy averages classwise recall. Weighted precision and F1 can therefore differ from overall accuracy, and balanced accuracy may be more revealing when class frequencies differ.

$$\text{Accuracy} = (1/n) \Sigma_i \, 1[\hat{y}_i = y_i] \quad (2)$$

$$\text{Balanced accuracy} = (1/K) \Sigma_k \, TP_k/(TP_k + FN_k) \quad (3)$$

$$\text{Weighted F1} = \Sigma_k \, (n_k/n) \cdot [2P_kR_k/(P_k + R_k)] \quad (4)$$

5.3 Regression metrics

Regression reports mean absolute error, root-mean-square error, and the coefficient of determination R^2. Both MAE and RMSE use target units; a value of RMSE from one dataset should not be compared directly against a different dataset's RMSE. In contrast, within-dataset comparisons between estimators are meaningful under the same partitioning protocol. RMSE weights large residuals more heavily than MAE.

$$\text{MAE} = (1/n) \Sigma_i \, |y_i - \hat{y}_i| \quad (5)$$

$$\text{RMSE} = \sqrt{[(1/n) \Sigma_i \, (y_i - \hat{y}_i)^2]} \quad (6)$$

$$R^2 = 1 - [\Sigma_i \, (y_i - \hat{y}_i)^2 / \Sigma_i \, (y_i - \bar{y})^2] \quad (7)$$

5.4 Selection versus independent assessment

The app ranks classifiers by maximum holdout accuracy and regressors by minimum holdout RMSE. It then displays that same split's selected-model metrics. Correct training-only preprocessing prevents one type of leakage, but it does not make this holdout a fully untouched test set after selection. The study therefore reports (i) descriptive model-by-model results from the implemented one-split workflow and (ii) a separate supplementary audit in which candidate selection occurs on an inner split and the selected fitted pipeline is evaluated on an outer split not accessed during selection (Varma & Simon, 2006; Raschka, 2018; scikit-learn developers, 2026a).

6. Experimental Methodology

6.1 Evaluation datasets

Seven datasets were used: four classification tasks and three regression tasks. Iris, Wine, Wisconsin Breast Cancer Diagnostic, and Diabetes come from scikit-learn's bundled dataset loaders (scikit-learn developers, 2026d; scikit-learn developers, 2026e; scikit-learn developers, 2026b; scikit-learn developers, 2026c). Friedman-1 is produced by scikit-learn's standard synthetic generator with 600 rows, 10 features, noise=1.0, and fixed random_state=20261031 (scikit-learn developers, 2026f). Two additional deterministic synthetic datasets include mixed numeric/categorical features and inserted missing values. The synthetic classification dataset uses numerical features labeled income and visits and a categorical channel predictor; the synthetic regression dataset uses age, experience, and segment. These are generated examples, not real individuals, and are used solely to exercise the pipeline on mixed types.

Dataset	Task	Rows	Features	Classes	Missing cells	Origin
Iris	Class.	150	4	3	0	Public library
Wine	Class.	178	13	3	0	Public library
Breast cancer	Class.	569	30	2	0	Public library
Mixed-type classification	Class.	600	3	2	150	Synthetic
Diabetes	Regr.	442	10	—	0	Public library
Friedman-1	Regr.	600	10	—	0	Synthetic
Mixed-type regression	Regr.	600	3	—	142	Synthetic

Table 2. Dataset inventory. "Missing cells" counts missing predictor entries, not missing targets.

6.2 Primary repeated-split protocol

The primary experimental evaluation was conducted independently on each of the seven selected datasets using the supplied fit_and_compare function. The original implementation was retained without modifying any estimator configurations, evaluation metrics, or preprocessing operations. Five different random seeds, namely 42, 52, 62, 72, and 82, were used to generate separate training and testing partitions. For each seed, the dataset was divided into 80% training data and 20% testing data. This repeated-split approach allowed the performance of individual candidate estimators to be examined under different data partitioning conditions. For classification tasks, stratified sampling was applied to maintain the relative distribution of target classes across training and testing subsets, provided that each class contained at least two records. This ensured that the class distribution was reasonably represented during model evaluation. For regression tasks, ordinary random splitting was employed because the target variable consisted of continuous numerical values. Within each experimental split, all three task-compatible candidate estimators were trained and evaluated using the same partition. This maintained consistent evaluation conditions and enabled direct comparisons between competing algorithms. The four classification datasets and three regression datasets were processed using their respective predefined candidate model configurations. Consequently, each dataset contributed fifteen estimator fits across the five experimental seeds. The overall experimental workload can be represented as follows:

Total estimator fits = 7 datasets × 5 random seeds × 3 estimators = 105 fits.

The performance measurements obtained from these experiments were aggregated separately for each candidate estimator. Mean values were calculated to summarize the average performance across the five repeated splits. Sample standard deviations were also computed to represent the variability in performance resulting from different train/test partitions. These measurements provide an indication of the consistency of each estimator under the selected experimental conditions. The same random partitions were shared among competing estimators within each seed to reduce partition-related differences during comparison. However, the repeated splits are not statistically independent datasets and should not be interpreted as independent observations. Therefore, the reported means and standard deviations are descriptive measures of experimental variability rather than confidence intervals or formal statistical tests of superiority. No additional hyperparameter tuning or modifications to the supplied training workflow were introduced during this primary evaluation.

6.3 Supplementary external-holdout audit

A second experiment assesses the selected model on previously untouched rows. For each dataset and seed, an outer 80/20 split is created. The supplied fit_and_compare function is then called on the outer training partition alone with a 25% internal validation fraction. Thus approximately 60% of records train each candidate, 20% select the candidate, and the remaining 20% form an independent outer test set. The inner-selected fitted pipeline is applied directly to the outer test rows without refitting; in particular, no outer test outcomes influence selection. This wrapper is an evaluation procedure written for this paper, not a claim that the Streamlit app already implements a three-way split. Differences between inner and outer scores reflect both split variability and model-selection effects; they should not be interpreted as a precise causal estimate of selection bias.

6.4 Verification, environment, and reporting decisions

All three provided unit test cases were executed successfully, demonstrating the basic functional correctness of the supplied implementation. The first test evaluated classification using a dataset containing both numerical and categorical features, including a missing numerical value. The second test examined the regression workflow using a numerical prediction target. The third test verified the exclusion of records containing missing target labels during the training process. All three tests passed without requiring modifications to the original core implementation. The experimental evaluation was conducted using Python version 3.13.5, scikit-learn version 1.8.0, pandas version 2.2.3, and NumPy version 2.3.5. These software versions represent the execution environment used to obtain the reported results. Although the project README recommends Python versions 3.10–3.12, the successful execution of the experiments on Python 3.13.5 demonstrates compatibility within the tested environment. However, this observation should not be interpreted as a guarantee of compatibility across all operating systems or software configurations. The evaluation datasets were prepared using the specified public dataset loaders and deterministic synthetic data-generation procedures. Feature engineering was restricted to the transformations required to construct the evaluation datasets and accommodate their respective data types. No additional feature selection, manual feature engineering, or hyperparameter optimization was introduced during the experimental evaluation. All candidate estimators were evaluated using the configurations provided by the original implementation. Random seeds were maintained across repeated experimental runs to support reproducibility and consistent comparisons. The same data partitions were used for competing estimators within each experimental split. This approach helped minimize differences arising solely from variations in dataset partitioning. The supplementary independent-holdout evaluation was conducted separately using an additional evaluation wrapper. This procedure was not incorporated into the original Streamlit application. Numerical results were rounded only for presentation

purposes, while the underlying experimental calculations retained their original precision. Standard deviations were reported alongside mean performance values to illustrate variability across repeated splits. The reported findings are therefore interpreted as descriptive experimental observations rather than formal statistical evidence of model superiority. These verification and reporting decisions establish a transparent basis for understanding the experimental environment, implementation behavior, and limitations of the reported results.

7. Experimental Results

7.1 Classification: task-dependent performance

Dataset	Estimator	Accuracy, mean ± SD	Weighted F1	Selected / 5
Iris	Logistic Regression	0.9533 ± 0.0298	0.9532	4
Iris	Random Forest	0.9467 ± 0.0380	0.9463	1
Iris	Extra Trees	0.9600 ± 0.0279	0.9598	0
Wine	Logistic Regression	0.9778 ± 0.0124	0.9778	2
Wine	Random Forest	0.9833 ± 0.0248	0.9832	2
Wine	Extra Trees	0.9889 ± 0.0248	0.9887	1
Breast cancer	Logistic Regression	0.9684 ± 0.0202	0.9682	3
Breast cancer	Random Forest	0.9649 ± 0.0139	0.9649	1
Breast cancer	Extra Trees	0.9667 ± 0.0114	0.9666	1
Mixed-type classification	Logistic Regression	0.7417 ± 0.0177	0.7418	5
Mixed-type classification	Random Forest	0.6800 ± 0.0389	0.6800	0
Mixed-type classification	Extra Trees	0.6683 ± 0.0285	0.6685	0

Table 3. Measured classification results from five 80/20 splits; model selected by held-out accuracy (ties can affect selection counts).

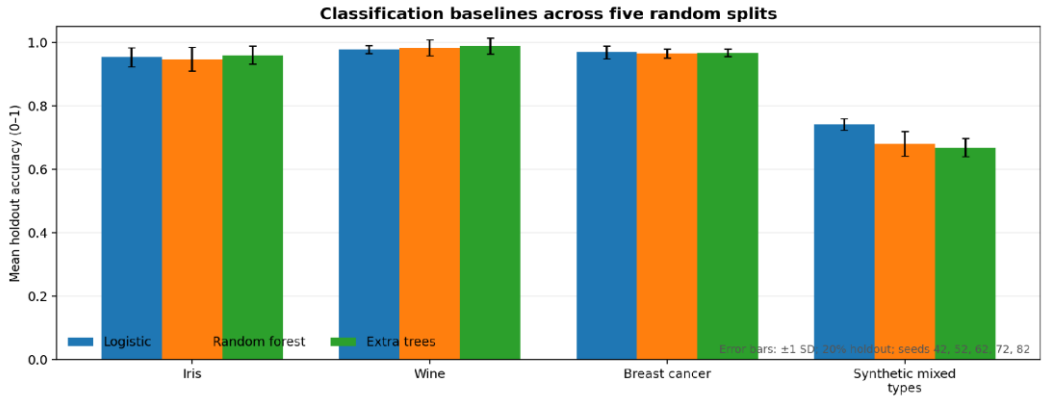


Figure 2. Measured classification accuracy by estimator and dataset. Error bars show one sample standard deviation.

On the Wine dataset, Extra Trees had the highest average accuracy (0.9889), while logistic regression averaged 0.9778. On Iris, all three estimators scored above 0.94 on average; Extra Trees had the highest mean (0.9600), yet was never selected by the app in these five seeds because selection is performed separately on each split and equal scores can favor an earlier listed candidate. On the breast-cancer benchmark, logistic regression obtained 0.9684 mean accuracy, marginally above the ensembles. On synthetic mixed-type classification, logistic regression led with 0.7417 mean accuracy versus 0.6800 for random forest and 0.6683 for Extra Trees. This pattern demonstrates why a fixed model ranking cannot be assumed across target-generating mechanisms. The breast-cancer data are used as a public algorithmic benchmark only; these scores do not establish clinical validity.

7.2 Regression: linear versus nonlinear data

Dataset	Estimator	RMSE, mean ± SD	Mean R^2	Selected / 5
Diabetes	Ridge Regression	53.312 ± 3.468	0.511	4
Diabetes	Random Forest	57.335 ± 2.465	0.438	0
Diabetes	Extra Trees	56.391 ± 2.818	0.456	1
Friedman-1	Ridge Regression	2.593 ± 0.130	0.713	0
Friedman-1	Random Forest	2.103 ± 0.087	0.812	0
Friedman-1	Extra Trees	1.971 ± 0.047	0.835	5
Mixed-type regression	Ridge Regression	4.374 ± 0.237	0.532	0
Mixed-type regression	Random Forest	3.961 ± 0.314	0.617	5
Mixed-type regression	Extra Trees	4.141 ± 0.316	0.582	0

Table 4. Measured regression results. RMSE is expressed in each dataset's own target units.

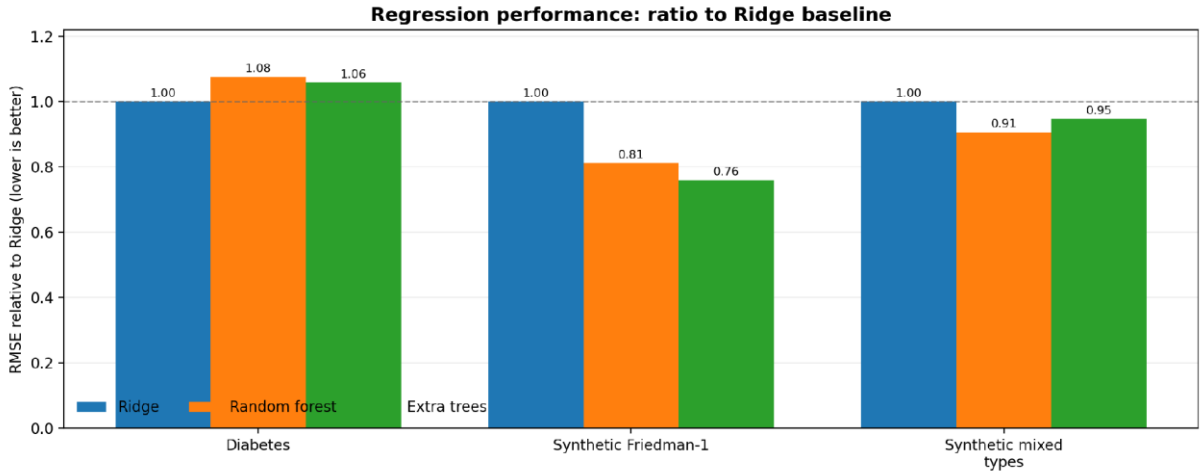


Figure 3. Normalized regression comparison: each estimator's mean RMSE divided by the corresponding Ridge RMSE.

Ridge regression gave the lowest mean RMSE for Diabetes (53.312), outperforming random forest (57.335) and Extra Trees (56.391) in these splits. On Friedman-1, the nonlinear tree ensembles improved considerably over ridge; Extra Trees achieved 1.971 mean RMSE and mean $R^2 = 0.835$. On synthetic mixed-type regression, random forest was best (3.961 mean RMSE), followed by Extra Trees (4.141) and ridge (4.374). The results support the practical value of including both linear and tree-based baselines, but cannot establish that this particular set is optimal or comprehensive.

7.3 Independent model-selection audit

Classification dataset	Inner selection accuracy	Independent test accuracy	Test SD
Iris	0.9267	0.9533	0.0298
Wine	0.9889	0.9722	0.0196
Breast cancer	0.9754	0.9702	0.0147
Mixed-type classification	0.7483	0.7400	0.0216

Table 5. Supplementary 60/20/20 selection-and-audit protocol; five outer splits per dataset.

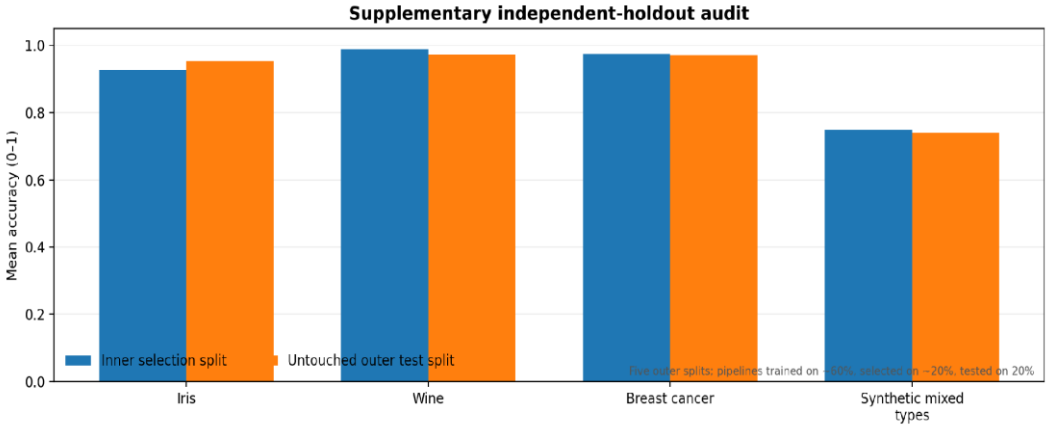


Figure 4. Scores used for inner model choice versus independent outer-test accuracy. The two sets are not interchangeable.

Regression dataset	Inner selection RMSE	Independent test RMSE	Independent R^2
Diabetes	52.884	54.874	0.484
Friedman-1	2.223	2.075	0.817
Mixed-type regression	4.036	3.974	0.615

Table 6. Supplementary independently evaluated regression models. Lower RMSE is better.

Independent outer accuracies averaged 0.9533 on Iris, 0.9722 on Wine, 0.9702 on breast cancer, and 0.7400 on synthetic mixed-type classification. Independent outer RMSE averaged 54.874 for Diabetes, 2.075 for Friedman-1, and 3.974 for synthetic mixed-type regression. Inner-versus-outer scores did not change in a single direction: Iris outer accuracy was above inner selection accuracy, whereas Wine inner accuracy was higher. This mixed pattern reinforces the need for an untouched evaluation split but does not quantify a universal optimism correction. Since the outer protocol uses fewer training rows than the primary experiment, its outcomes should be read as a separate design rather than a direct adjusted version of Tables 3–4.

7.4 Functional verification and observed scope

The test suite passed all three available cases without requiring modifications to the original core implementation. The seven experimental datasets produced successful fits for all three applicable candidate estimators across the five seeds. The synthetic mixed-type cases additionally verify that the system can process combined categorical and numeric inputs containing missing predictor values in the tested environment. This evidence confirms coverage of these execution paths, not exhaustive testing of file parsing, Streamlit user interaction, malformed schemas, adversarial data, or portability to every supported Python and operating-system combination.

8. Discussion

8.1 Interpretation of "adaptive"

The demonstrated adaptation is structural rather than optimization-driven. Selecting Classification instead of Regression changes the candidate model set and scoring metric; selecting different column types changes the preprocessing branches. These are legitimate forms of workflow adaptation, but no learned meta-policy, conditional hyperparameter search, or self-improving model search was found in the code. A technically precise description is therefore "task-conditional baseline AutoML platform." Advertising full automated model optimization would exceed the implemented evidence.

8.2 Dataset dependence and baseline diversity

The benchmark results show different winners depending on data geometry. The mixed-type classification generator exhibits a strong approximately additive relationship, favoring logistic regression. Friedman-1 explicitly includes nonlinear interactions, on which the ensembles perform better. Diabetes favors ridge under these configurations and partitions. The result does not imply that a given family universally represents a dataset's true structure; it shows that even a compact search space can capture useful diversity. The benefit of the third candidate should eventually be assessed with prespecified comparison metrics, statistical uncertainty, and multiple-seed evaluation across more independent datasets.

8.3 Selection on a reporting split

The most important validity limitation is the dual use of the holdout split. Preprocessing is correctly isolated to training data, but the selected model is the one that happens to maximize observed accuracy or minimize observed RMSE on the reporting data. The winner's displayed score is conditional on that selection. Varma and Simon demonstrate why model-selection procedures must themselves be nested within an independent evaluation process (Varma & Simon, 2006). The remedy is not merely adding more seeds: a training partition, a validation partition for model choice, and an untouched final test (or an appropriate nested cross-validation design) must be maintained end to end. The supplementary audit operationalizes this distinction without altering the evaluated application.

8.4 Choosing metrics that reflect user cost

Accuracy is understandable but may obscure minority-class performance; weighted F1 can similarly be dominated by frequent labels. Although balanced accuracy is reported, it is not presently used for model selection. Regression selection uses RMSE, which is sensitive to large errors. Future versions should permit the metric to reflect the actual operational cost structure and, when probabilities are used, assess calibration and threshold selection. A no-code interface does not relieve users of the responsibility to define the prediction target, acceptable error, or deployment context.

8.5 Reproducibility versus generalizability

The presence of deterministic seeds, explicit source code, a small test suite, and bundled benchmark datasets improves reproducibility. Nevertheless, small tabular benchmarks are familiar and may not represent difficult operational settings with evolving distributions, high-cardinality identifiers, data-entry inconsistencies, or entity overlap. Results presented here do not benchmark speed, RAM use, accessibility, usability, cybersecurity, long-term stability, or advanced AutoML competitors. A subsequent study should record hardware and training time, package-lock information, uncertainty intervals, external datasets, and subgroup failure modes before making broader claims.

9. Threats to Validity and Limitations

Internal validity: training-only transformation fitting is a strength, but model selection and score reporting share a holdout. Repeated seeds are not independent datasets and the five splits are correlated. Missing-value mechanisms in synthetic datasets are imposed rather than observed, and real-data examples have no induced missingness in this evaluation. Construct validity: a completed prediction workflow is not a direct measurement of user productivity or accessibility; neither was formally tested. External validity: seven small-to-medium datasets, including three synthetic tasks, do not establish performance on arbitrary organizations, time-series, text, images, or high-stakes applications. Statistical validity: the investigation does not report confidence intervals, significance tests, nested cross-validation over a broad data suite, or a benchmark against full-scale AutoML systems.
Implementation limitations are also concrete. There is no automatic grouped or chronological validation, sophisticated date-feature engineering, explicit handling of extreme categorical cardinality, calibrated probabilities, interpretability, hyperparameter search, trained-model export, versioned experiment tracking, API deployment, access control, or drift monitoring. Class counts are checked for stratification, but some class-size/test-size combinations can still cause split errors, which the UI reports rather than automatically repairing. Session-state results can also become stale if certain training controls are changed without retraining. These limitations follow from the current code and should guide priority fixes rather than be concealed.

10. Responsible Use, Privacy, and Operational Considerations

Because the application handles uploaded tables in the active Streamlit process, users should not infer that data processing is certified private, encrypted end to end, or compliant with a particular regulatory framework. The current deployment topology, authentication policy, retention behavior, and hosting environment have not been assessed. For sensitive tabular information, organizations would need explicit data-governance arrangements, controlled hosting, schema validation, logging and access restrictions, retention rules, and separate security review.
Users must also evaluate whether the selected features would actually be available at inference time, whether a person or entity can appear in both training and test sets, and whether performance is consistent across relevant subpopulations. Classification probabilities are algorithm outputs, not guaranteed calibrated risks. Public breast-cancer and diabetes benchmarks are included to test software behavior; the platform as implemented is not validated for clinical decision-making, diagnosis, or treatment.

11. Future Research and Engineering Roadmap

11.1 Short-term: statistical validity

The highest-priority improvement is to separate model choice from final testing. A configuration panel should provide stratified or ordinary train/validation/test splitting, with a separate final test evaluated only once, and optionally nested cross-validation for research settings. Add clear handling for minority classes, alternate scoring criteria, confidence intervals, and a warning when sample sizes are too small. User-visible experiments should explicitly distinguish validation scores from final test scores.

11.2 Medium-term: smarter AutoML and interpretation

Expand candidate diversity to include gradient boosting and other context-appropriate algorithms, but introduce optimization in a measured way. Randomized search with a training budget is a reasonable first comparison (Bergstra & Bengio, 2012); later methods might use Bayesian optimization, early stopping, or a meta-learning initialization, provided their search overhead and benefit are evaluated. Feature importance, local explanations, error breakdowns, calibration metrics, and class threshold tuning would make predictions easier to scrutinize. Benchmarks should compare fixed baselines with tuned variants under equal resource budgets.

11.3 Longer-term: deployability and data realism

Support schema contracts, artifact serialization with version metadata, reproducible environments, user authentication, audit logs, drift detection, and controlled retraining. Time-aware, entity-aware, or group-aware data splitting should be selectable when independent-row assumptions fail. Formal usability testing could compare task completion time, errors, and comprehension between novice users of Adaptive AutoML and conventional notebook workflows. Such studies would transform a plausible ease-of-use motivation into empirically supported claims.

12. Conclusion

Adaptive AutoML is a functioning, lightweight platform that automates common steps in supervised tabular prediction while keeping model families and transformations visible. The code successfully executes ingestion-to-prediction logic for classification and regression, the three supplied unit tests

pass, and this study's 105 estimator fits produce reproducible, task-dependent baseline results. Logistic and ridge regression remain competitive on some datasets, while tree ensembles outperform linear baselines on certain nonlinear tasks. These findings substantiate usefulness as an introductory and reusable baseline, not superiority over advanced AutoML. The central methodological result is that correct train-only preprocessing and independent post-selection evaluation solve different problems. Improving the latter, along with validation design, model explanation, and experiment tracking, is necessary before the platform can support stronger research or production claims.

Declarations

Data availability. Public benchmark data are available directly through scikit-learn; synthetic data are generated deterministically by the accompanying reproduction scripts. No private training dataset was used to produce the numerical results in this paper.
Code availability. The research package contains the supplied automl_core.py and test_automl.py, along with separate scripts and CSV summaries created for this evaluation. The supplementary external-holdout wrapper is not part of the original app implementation.
Competing interests and funding. No funding or conflict-of-interest statements were provided in the supplied source material; authors should complete these declarations before external submission.

Appendix A. Reproducibility and Artifact Checklist

The artifact package should preserve (1) original source files: app.py, automl_core.py, README.md, requirements.txt, and test_automl.py; (2) new baseline evaluation script: automl_reproduce_experiments.py; (3) new independent-audit script: automl_independent_holdout.py; and (4) the per-run and aggregate CSV files. The experiments are deterministic conditional on the documented software behavior, datasets, and random seeds. Reproduction commands are shown below.

```
python -m unittest test_automl.py -v
python automl_reproduce_experiments.py
python automl_independent_holdout.py
```

The first script evaluates the candidate estimators on 80/20 held-out splits; the second runs an 80/20 outer split and 75/25 inner split of the outer-training data. Repeated test partitions within one dataset are not statistically independent samples; no p-values are claimed. The recorded CSVs provide row-level metric and selected-model traceability for every seed.

Appendix B. Practical Error and Edge-Case Considerations

The application intentionally drops rows with unknown target labels, not rows with unknown predictor values. When the remaining classification data have fewer than two target classes or any target class has fewer than two records, training raises a clear error. With very small stratified datasets, the requested test percentage may still be too small to represent every class. Numeric regression targets are mandatory. All-empty columns are removed, but columns that are almost empty, identification-like, or unexpectedly high-cardinality still require domain review.
The current selected-model result is stored in Streamlit session state along with the target and prediction type. A robust implementation should invalidate or retrain this state when the uploaded data, test proportion, or random seed changes; checking only target and task is insufficient to ensure the displayed results match every current control. Before deployment, unit and integration tests should also cover file encodings, duplicate entities, out-of-range feature values, unseen categories, schema drift, permissions, session isolation, and CSV export.

REFERENCES :

- Bergstra, J., & Bengio, Y. (2012). Random Search for Hyper-Parameter Optimization. Journal of Machine Learning Research, 13, 281–305. https://www.jmlr.org/papers/v13/bergstra12a.html
- Breiman, L. (2001). Random Forests. Machine Learning, 45, 5–32. https://doi.org/10.1023/A:1010933404324
- Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., & Hutter, F. (2015). Efficient and Robust Automated Machine Learning. Advances in Neural Information Processing Systems 28. https://papers.neurips.cc/paper_files/paper/2015/hash/11d0e6287202fced83f79975ec59a3a6-Abstract.html
- Geurts, P., Ernst, D., & Wehenkel, L. (2006). Extremely Randomized Trees. Machine Learning, 63, 3–42. https://doi.org/10.1007/s10994-006-6226-1
- Hoerl, A. E., & Kennard, R. W. (1970). Ridge Regression: Biased Estimation for Nonorthogonal Problems. Technometrics, 12(1), 55–67. https://doi.org/10.1080/00401706.1970.10488634
- Hunter, J. D. (2007). Matplotlib: A 2D Graphics Environment. Computing in Science & Engineering, 9(3), 90–95. https://doi.org/10.1109/MCSE.2007.55
- Hutter, F., Kotthoff, L., & Vanschoren, J. (Eds.). (2019). Automated Machine Learning: Methods, Systems, Challenges. Springer. https://doi.org/10.1007/978-3-030-05318-5
- McKinney, W. (2010). Data Structures for Statistical Computing in Python. Proceedings of the 9th Python in Science Conference, 56–61. https://doi.org/10.25080/Majora-92bf1922-00a
- Olson, R. S., & Moore, J. H. (2016). TPOT: A Tree-based Pipeline Optimization Tool for Automating Machine Learning. Proceedings of Machine Learning Research, 64, 66–74. https://proceedings.mlr.press/v64/olson_tpot_2016.html
- Pedregosa, F., et al. (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 12, 2825–2830. https://www.jmlr.org/papers/v12/pedregosa11a.html
- Raschka, S. (2018). Model Evaluation, Model Selection, and Algorithm Selection in Machine Learning. arXiv:1811.12808. https://arxiv.org/abs/1811.12808

- scikit-learn developers. (2026a). Common Pitfalls and Recommended Practices. https://scikit-learn.org/stable/common_pitfalls.html
- scikit-learn developers. (2026b). load_breast_cancer API Reference. https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_breast_cancer.html
- scikit-learn developers. (2026c). load_diabetes API Reference. https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_diabetes.html
- scikit-learn developers. (2026d). load_iris API Reference. https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_iris.html
- scikit-learn developers. (2026e). load_wine API Reference. https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_wine.html
- scikit-learn developers. (2026f). make_friedman1 API Reference. https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_friedman1.html
- Streamlit. (2026). Streamlit Documentation. https://docs.streamlit.io/
- Varma, S., & Simon, R. (2006). Bias in Error Estimation When Using Cross-Validation for Model Selection. BMC Bioinformatics, 7, Article 91. https://doi.org/10.1186/1471-2105-7-91