

000 001 002 003 004 005 DARE-BENCH: EVALUATING MODELING AND 006 INSTRUCTION FIDELITY OF LLMs IN DATA SCIENCE 007 008 009

010 **Anonymous authors**
011 Paper under double-blind review
012
013
014
015
016
017
018
019
020
021
022
023
024
025
026
027
028
029

ABSTRACT

030 The fast-growing demands in using Large Language Models (LLMs) to tackle
031 complex multi-step data science tasks create a emergent need for accurate bench-
032 marking. There are two major gaps in existing benchmarks: (i) the lack of stan-
033 dardized, process-aware evaluation that captures instruction adherence and pro-
034 cess fidelity, and (ii) the scarcity of accurately labeled training data. To bridge
035 these gaps, we introduce DARE-bench, a benchmark designed for machine learn-
036 ing modeling and data science instruction following. Unlike many existing bench-
037 marks that rely on human- or model-based judges, all tasks in DARE-bench have
038 verifiable ground truth, ensuring objective and reproducible evaluation. To cover
039 a broad range of tasks and support agentic tools, DARE-bench consists of 6,300
040 Kaggle-derived tasks and provides both large-scale training data and evaluation
041 sets. Extensive evaluations show that even highly capable models such as gpt-
042 o4-mini struggle to achieve good performance, especially in machine learning
043 modeling tasks. Using DARE-bench training tasks for fine-tuning can substan-
044 tially improve model performance. For example, supervised fine-tuning boosts
045 Qwen3-32B’s accuracy by 1.83 \times and reinforcement learning boosts Qwen3-4B’s
046 accuracy by more than 8 \times . These significant improvements verify the importance
047 of DARE-bench both as an accurate evaluation benchmark and critical training
048 data.

1 INTRODUCTION

049 Large language models (LLMs) (Anthropic, 2025a;b; OpenAI, 2025a;c; Yang et al., 2025) are in-
050 creasingly employed as data-science (DS) agents to perform data reading, transformation, and mod-
051eling through tool-augmented code execution. Such a rapid adoption demands rigorous benchmarks
052 to evaluate and enhance the effectiveness and reliability in performing these complex, multi-step
053 workflows. However, due to the cost and complexity of evaluation, existing benchmarks can only
054 evaluate final-answer accuracy, and leaving other valuable metrics such as process fidelity and re-
055 producibility largely unmeasured (Zhang et al., 2024; Jing et al., 2024). Meanwhile, many existing
056 works (Guo et al., 2024; Zhang et al., 2023; Hong et al., 2024) in this area focus on using prompt
057 engineering and workflow design to improve model performance. We compliment these works by
058 taking a benchmark approach to train LLM agents with high fidelity data and sophisticated yet re-
059 producible evaluation to better acquire domain-specific skills in DS workflows.



060 Figure 1: DARE-bench defines each task by providing a natural-language question and structured
061 files (metadata and train/test splits). An LLM agent executes code within a sandbox to generate
062 predictions, which are compared against ground truth for automatic and reproducible evaluation.

063 Creating benchmarks that capture process fidelity for both training and evaluation is significantly
064 challenging. The main challenge comes from two-fold. First, the sources for crafting training data

054

055

056

057

058

059

060

061

062

063

064

065

066

067

068

069

070

071

072

073

074

075

076

077

078

079

080

081

082

083

084

085

086

087

088

089

090

091

092

093

094

095

096

097

098

099

100

101

102

103

104

105

106

107

Table 1: Comparison between DARE-bench and existing benchmarks.

Benchmark	Domain	Data File	Inst-follow	Time Series	Verifiable	Train Tasks	Tasks
MLAgentBench (Huang et al., 2023)	Deep Learning	✓	-	-	✓	✗	13
MLE-bench (Chan et al., 2024)	Deep Learning	✓	-	-	✓	✗	75
SWE-bench (Jimenez et al., 2024)	Software Eng.	✓	-	-	✓	✓	21,294
DS-1000 (Lai et al., 2023)	Data Science	✗	✗	✗	✗	✗	1,000
Arcade (Yin et al., 2022)	Data Science	✗	✗	✗	✗	✗	1,082
Spider2V (Cao et al., 2024)	Data Science	✗	✗	✗	✓	✗	494
DSEval (Zhang et al., 2024)	Data Science	✓	✗	✗	✓	✗	825
DSBench (Jing et al., 2024)	Data Science	✓	✗	✗	✓	✗	540
DA-Code (Huang et al., 2024)	Data Science	✓	✗	✗	✓	✗	500
DataSciBench (Zhang et al., 2025)	Data Science	✓	✗	✗	✗	✗	222
DABstep (Egg et al., 2025a)	Data Science	✓	✗	✗	✓	✗	450
DSBC (Kadiyala et al., 2025)	Data Science	✓	✗	✗	✓	✗	303
DARE-bench (Ours)	Data Science	✓	✓	✓	✓	✓	6,300

(e.g., expert-level, executable DS process traces) are scarce and prohibitively expensive to acquire. Existing benchmarks largely rely on human-processed data and often center on Kaggle competitions, creating a major data bottleneck. Second, evaluating “process fidelity” is highly non-trivial as randomness and environment affects confound behavior, and verifying that an agent follows permissible DS practices requires a controlled, instrumented harness. These challenges limit the data quality and evaluation scope of existing benchmarks, and thus miss the opportunities to better release the full potential of models.

To address the challenge of data quality and scarcity, we leverage LLMs to process auxiliary content, such as task descriptions, metadata normalization, rule extraction, instead of heavily relying on human involvement so that the data generation is scalable with quality. We further improve the data quality with better diversity by pivoting from leaderboard-oriented Kaggle competitions to the broader pool of Kaggle datasets, yielding a more diverse and representative problem set such as time-series domains. To address the evaluation challenge, we engineer determinism (e.g., fixed seeds, reproducible environments) so that process fidelity is enabled by an outcome-based, verifiable reward—enabling reinforcement learning (RLVR) instead of human-involved reward. These approaches work coherently to construct a large-scale, trainable benchmark for data science that measures modeling performance and process fidelity, and boosts training performance.

To this end, we introduce **Datascience Agentic REasoning** bench (DARE-bench), a training-focused DS agent benchmark featuring two verifiable task families: (i) process-aware instruction-following tasks with ground truth from executing reference solutions that strictly follow the task instruction; and (ii) ML modeling tasks evaluated against the dataset’s original ground truth under reproducible metrics. Our design for the instruction-following tasks leverages a key advantage of data science: the high degree of reproducibility. We find that by controlling the randomness and providing explicit instructions, a procedurally faithful execution can produce a deterministic outcome. This allows us to robustly and automatically evaluate process fidelity by verifying the agent’s final answer against the ground truth. As shown in Figure 1, for both task families, each task provides a natural-language question and structured files. The LLMs execute code within a sandbox to generate predictions, which is checked automatically for scoring. In Table 1, we compare DARE-bench against existing benchmarks in terms of the task coverage, verifiability, training task support, and number of tasks to demonstrate DARE-bench’s significant advancements.

We conduct extensive evaluation on both strong general-purpose and code-centric LLMs. The evaluation results reveal that many LLMs without task-aligned training fail miserably due to process deviations, runtime errors, and metric mis-specification. For instance, Qwen3-32B baseline only achieves a total score of 23.25, while the smaller Qwen3-4B baseline performs even worse which scores 4.39. By contrast, DARE-bench bridges this gap by providing a training-focused benchmark with verifiable large-scale training data and useful and sophisticated reproducible evaluation. Supervised fine-tuning yields absolute gains of nearly 20 points, while reinforcement learning boosts Qwen3-4B from 4.39 to 37.40. Overall, DARE-bench significantly improve success rates, process adherence, predictive performance, and robustness across a variety of practical data science tasks.

108

109
110
111
Table 2: Overview of DARE-bench benchmark composition and the primary capabilities evaluated
by each task type. Variants are denoted as IF = Instruction Following, MM = ML Modeling, XF =
eXogenous Features, CF = Canonical Forecasting.

112

Task Type	Train Tasks	Test Tasks	Capability Assessed
Classification-IF	1160	74	Instruction following
Classification-MM	1160	74	ML Modeling
Regression-IF	899	45	Instruction following
Regression-MM	899	45	ML Modeling
Time-series-XF	915	57	Predictive ML, forecasting
Time-series-CF	915	57	Predictive ML, forecasting

118

119

120
2 RELATED WORK

121

LLM Agents. Research into Agentic LLMs focuses on their ability as independent agents through planning, tool calling, and memory capabilities. The integration of reasoning with actions or APIs occurs through ReAct (Yao et al., 2023) and Toolformer (Schick et al., 2023) frameworks as researchers work on multi-agent collaboration and autonomous tool-augmented systems. Applying these to real-world data science remains difficult because current benchmarks lack adequate training resources and often omit critical domains such as time-series forecasting or the distinction between open-ended problem solving and strict instruction-following.

128

LLMs for Coding and Data Science Benchmarks. The advancement of coding benchmarks depends on the use of testable pass/fail signals. The HumanEval (Chen et al., 2021) and MBPP (Austin et al., 2021) provided short self-contained functions with hidden unit tests while SWE-bench (Jimenez et al., 2024) tests models on actual GitHub issues that need multiple file modifications and complete project testing. The community now performs end-to-end data science (DS) tasks as its new approach to this paradigm. The DS-1000 (Lai et al., 2023) teaches NumPy/Pandas programming but DSBBench (Jing et al., 2024) and MLE-bench (Chan et al., 2024) use Kaggle competition problems which require multi-step analytics. The DABstep (Egg et al., 2025b) dataset contains 450 financial tasks from real-world applications and DataSciBench (Zhang et al., 2025) uses Task-Function-Code (TFC) to evaluate programs which are then verified by human evaluators. DSBC (Kadiyala et al., 2025) addresses private datasets via structured metadata. The research uses Chen et al. (2024) to evaluate visualization skills and Bendinelli et al. (2025) to assess data cleaning abilities and Kaggle leaderboards (Grosnit et al., 2024; Chan et al., 2024) to measure performance. The benchmarks show a sequential development from basic unit testing code to sophisticated tool-based agents which perform complete DS workflows and produce quantifiable results.

143

144

Reinforcement Learning with Verifiable Rewards. The implementation of verifiable programmatic signals in reinforcement learning enables model training at scale without requiring preference data. The automatic checking system consists of unit tests and solvers and execution traces for math and code verification. GRPO (Shao et al., 2024) achieves learning stability through its relative rollout feedback system which DeepSeek-R1 (Guo et al., 2025) and GPT o-series (OpenAI, 2025d) extend by verifier-enhanced objectives. The methods combine symbolic proofs with coding tests and retrieval/search execution graphs to improve reward-as-checker for both correct answers and verifiable reasoning trace generation.

150

151

3 DARE-BENCH

153

DARE-bench consists of three data science task-families - classification, regression and time-series forecasting, each with two variants that probe distinct agent capabilities. For clarity, we denote these variants using intuitive abbreviations: **IF** (Instruction Following) and **MM** (ML Modeling) for classification and regression; **XF** (eXogenous Features) and **CF** (Canonical Forecasting) for time-series forecasting. In classification and regression, the IF variant emphasizes instruction-following by requiring LLM to faithfully reproduce reference workflows, whereas the MM variant targets ML modeling with outcome-based evaluation. **These variants capture complementary real-world needs.** **IF** simulates a workflow where an agent must strictly execute a senior scientist’s detailed design. Conversely, **MM** reflects an outcome-driven scenario where customers only care about the final

accuracy, granting full freedom to the LLM. For time-series forecasting, the distinction between the two variants is more nuanced: in the XF variant, we retain not only the timestamp and entity identification columns but also all exogenous features from the original dataset; in the CF variant, however, while exogenous features remain available for training, the test set is constrained to only the timestamp and entity columns, making it closer to a classical forecasting setup. We partition our collection of 6,300 tasks into an approximately 95/5 train/test split, designating the most recently updated tasks as the test set. Table 2 summarizes the dataset scale and the primary capability assessed in each task type. Tool schema and task examples are shown in Appendix I.

3.1 DATASET CURATION



Figure 2: **Automated pipeline of DARE-bench.** The construction process consists of four stages: (1) *Dataset Sourcing*, where Kaggle datasets are filtered by tags, license, size, and metadata; (2) *Task Design*, where schema summaries, targets, features, and feasibility are analyzed with the help of LLM; (3) *Post-Process*, including splitting, noise injection for IF tasks or resampling or entity checks for time-series-CF tasks; and (4) *Finalization*, which validates solvability in a sandbox for IF tasks and produces standardized benchmark artifacts.

To construct DARE-bench, we design an automated data curation pipeline that systematically transforms raw Kaggle datasets into standardized machine learning tasks. Unlike prior benchmarks which rely mainly on manual curation, our approach integrates web crawling, LLM-based task formulation, controlled data transformations, and sandbox verification to ensure both quality and scale. Shown in Figure 2, the pipeline consists of four stages. Detailed prompts are shown in Appendix G.

Dataset Sourcing with Augmented Metadata. We selected Kaggle as the primary data source due to its breadth of real-world, user-contributed datasets. The official API of Kaggle retrieves candidate datasets that meet specific criteria including tabular format and valid open license. Additionally, we develop a lightweight web crawler to extract additional data from webpage descriptions that were present in the dataset, providing additional metadata elements to the LLM through column previews and natural-language descriptions which help the model understand the context of the task formulation.

LLM-Assisted Task Design and Feasibility Analysis. For each sourced candidate dataset, we employ an LLM to assess whether it can support a well-posed predictive task. The model receives both the dataset preview and the detailed description to duplicate expert assessment on a large scale. The LLM detects a target column which can be either categorical or continuous for classification and regression tasks along with structured features and their corresponding data types. For time-series forecasting tasks, the model detects timestamp columns and numerical targets that evolve through time and exogenous features in addition to identifying the temporal frequency of the data. Only datasets deemed feasible by this automated analysis proceed to the next stage.

Post-Process. Feasible datasets are then transformed into uniform benchmarking tasks. The data is split randomly into training and testing sets. For instruction-following tasks, controlled noise is injected into roughly twenty percent of the training data, which simulates real-world data quality issues through numerical values that exceed valid ranges and unexpected categorical entries, and the testing set serves as the clean reference data. The chronological split method is used for time-series forecasting to preserve the natural order of time in the data. LLM then detects entity identifiers to stop data leakage between groups and it performs automatic resampling of irregular time series data to uniform intervals through an aggregation method suggested by the model.

Finalization. After the post-process step, for instruction-following tasks, the validation process for each task runs independently in a sandbox environment by executing the reference solution code sequence including data loading, preprocessing, training, and prediction generation. Since these tasks rely on reference outputs rather than fixed ground truth values, the sandbox ensures that the instructions can be faithfully executed and the generated predictions are fully reproducible under the same random seed. In contrast, ML modeling tasks directly use ground-truth values (e.g., class

216

217

Table 3: Distribution of task domains across the DARE-bench train and test sets.

218

219

Dataset	Finance	Health	Business	Technology	Automotive	Education	Environment	Others
Train	16.9%	10.2%	7.3%	4.0%	4.5%	2.8%	6.8%	47.5%
Test	17.1%	8.4%	8.2%	5.6%	3.3%	3.1%	2.4%	51.9%

220

221

222 labels or numerical targets) for evaluation and do not require sandbox execution. Finally, the task is
 223 packaged into a standardized format that includes training and testing data, metadata describing the
 224 dataset and task, the natural language task description, and the corresponding reference.
 225

226

227

3.2 TASK FORMULATION

228

229

230 **Input and output.** Suppose we have the task description Q , an accompanying dataset description
 231 M , a training set $\mathcal{D}_{\text{train}} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{n_{\text{train}}}$, a testing set without target values $\mathcal{D}_{\text{test}} = \{\mathbf{x}_i\}_{i=1}^{n_{\text{test}}}$, and
 232 access to a code execution tool \mathcal{T} . The tool \mathcal{T} enforces a maximum wall-clock runtime T_{\max} , while
 233 the agent \mathcal{G} is subject to an interaction budget of K turns. Given these inputs and constraints, \mathcal{G}
 234 produces executable code \mathcal{C} , which is run within \mathcal{T} on $\mathcal{D}_{\text{train}}$ to fit a model and subsequently on $\mathcal{D}_{\text{test}}$
 235 to generate predictions $\hat{\mathbf{y}}$, i.e., $\hat{\mathbf{y}} = \mathcal{G}(Q, \mathcal{D}_{\text{train}}, \mathcal{D}_{\text{test}}, M, \mathcal{T}(T_{\max}, K))$.

236

237

238 **Evaluation metrics.** We evaluate models differently depending on the task type. For instruction-
 239 following tasks (i.e., Classification-IF and Regression-IF), we compare the model’s generated pre-
 240 diction $\hat{\mathbf{y}}$ against the simulated reference output \mathbf{y}_{ref} obtained from the reference solution code \mathcal{C}_{ref} ,
 241 and assign a score of 1 if $\hat{\mathbf{y}} = \mathbf{y}_{\text{ref}}$ and 0 otherwise. For ML modeling tasks, including Classification-
 242 MM, Regression-MM, and both Time-series-XF and Time-series-CF, we directly compare the model
 243 predictions $\hat{\mathbf{y}}$ against the masked ground-truth values \mathbf{y}_{gt} . Specifically, we adopt the macro-F1 score
 244 for classification-MM tasks to account for class imbalance, and use the clipped coefficient of deter-
 245 mination for regression and time-series forecasting, defined as $\text{clip}(R^2) = \min\{1, \max\{0, R^2\}\}$.
 246 For tasks with multiple prediction targets, the evaluation metric is computed by averaging over all
 247 targets. Details of our reference solution code can be found in Appendix H and calculation of
 248 macro-F1 and R^2 can be found in Appendix D.

249

250

3.3 FEATURES OF DARE-BENCH

251

252

253 DARE-bench introduces several key features that distinguish it from prior benchmarks in data sci-
 254 ence and machine learning:

255

256

257 **ML Modeling and Instruction Following.** DARE-bench differs from other existing benchmarks
 258 because it assesses two fundamental data science capabilities which are essential for real-world ap-
 259 plications: ML modeling and task instruction following for data processing and model development.

260

261

262 **Verifiable Ground Truth.** The evaluation process of DARE-bench depends on actual labels and
 263 simulated reference solution outputs to produce results that can be replicated. The system design
 264 removes all dependencies on human judgment and model-based assessments that enables evaluation
 265 metrics to directly assess task performance. This design is similar to coding benchmarks such as
 266 SWE-bench (Jimenez et al., 2024) and math benchmarks like AIME (Balunović et al., 2025), making
 267 it extremely suitable for supervised fine-tuning (SFT) and reinforcement learning with verifiable
 268 rewards (RLVR).

269

270 **Dual Role as Evaluation and Training Resource.** The benchmark offers a training dataset which
 271 enables users to perform model fine-tuning and alignment. As we will demonstrate in Section 5,
 272 the models trained on DARE-bench achieve better results than their baselines, which proves that the
 273 dataset serves as a benchmark and a resource to improve data science LLMs.

274

275

276 **Diversity, Realism, and Practical Constraints.** Our datasets are created from Kaggle sources,
 277 making them naturally diverse, multilingual, and spanning various domains while capturing real-
 278 world challenges such as class imbalance, missing values, and noise. **As illustrated in Table 3,**
 279 **quantitative analysis confirms this broad coverage, showing that DARE-bench spans a wide spec-
 280 trum of real-world verticals across both training and test sets. Details in categorization can be found
 281 in Appendix K.** In addition, enforced constraints—such as a 10-minute execution limit and bounded

270

271 Table 4: Hyperparameter sensitivity analysis for o4-mini across different turns and sandbox maxi-
272 mum execution time limit configurations.

273

turns	time	class-IF	class-MM	reg-IF	reg-MM	time-XF	time-CF
3	300	37.16	55.44	29.71	51.69	37.99	6.67
5	200	67.56	57.89	53.62	57.60	42.29	9.67
6	180	73.42	61.07	63.76	60.92	41.59	9.79
8	120	73.87	61.42	65.21	61.05	42.11	8.82
10	100	75.22	63.36	62.31	62.07	42.03	10.97
15	100	76.80	65.88	66.66	62.41	40.03	9.92

278

279 Table 5: Main evaluation results on our benchmark (test tasks) under the configuration where turns
280 set as 5 and sandbox maximum execution time set as 200 s. The best score in each column is bolded.

281

Model	class-IF	class-MM	reg-IF	reg-MM	time-XF	time-CF
gpt-4o	32.88	40.45	20.28	40.60	35.54	4.77
gpt-4.1	55.82	57.83	52.17	58.62	40.78	6.60
gpt-5	69.81	43.40	57.24	56.29	36.83	10.13
gpt-o4-mini	67.56	57.89	53.62	57.60	42.29	9.67
Claude-Sonnet-3.7	61.48	61.03	46.37	63.20	49.88	13.70
Claude-Sonnet-4	16.21	18.27	15.21	11.33	4.80	0.01
Qwen3-32B	17.11	30.71	15.21	35.86	26.96	0.00
Qwen3-4B	3.60	5.23	0.72	3.29	6.97	0.00

289

290 tool invocation turns—reflect realistic user expectations for efficient, accurate solutions. See more
291 details on Appendix F.

292

293

4 EVALUATION

294

295 In this section, we present the experimental results and analysis of several LLMs evaluated using
296 DARE-bench.

297

298

299

4.1 EXPERIMENT SETTINGS

300

301 We experiment with state-of-the-art LLMs from open-source ones such as Qwen3-32B and Qwen3-
302 4B (Yang et al., 2025), to proprietary models such as gpt-o4-mini (OpenAI, 2025d), gpt-4o, gpt-
303 4.1 (OpenAI, 2025a), gpt-5 (OpenAI, 2025b), Claude-Sonnet-3.7 (Anthropic, 2025a), and Claude-
304 Sonnet-4 (Anthropic, 2025b).

305

306 For all the experiments, we employ a greedy decoding strategy whenever applicable, along with
307 sandbox (ByteDance-Seed Foundation Code Team, 2024) for code execution. To reduce random-
308 ness, each task is repeated three times and we report the average score. We evaluate all tasks using
309 either accuracy or the macro-F1/clipped R^2 score. The ‘classification-IF’ and ‘regression-IF’ met-
310 rics are measured using a strict, binary (0/1) accuracy. ‘classification-MM’ is measured using a
311 graded (0.0-1.0) macro-F1 score. The remaining metrics, ‘regression-MM’, ‘time-series-XF’, and
312 ‘time-series-CF’, are all evaluated using the clipped R^2 score.

313

314 We conduct our evaluation in two stages. First, we perform a sensitivity analysis on the key hy-
315 perparameters for our evaluation framework using one of the most advanced models, gpt-o4-mini,
316 specifically turns and sandbox maximum execution time. These limits are set to simulate real-world
317 applications, as a user would not wait infinite time for an agent to complete a task. Our goal is to find
318 a balanced configuration. Second, with this configuration, we conduct a comprehensive comparison
319 of several leading LLMs on our benchmark.

320

321

4.2 HYPERPARAMETER SENSITIVITY ANALYSIS

322

323 The results, shown in Table 4, clearly indicate a clear trend emerges: performance generally im-
324 proves with a higher number of interactive turns. We observe a dramatic leap in performance when
325 moving from 3-turn configurations to 5-turn configurations. For example, the classification-IF score
326

324
 325 **Table 6: Average number of tokens and tool calls for completions of different models and prompts.**
 326 **All token counts are calculated using the Qwen3 tokenizer.**

Model	IF Tokens	IF Tool Calls	MM Tokens	MM Tool Calls	Overall Tokens	Overall Tool Calls
prompt	596.7	-	224.6	-	350.7	-
gpt-5	609.5	2.2	582.5	2.4	591.2	2.4
Claude-Sonnet-3.7	675.2	3.6	894.3	4.8	830.0	4.4
Qwen3-32B	2093.3	3.1	1693.0	3.5	1816.8	3.4
Qwen3-32B-SFT-DV	1778.3	3.3	1572.0	3.6	1638.7	3.5
Qwen3-4B	1691.1	3.7	1151.7	3.9	1328.1	3.8
Qwen3-4B-RL	1549.4	3.7	1140.1	3.7	1277.9	3.7

334
 335 jumps from 37.16 (at 3 turns, 300 s) to 67.56 (at 5 turns, 200 s). This suggests that allowing the
 336 agent more opportunities to iterate and refine its approach is crucial.
 337

338 The highest performance on classification-IF (76.80) was achieved at the (15 turns, 100 s) setting.
 339 However, for our main model comparison, we sought a balance between performance and computa-
 340 tional efficiency (i.e., cost and latency). We selected the (5 turns, 200 s) configuration as our standard
 341 setting. This configuration (5 turns, 200 s) serves as a robust and practical baseline; it significantly
 342 outperforms 3-turn setups and achieves strong, representative scores across all metrics (e.g., 67.56
 343 on classification-IF, 53.62 on regression-IF, and 42.29 on time-series-XF) within a practical time
 344 constraint, i.e., approximately 1000 s user wait time in total.
 345

346 4.3 MODEL COMPARISON

347 Based on our sensitivity analysis, we adopt the (5 turns, 200 s) configuration for a comprehensive
 348 comparison of all models. The main results are presented in Table 5. **Statistics on the average token**
 349 **usage and the number of tool invocations are listed in Table 6.**

350 In this standardized setting, Claude-Sonnet-3.7 emerges as the top-performing model. It achieves the
 351 highest scores on four of the six evaluation metrics: ‘classification-IF’ (69.81), ‘classification-MM’
 352 (61.03), ‘regression-MM’ (63.20), ‘time-series-XF’ (49.88) and ‘time-series-CF’ (13.70), demon-
 353 strating its strong overall capabilities for this benchmark. gpt-5 leads the two IF columns, achieving
 354 the highest ‘classification-IF’ (69.81) and ‘regression-IF’ (57.24).
 355

356 The results also reveal marked disparities between model generations. Claude-Sonnet-4 under-
 357 performs significantly compared to its predecessor Claude-Sonnet-3.7, with notably weaker scores
 358 across all metrics. A key reason is that Claude-Sonnet-4 tends to decompose tasks into very fine-
 359 grained substeps, executing almost every small operation separately. As a result, completing a single
 360 benchmark task often requires a very large number of steps, and the model nearly always exceeds
 361 the allowed step limit, leading to premature failures. Meanwhile, among the open-source models,
 362 Qwen3-32B and Qwen3-4B perform far below the proprietary models, struggling in all categories
 363 and failing entirely on time-series-CF. This highlights that complex, multi-step data analysis in sand-
 364 boxed environments remains a considerable challenge for current open-source LLMs.

365 Moving beyond the quantitative scores in Table 5 to understand why models fail on our benchmark,
 366 we conducted a systematic qualitative analysis of failed trajectories. Our goal is to identify the
 367 primary bottlenecks and limitations of current SOTA agents.
 368

Incorrect Tool Argument Passing. A fundamental failure mode observed was that LLMs often
 369 failed to correctly interface with the code-execution tool. While the generated Python code was
 370 logically correct, they frequently mismatched tool parameters (e.g., forgetting to pass filenames),
 371 causing execution to fail before code could run. Definition of our tool can be found in Appendix I.
 372

Instruction Following Failures. LLMs often ignored explicit constraints: processing steps in the
 373 wrong order, skipping required transformations, or omitting critical function arguments (Figure 3).
 374 These errors show weak adherence to task specifications.

375 **Flawed Reasoning in Open-Ended Tasks.** More subtle problems came from brittle reasoning.
 376 Common issues included misuse of metadata (hard-coding values), risky preprocessing (e.g., naive
 377

378
379
380
381
382
383
384
385
386
387
388
389

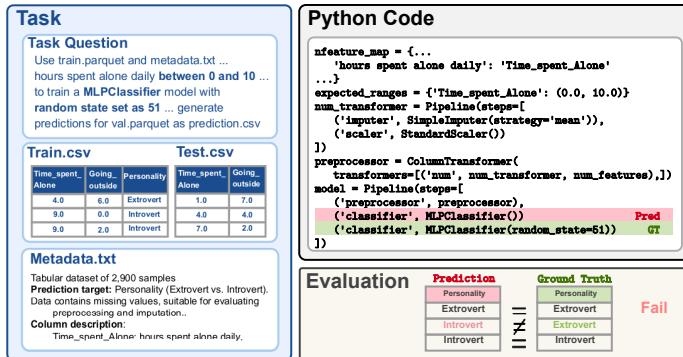


Figure 3: Example of an instruction-following task where the agent fails to respect explicit constraints. Despite being asked to fix the random seed, the model omitted the required argument, leading to incorrect predictions and an evaluation failure.

Table 7: Fine-tuning and RL improve performance over baselines. Superscripts denote absolute gains compared to the baseline of the same model.

Model	Setting	class-IF	class-MM	reg-IF	reg-MM	time-XF	time-CF	Total	Model-Perf
Qwen3-32B	Baseline	17.11	30.71	15.21	35.86	26.96	0.00	23.25	65.03
Qwen3-32B	SFT-FV	40.54 ^{+23.43}	44.71 ^{+13.99}	42.75 ^{+27.54}	49.21 ^{+13.35}	39.95 ^{+12.99}	0.07 ^{+0.07}	42.42 ^{+19.17}	72.32 ^{+7.29}
Qwen3-32B	SFT-AV	40.54 ^{+23.43}	47.20 ^{+16.49}	42.02 ^{+26.81}	55.56 ^{+19.70}	33.56 ^{+6.60}	0.00 ^{+0.00}	42.91 ^{+19.72}	70.27 ^{+5.24}
Qwen3-32B	SFT-BV	38.06 ^{+20.95}	48.91 ^{+18.20}	42.75 ^{+27.54}	54.55 ^{+18.69}	35.91 ^{+8.95}	0.00 ^{+0.00}	42.83 ^{+19.58}	71.01 ^{+5.98}
Qwen3-32B	SFT-DV	38.58 ^{+21.47}	43.82 ^{+13.11}	39.13 ^{+23.92}	51.00 ^{+15.14}	38.92 ^{+11.96}	0.00 ^{+0.00}	41.12 ^{+17.18}	71.68 ^{+6.65}
Qwen3-4B	Baseline	3.60	5.23	0.72	3.29	6.97	0.00	4.39	54.18
Qwen3-4B	RL	38.96 ^{+35.36}	39.44 ^{+34.21}	31.88 ^{+31.16}	37.04 ^{+33.75}	32.28 ^{+25.31}	2.28 ^{+2.28}	37.40 ^{+33.01}	62.55 ^{+8.37}

label encoding, mishandling NaNs), and unreliable type inference. Such shortcuts led to fragile pipelines and frequent errors.

Time-Series Task Failures. Performance on ‘time-series-CF’ was especially poor. Reflecting a lack of exposure to complex time-series reasoning, LLMs often failed to produce valid output formats or relied on trivial heuristics (last value, mean), resulting in near-zero predictive accuracy.

This qualitative analysis reveals that current agent failures are multi-faceted. They range from basic API misuse and poor instruction following to, most critically, a lack of robust, generalizable reasoning for complex tasks. The widespread use of brittle preprocessing and the near-total failure on complex time-series formatting suggest that current agents, while proficient at simple code generation, still lack the deep, domain-specific reasoning required for autonomous data science.

5 FINE-TUNING LLMs WITH DARE-BENCH

To further strengthen the performance of foundation LLMs on DARE-bench, we explore two complementary training paradigms: supervised fine-tuning (SFT) and reinforcement learning (RL). SFT leverages curated supervision from rejection-sampled traces to align models more closely with task requirements, while RL directly optimizes models with verifiable outcome rewards. The following subsections detail each approach, their implementation, and the improvements they yield.

Rejection Sampling and Supervised Fine-tuning. To obtain high-quality supervision signals, we rejection-sample traces generated across multiple runs, using task-specific filtering strategies.

We generate data for supervised fine-tuning through rejection sampling using task-independent filters that evaluate trajectories for *validity*, *quality*, and *speed*. A trajectory is *valid* if it achieves exact match for IF tasks or exceeds a type-specific score threshold for predictive tasks. A task is considered *diverse* if its sampled runs contain both successes and failures (IF) or if the variance of its

432

433 Table 8: *Ablation study on the impact of Instruction Following (IF) and ML Modeling (MM) data*
434 *with SFT-DV rejection sampling data.*

435

Train Data	class-IF	class-MM	reg-IF	reg-MM	time-XF	time-CF
baseline	17.11	30.71	15.21	35.86	26.96	0.00
IF	40.99 ^{+23.88}	22.38 ^{-8.33}	47.82 ^{+32.61}	27.85 ^{-8.01}	23.83 ^{-3.13}	0.00 ^{+0.00}
MM	11.71 ^{-5.40}	45.69 ^{+14.98}	18.84 ^{+3.63}	45.38 ^{+9.52}	34.12 ^{-7.16}	0.00 ^{+0.00}
IF+MM	38.58 ^{+21.47}	43.82 ^{+13.11}	39.13 ^{+23.92}	51.00 ^{+15.14}	38.92 ^{+11.96}	0.00 ^{+0.00}

440

441

442 Table 9: External validation on DSBench (Jing et al., 2024) after converting tasks into our format.
443 Superscripts denote absolute gains over the baseline of the same model.

444

Model	Setting	Competition-level Accuracy
Qwen3-32B	Baseline	32.38
Qwen3-32B	SFT-FV	37.82 ^{+5.44}
Qwen3-32B	SFT-AV	41.08 ^{+8.70}
Qwen3-32B	SFT-BV	40.06 ^{+7.68}
Qwen3-32B	SFT-DV	42.41 ^{+10.03}
Qwen3-4B	Baseline	18.23
Qwen3-4B	RL	40.00 ^{+21.77}

451

452

453 predictive scores exceeds a threshold. We study four strategies: **FV** (Fastest-Valid), which keeps the
454 quickest valid trace for each task; **AV** (All-Valid), which retains all valid traces; **BV** (Best-Valid),
455 which for diverse tasks selects the single best valid trace; and **DV** (Duo-Valid), which for diverse
456 tasks retains the top-2 valid traces (fastest for IF, highest-scoring above the mean for predictive).
457 Both IF and predictive tasks use their natural evaluation metrics (exact match or macro-F1 / clipped
458 R^2) to define validity and rank trajectories. More implementation details are provided in Appendix J.

459

460 **Reinforcement Learning.** We perform reinforcement learning with GRPO (Shao et al., 2024) on
461 Qwen3-4B (Yang et al., 2025) using the DARE-Bench training tasks with the verl (Sheng et al.,
462 2025) framework. During training, we found that the group normalization used in GRPO intro-
463 duces training instability. Therefore, we chose to remove the normalization component similar to
464 Dr.GRPO (Liu et al., 2025), which mitigates the training stability issue. Moreover, we use sequence-
465 level aggregation as in the original GRPO, rather than token-level aggregation used by DAPO (Yu
466 et al., 2025). Additional training details can be found in Appendix E.

467

468 **Fine-tuning Results.** Table 7 summarizes results for both SFT (Qwen3-32B) and RL (Qwen3-
469 4B). Specifically, **Model-Perf** measures the quality of the model’s predictions by focusing solely
470 on successful attempts for MM tasks. This metric isolates the quality dimension from the validity
471 dimension, confirming that fine-tuning improves the model’s actual data science proficiency, not just
472 its adherence to syntax rules. Across IF and MM tasks, fine-tuning yields substantial improvements
473 over the baseline, with absolute gains of nearly $1.83 \times$ in total score and 10% in ModelPerf. Different
474 strategies bring complementary benefits: AV yields the strongest overall improvements for MM
475 tasks, while FV favors IF tasks. Reinforcement learning on Qwen3-4B provides even larger relative
476 gains, boosting the total score from 4.39 to 37.40 and ModelPerf from 54.18 to 62.55. These results
477 confirm that DARE-bench not only improves instruction following but also translates into better
478 downstream modeling accuracy once correct predictions are generated.

479

480 **Impact of Data Composition.** As shown in Table 8, we use SFT-DV to further investigate the spe-
481 cific contributions of IF and MM data through an ablation study. Training exclusively on MM data
482 boosts predictive modeling performance but degrades instruction adherence, while training solely
483 on IF data shows the inverse. Only the combined approach successfully integrates both capabili-
484 ties, achieving a robust balance. This confirms that process-oriented and outcome-oriented tasks are
485 complementary and essential for a comprehensive data science agent.

486

487 **Failure Analysis.** Shown in Table 10, we categorized incorrect trajectories to identify specific
488 reasoning bottlenecks. Proprietary models mainly face problems with Code Errors, while open-
489 source baselines frequently exceed execution limits because of inefficient exploration. Training on
490 DARE-bench effectively mitigates these issues; notably, RL on Qwen3-4B reduces code errors by

486

487

Table 10: Failure mode analysis across different models.

488

489

Model	Inst Adhere	Code Error	Code Exec Limit	Max Token Limit
gpt-5	126	333	0	0
Claude-Sonnet-3.7	158	262	0	0
Qwen3-32B	48	106	257	372
Qwen3-32B + SFT-DV	43	80	236	256
Qwen3-4B	79	174	661	102
Qwen3-4B + RL	49	91	331	119

494

495

Table 11: Comparison between Native Function Call² and DataWiseAgent on DARE-bench and DSBench.

496

497

Framework	Model	class-IF	class-MM	reg-IF	reg-MM	time-XF	time-CF	DSBench
Native Function Call	Qwen3-32B	17.11	30.71	15.21	35.86	26.96	0.0	32.38
Native Function Call	Qwen3-32B + SFT-DV	38.58	43.82	39.13	51.00	38.92	0.0	42.41
DataWiseAgent	Qwen3-32B	21.62	29.63	34.78	34.40	30.45	0.0	29.17

502

503

48 percent and halves code execution limit errors, demonstrating that our supervision significantly enhances both code correctness and efficiency.

504

505

Case Studies of Fine-tuning Effects. To further illustrate the benefits of fine-tuning, we highlight two representative failure modes that were substantially reduced. First, a common pre-fine-tuning error occurred when LLM provided tool with incorrectly generated tool arguments. The code executor tool requires three explicit arguments including code, input file and output file. However, LLMs frequently generated correct Python code that opened files but failed to pass the filename into the tool’s `file_to_load` argument, causing sandbox execution to fail. After fine-tuning, the frequency of such mismatches decreased remarkably. Second, the baseline models tried to use natural-language column names from the task description without checking the provided `metadata.txt` which led to `KeyErrors`. The first step of the fine-tuned models involved examining the metadata file for references to actual column identifiers which led to the development of reliable executable solutions.

516

517

External Validation and Comparison. To further assess generalization and compare with state-of-the-art specialized agents, we adapt data modeling tasks from DSBench (Jing et al., 2024) into the DARE-bench task format. As shown in Table 9, all fine-tuned versions outperform the original baseline, proving that DARE-bench enhances performance beyond in-domain tasks. Specifically, inclusive sampling methods (AV and DV) yield the most significant improvements by leveraging a wider range of valid traces compared to stricter filtering (FV and BV). Furthermore, we compare our fine-tuned models with DataWiseAgent (You et al., 2025) under identical settings. As detailed in Table 11, our model compare favorably to DataWiseAgent, achieving a score of 42.41 compared to 29.17. This demonstrates that our framework offers competitive adaptability and robustness in diverse data science workflows compared to existing specialized agents.

525

526

6 CONCLUSION AND FUTURE WORKS

527

We present DARE-bench, a training-focused benchmark for DS agents which enables executable evaluation and trainable supervision through two verifiable task families: (i) process-aware instruction following with reference-code ground truths, and (ii) ML modeling with dataset ground truths. The 6,300 Kaggle-derived tasks show poor performance from strong general-purpose LLMs until they receive task-specific data but fine-tuning on DARE-bench artifacts produce reliable and repeatable enhancements in process fidelity and predictive performance and execution failure reduction. Our design uses the executable-benchmark approach which software engineering professionals have adopted to solve DS-specific problems that recent evaluations have identified.

536

537

We will expand our task type coverage (figures/speeches/clustering), strengthen procedural constraints and verifier-based objectives, and add anomaly detection tracks (tabular and time-series) with appropriate event/segment-level metrics and weak/unsupervised scoring protocols.

²https://qwen.readthedocs.io/en/latest/framework/function_call.html

540 REFERENCES
541

- 542 Anthropic. Claude sonnet 3.7. <https://cloud.google.com/vertex-ai/generative-ai/docs/partner-models/clause>, 2025a. Extended thinking mode,
543 step-by-step reasoning capability.
- 544 Anthropic. Introducing claude sonnet 4. <https://www.anthropic.com/news/clause-4>,
545 2025b. Improved coding and reasoning, available via API / Bedrock / Vertex.
- 546 Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan,
547 Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language
548 models. *arXiv preprint arXiv:2108.07732*, 2021.
- 549 Mislav Balunović, Jasper Dekoninck, Ivo Petrov, Nikola Jovanović, and Martin Vechev. Matharena:
550 Evaluating llms on uncontaminated math competitions. *arXiv preprint arXiv:2505.23281*, 2025.
- 551 Tommaso Bendinelli, Artur Dox, and Christian Holz. Exploring llm agents for cleaning tabular
552 machine learning datasets. *arXiv preprint arXiv:2503.06664*, 2025.
- 553 ByteDance-Seed Foundation Code Team. Sandboxfusion: A multi-language code sandbox exe-
554 cution tool for evaluating code generation models. <https://github.com/bytedance/SandboxFusion>, 2024. Used in FullStack Bench: Evaluating LLMs as Full Stack Coders.
- 555 Ruisheng Cao, Fangyu Lei, Haoyuan Wu, Jixuan Chen, Yeqiao Fu, Hongcheng Gao, Xinzhuang
556 Xiong, Hanchong Zhang, Wenjing Hu, Yuchen Mao, et al. Spider2-v: How far are multimodal
557 agents from automating data science and engineering workflows? *Advances in Neural Infor-
558 mation Processing Systems*, 37:107703–107744, 2024.
- 559 Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio
560 Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, et al. Mle-bench: Evaluating machine learn-
561 ing agents on machine learning engineering. *arXiv preprint arXiv:2410.07095*, 2024.
- 562 Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared
563 Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large
564 language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.
- 565 Nan Chen, Yuge Zhang, Jiahang Xu, Kan Ren, and Yuqing Yang. Viseval: A benchmark for data
566 visualization in the era of large language models. *IEEE Transactions on Visualization and Com-
567 puter Graphics*, 2024.
- 568 Alex Egg, Martin Iglesias Goyanes, Friso Kingma, Andreu Mora, Leandro von Werra, and Thomas
569 Wolf. Dabstep: Data agent benchmark for multi-step reasoning. *arXiv preprint arXiv:2506.23719*,
570 2025a.
- 571 Alex Egg, Martin Iglesias Goyanes, Friso Kingma, Andreu Mora, Leandro von Werra, and Thomas
572 Wolf. Dabstep: Data agent benchmark for multi-step reasoning, 2025b. URL <https://arxiv.org/abs/2506.23719>.
- 573 Antoine Grosnit, Alexandre Maraval, James Doran, Giuseppe Paolo, Albert Thomas, Refinath
574 Shahul Hameed Nabeezath Beevi, Jonas Gonzalez, Khyati Khandelwal, Ignacio Iacobacci, Ab-
575 delhakim Benechehab, et al. Large language models orchestrating structured reasoning achieve
576 kaggle grandmaster level. *arXiv preprint arXiv:2411.03562*, 2024.
- 577 Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu,
578 Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms
579 via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.
- 580 Siyuan Guo, Cheng Deng, Ying Wen, Hechang Chen, Yi Chang, and Jun Wang. Ds-agent: Au-
581 tomated data science by empowering large language models with case-based reasoning. *arXiv
582 preprint arXiv:2402.17453*, 2024.
- 583 Sirui Hong, Yizhang Lin, Bang Liu, Bangbang Liu, Binhao Wu, Ceyao Zhang, Chenxing Wei,
584 Danyang Li, Jiaqi Chen, Jiayi Zhang, et al. Data interpreter: An llm agent for data science. *arXiv
585 preprint arXiv:2402.18679*, 2024.

- 594 Qian Huang, Jian Vora, Percy Liang, and Jure Leskovec. Mlagentbench: Evaluating language agents
 595 on machine learning experimentation. *arXiv preprint arXiv:2310.03302*, 2023.
- 596
- 597 Yiming Huang, Jianwen Luo, Yan Yu, Yitong Zhang, Fangyu Lei, Yifan Wei, Shizhu He, Lifu
 598 Huang, Xiao Liu, Jun Zhao, et al. Da-code: Agent data science code generation benchmark for
 599 large language models. *arXiv preprint arXiv:2410.07331*, 2024.
- 600 Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik
 601 Narasimhan. Swe-bench: Can language models resolve real-world github issues?, 2024. URL
 602 <https://arxiv.org/abs/2310.06770>.
- 603
- 604 Liqiang Jing, Zhehui Huang, Xiaoyang Wang, Wenlin Yao, Wenhao Yu, Kaixin Ma, Hongming
 605 Zhang, Xinya Du, and Dong Yu. Dsbench: How far are data science agents from becoming data
 606 science experts? *arXiv preprint arXiv:2409.07703*, 2024.
- 607 Ram Mohan Rao Kadiyala, Siddhant Gupta, Jebish Purbey, Giulio Martini, Suman Debnath, and
 608 Hamza Farooq. Dsbc: Data science task benchmarking with context engineering. *arXiv preprint*
 609 *arXiv:2507.23336*, 2025.
- 610 Yuhang Lai, Chengxi Li, Yiming Wang, Tianyi Zhang, Ruiqi Zhong, Luke Zettlemoyer, Wen-tau
 611 Yih, Daniel Fried, Sida Wang, and Tao Yu. Ds-1000: A natural and reliable benchmark for data
 612 science code generation. In *International Conference on Machine Learning*, pp. 18319–18345.
 613 PMLR, 2023.
- 614
- 615 Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee,
 616 and Min Lin. Understanding r1-zero-like training: A critical perspective. *arXiv preprint*
 617 *arXiv:2503.20783*, 2025.
- 618 OpenAI. Introducing GPT-4.1 in the api. <https://openai.com/index/gpt-4-1/>, April
 619 2025a. Lower-latency, cheaper GPT model family optimized for instruction following and long
 620 context.
- 621
- 622 OpenAI. Introducing GPT-5. <https://openai.com/index/introducing-gpt-5/>, Au-
 623 gust 2025b. Latest flagship model with improved code and agentic performance.
- 624 OpenAI. Introducing openai o3 and o4-mini. <https://openai.com/index/introducing-o3-and-o4-mini/>, April 2025c.
- 625
- 626 OpenAI. Openai o3 and o4-mini system card. <https://cdn.openai.com/pdf/2221c875-02dc-4789-800b-e7758f3722c1/o3-and-o4-mini-system-card.pdf>, April 2025d. System card describing capa-
 627 bilities, evaluations, and safety considerations.
- 628
- 629
- 630
- 631 Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro,
 632 Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can
 633 teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36:68539–
 634 68551, 2023.
- 635
- 636 Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang,
 637 Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathemati-
 638 cal reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.
- 639
- 640 Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng,
 641 Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. In *Proceedings*
 642 *of the Twentieth European Conference on Computer Systems*, pp. 1279–1297, 2025.
- 643
- 644 An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu,
 645 Chang Gao, Chengan Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint*
 646 *arXiv:2505.09388*, 2025.
- 647
- Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao.
 React: Synergizing reasoning and acting in language models. In *International Conference on
 Learning Representations (ICLR)*, 2023.

- 648 Pengcheng Yin, Wen-Ding Li, Kefan Xiao, Abhishek Rao, Yeming Wen, Kensen Shi, Joshua How-
649 land, Paige Bailey, Michele Catasta, Henryk Michalewski, et al. Natural language to code gener-
650 ation in interactive data science notebooks. *arXiv preprint arXiv:2212.09248*, 2022.
651
- 652 Ziming You, Yumiao Zhang, Dexuan Xu, Yiwei Lou, Yandong Yan, Wei Wang, Huaming Zhang,
653 and Yu Huang. Datawiseagent: A notebook-centric llm agent framework for automated data
654 science. *arXiv preprint arXiv:2503.07044*, 2025.
655
- 656 Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian
657 Fan, Gaohong Liu, Lingjun Liu, et al. Dapo: An open-source llm reinforcement learning system
658 at scale. *arXiv preprint arXiv:2503.14476*, 2025.
659
- 660 Dan Zhang, Sining Zhoubian, Min Cai, Fengzu Li, Lekang Yang, Wei Wang, Tianjiao Dong, Ziniu
661 Hu, Jie Tang, and Yisong Yue. Datascibench: An llm agent benchmark for data science. *arXiv*
662 *preprint arXiv:2502.13897*, 2025.
663
- 664 Lei Zhang, Yuge Zhang, Kan Ren, Dongsheng Li, and Yuqing Yang. Mlcopilot: Unleash-
665 ing the power of large language models in solving machine learning tasks. *arXiv preprint*
666 *arXiv:2304.14979*, 2023.
667
- 668 Yuge Zhang, Qiyang Jiang, Xingyu Han, Nan Chen, Yuqing Yang, and Kan Ren. Benchmarking
669 data science agents. *arXiv preprint arXiv:2402.17168*, 2024.
670
- 671 Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Livia Sun, Jeff Huang, Cody Hao Yu, Shiyi
672 Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. Sqlang: Efficient execution of
673 structured language model programs. *Advances in neural information processing systems*, 37:
674 62557–62583, 2024.
675
- 676
- 677
- 678
- 679
- 680
- 681
- 682
- 683
- 684
- 685
- 686
- 687
- 688
- 689
- 690
- 691
- 692
- 693
- 694
- 695
- 696
- 697
- 698
- 699
- 700
- 701

702

703

704

OVERVIEW OF THE APPENDIX

705

707 The Appendix is organized as follows:

708

- Appendix A contains reproducibility statement.
- Appendix B contains the use of LLMs in this work.
- Appendix C provides the limitation of the work.
- Appendix D contains the explanation of the evaluation metrics used in this work.
- Appendix E provides training details of the RL experiments in this work.
- Appendix F contains detailed description of DARE-bench features.
- Appendix G provides example prompt of the preprocessing steps of this work, including column inference and task identification.
- Appendix H contains reference code for instruction-following tasks in this work.
- Appendix I contains tool schema used in our experiments and some task examples.
- Appendix J provides details of the rejection sampling implementation of this work.
- Appendix K provides details on how we make the use of LLM to classify tasks.

723

724

A REPRODUCIBILITY STATEMENT

725

726

We have attached the subset of our test set in the supplementary materials. Once accepted, we will release the full test set of our benchmark. The training set and model checkpoints will also be provided upon request, and we plan to release them publicly depending on the feedback we receive from the research community. Also, a detailed description of our data processing procedure is included in subsection 3.1. These resources are intended to facilitate reproducibility and allow future researchers to build upon our work.

732

733

B THE USE OF LARGE LANGUAGE MODELS (LLMs)

734

735

In this project, LLMs were used as assistive tools. Specifically, we used LLMs to polish the writing of the paper and to assist in finding related works. In addition, LLMs were used during the data processing stage, for tasks such as data filtering, question rewriting, and identifying task targets. Beyond these uses, the research ideas, experimental design, and analyses were developed independently by the authors. The authors take full responsibility for all content presented in this paper.

740

741

C LIMITATIONS

742

743

While DARE-bench provides a large-scale, verifiable, and trainable benchmark, several limitations remain. First, the current tasks are primarily tabular based, so the benchmark does not yet cover multimodal inputs such as text–image combinations or code–diagram interactions. Second, the cost of generating large numbers of executable traces can be high, and the rejection sampling strategies, while effective, may introduce biases toward shorter trajectories.

748

749

D EVALUATION METRICS

750

751

We report results using two standard metrics for classification and regression tasks: macro-F1 and R^2 .

753

754

755

756 **Macro-F1.** For a classification task with C classes, let TP_c , FP_c , and FN_c denote the number of
 757 true positives, false positives, and false negatives for class c , respectively. The precision and recall
 758 for class c are defined as

$$760 \quad \text{Precision}_c = \frac{\text{TP}_c}{\text{TP}_c + \text{FP}_c}, \quad \text{Recall}_c = \frac{\text{TP}_c}{\text{TP}_c + \text{FN}_c}.$$

$$761$$

762 The F1-score for class c is

$$763 \quad \text{F1}_c = \frac{2 \cdot \text{Precision}_c \cdot \text{Recall}_c}{\text{Precision}_c + \text{Recall}_c}.$$

$$764$$

765 The macro-F1 is then the unweighted mean across all classes:

$$766 \quad \text{Macro-F1} = \frac{1}{C} \sum_{c=1}^C \text{F1}_c.$$

$$767$$

$$768$$

$$769$$

770 **R^2 (Coefficient of Determination).** For regression/time-series tasks with ground-truth values
 771 $\{y_i\}_{i=1}^n$ and predictions $\{\hat{y}_i\}_{i=1}^n$, define the mean of ground-truth values as $\bar{y} = \frac{1}{n} \sum_{i=1}^n y_i$. The R^2
 772 metric is

$$773 \quad R^2 = 1 - \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{\sum_{i=1}^n (y_i - \bar{y})^2}.$$

$$774$$

775 An R^2 value close to 1 indicates strong predictive performance, while values close to 0 or negative
 776 indicate weak or worse-than-baseline performance. Since R^2 can be negative when the model
 777 performs worse than predicting the mean, we adopt a *clipped* R^2 defined as

$$778 \quad R_{\text{clipped}}^2 = \max(R^2, 0),$$

$$779$$

780 to ensure that regression scores remain in $[0, 1]$ and are comparable to classification metrics.

782 E REINFORCEMENT LEARNING TRAINING DETAILS

784 E.1 REWARD DESIGN

786 **Instruction following tasks.** For instruction following tasks including Classification-IF and
 787 Regression-IF tasks. We have reference solution code \mathcal{C}_{ref} with corresponding simulated prediction
 788 for data $\mathcal{D}_{\text{test}}$ as $\mathbf{y}_{\text{ref}} = \mathcal{C}_{\text{ref}}(\mathcal{D}_{\text{test}})$. Given the model prediction $\hat{\mathbf{y}} = \mathcal{G}(Q, \mathcal{D}_{\text{train}}, \mathcal{D}_{\text{test}}, M, \mathcal{T})$
 789 and simulated ground truth \mathbf{y}_{ref} , we use the following reward:

$$790 \quad r = \begin{cases} 0.1, & \hat{\mathbf{y}} \text{ exists,} \\ 1.1, & \hat{\mathbf{y}} = \mathbf{y}_{\text{ref}}, \\ 0, & \text{otherwise.} \end{cases} \quad (1)$$

$$791$$

$$792$$

$$793$$

$$794$$

795 Note that LLMs may be unable to generate a `prediction.csv` file due to the max turns or
 796 sandbox execution time limit.

797 **Predictive ML tasks.** For other tasks, including classification-PM, regression-PM, time-series-
 798 XF, and time-series-CF, we have masked ground-truth data \mathbf{y}_{gt} . Given the prediction provided by
 799 LLM $\hat{\mathbf{y}}$, we define the reward as

$$801 \quad r = \begin{cases} 0.1 + d(\hat{\mathbf{y}}, \mathbf{y}_{\text{gt}}), & \hat{\mathbf{y}} \text{ exists,} \\ 0, & \text{otherwise,} \end{cases} \quad (2)$$

$$802$$

$$803$$

804 where $d : \mathcal{X} \times \mathcal{Y} \rightarrow [0, 1]$ denotes the distance measure between the prediction and the target. For
 805 classification tasks, we adopt the macro-F1 score to account for class imbalance. For regression and
 806 time-series tasks, we use the *clipped coefficient of determination*, defined as

$$807 \quad \text{clip}(R^2) = \min\{1, \max\{0, R^2\}\}.$$

$$808$$

809 If there are multiple prediction targets, we compute the distance by taking the average of them.

810 E.2 OTHER TRAINING PARAMETERS
811812 **Reinforcement learning.** We summarize our RL training hyper-parameters in Table 12.
813

814 Hyper-parameter	815 Value
816 RL algorithm	816 GRPO (Shao et al., 2024)
817 Loss aggregation	817 Sequence level
818 Group normalization	818 False
819 Learning rate	819 1×10^{-6}
820 Training mini-batch size	820 64
821 KL regularization	821 False
822 Rollout batch size	822 64
823 Number of rollouts per question	823 8
824 Rollout backend	824 SGLang (Zheng et al., 2024)
825 Rollout temperature	825 1.0
826 top_p	826 0.95
827 top_k	827 50
828 Model sequence length	828 32,768

829 Table 12: Hyper-parameters used for reinforcement learning experiments.
830831 F DETAILED DESCRIPTION OF OTHER DARE-BENCH FEATURES
832833 **Automated and Scalable Curation.** The task generation process in DARE-bench uses a defined
834 approach which collects data from Kaggle and incorporates web-scraped content before LLMs verify
835 the tasks and produce standardized definitions. The automated pipeline generates authentic work
836 assignments at large scale across multiple fields through an approach that needs minimal human
837 involvement.
838839 **Diverse and Realistic Coverage.** The benchmark contains 6,300 tasks which cover multiple domains
840 and languages, including tabular classification and regression as well as advanced time-series
841 forecasting. By drawing directly from real-world Kaggle datasets, it naturally incorporates common
842 data challenges such as class imbalance, missing values, noise, and temporal irregularities, providing
843 a more faithful simulation of practical data science scenarios.
844845 **Time and interaction constraints.** DARE-bench implements realistic usage scenarios through
846 its requirement for both time-limited wall-clock operation and restricted interaction turn counts.
847 In practice, end users are unlikely to wait hours for a model to train a full pipeline; hence, we
848 cap execution time to 10 minutes for fast-response settings. The system limits the total number of
849 agent-environment dialogues which forces models to find efficient solutions instead of performing
850 endless exploration. The established limitations in this benchmark create a testing environment
851 which mirrors actual operational conditions for interactive data science agents.
852853
854
855
856
857
858
859
860
861
862
863

864 **G EXAMPLE PROMPT FOR COLUMN INFERENCE AND TASK IDENTIFICATION**
865866 The following prompt guides the model to check task suitability and identify prediction target and
867 relevant features from the provided dataset description and data information.
868869 Task Target and Feature Identification Prompt
870871 You are given a dataset along with its description.
872 Your tasks are as follows:873 **Task 1: Assess Logistic/Linear Regression Suitability**
874875 Determine whether logistic regression (for classification) or linear regression (for regression) can be
876 appropriately used to model this dataset.
877 Use this strict checklist:

- 878
- 879 • Classification: target must be categorical; features structured; manageable missing values.
 - 879 • Regression: target must be numeric and continuous; features structured; manageable missing
880 values; categorical targets not allowed.

881 If all conditions are met, the method is appropriate. Otherwise, it is not. When uncertain, output False.
882883 **Task 2: Identify Task Type, Target Column, and Feature Columns**
884885 You must select column names only from the list below inside *Column list*:; avoid using names from
886 *Context / description*:
887888 Column list:
889 {all_columns}
890891 Context / description:
892 {filtered_metadata}
893 {scraper}

894 Infer:

- 895
- 896 • The task type (classification or regression)
 - 897 • A list (≤ 3) of candidate target columns
 - 898 • The best set of feature columns

899 **Task 3: Column Type Inference**
900

901 For each column in the list, classify it as:

- 902
- 903 • “numerical”: meaningful arithmetic operations
 - 904 • “categorical”: groups/codes, arithmetic not meaningful

905 Instructions:

- 906
- 907 • Return a Python dictionary with every column as a key
 - 908 • Value must be either “numerical” or “categorical”
 - 909 • Use dataset description to guide decisions

910 **Final Output Format**
911

912 Output exactly 5 lines, in LaTeX-boxed format:

- 913 1. Method suitability → \boxed{True} or \boxed{False}
-
- 914 2. Task type → \boxed{classification} or \boxed{regression}
-
- 915 3. Target column candidates → \boxed{["target1", "target2"]}
-
- 916 4. Feature columns → \boxed{["col1", "col2", ...]}
-
- 917 5. Column types → \boxed{\{"col1": "numerical", "col2": "categorical"\}}

918 The following prompt reformulates the user question into a precise and well-structured instruction.
 919

920 Rewrite Question Prompt
 921

922 You are given a machine learning task described in `final_question` and a dictionary of column
 923 metadata in `metadata_description`. Your job is to rewrite the `final_question` in fluent
 924 natural language, making it easier to read while keeping the meaning and structure intact.
 925 Here's what you must do:

- 926 1. Replace all column names and feature names in `final_question` with their natural lan-
 927 guage descriptions from `metadata_description`. Preserve the original ordering of
 928 features in lists.
 929 2. If a column or feature name is not present in `metadata_description`, rewrite it into a
 930 natural-sounding phrase using best judgment.
 931 3. Rewrite structured formats (lists, dicts) into natural language paragraphs, while retaining
 932 original item order.
 933 4. Keep existing natural language unchanged.
 934 5. Keep all file paths unchanged.
 935 6. File names or paths must be wrapped in backticks.
 936 7. Target column names must be wrapped in backticks.
 937 8. Final output must be a clear instruction in natural language.
 938 9. If the string `None` appears in value ranges, treat it as a categorical value `None`.
 939 10. Do not include headings, markdown, or extra explanations—return only the rewritten ques-
 940 tion.
 941 11. Use only standard English characters.
 942 12. Explicitly preserve ordering requirements in the rewritten question.

943 - - -

944 Here is the `final_question`:

945 {question}

946 - - -

947 Here is the `metadata_description`:

948 {description}

949 - - -

950 Now return only the rewritten version of the question, using natural language descriptions where pos-
 951 sible. Preserve file paths, model names, and categorical values exactly as given.

952

953

954

955

956

957

958

959

960

961

962

963

964

965

966

967

968

969

970

971

972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025

The following prompt determines whether the dataset is time-series and infers the appropriate temporal type information.

Time-Series Identification and Typing Prompt

You are given a dataset and its description.

Task 1: Assess Suitability

Check if Time Series Forecasting applies. Conditions:

1. Must have a clear timestamp column (e.g., 'date', 'time').
2. Must have a target variable changing over time (e.g., sales, temperature).
3. Observations should be sequential and time-dependent.
4. Time interval must be regular or resample-able (e.g., daily, hourly).

If all are met, output True; otherwise False.

Task 2: Identify Key Columns

From the list below, infer:

- Best **timestamp column**
- Best **target column**
- Optional exogenous feature columns

Column list:
{all_columns}

Context:
{filtered_metadata}
{scraper}

Preview (first 50 rows):
{df_preview}

Task 3: Column Typing

For each column, classify as: "timestamp", "numerical", "categorical", or "other".
If timestamp exists, also infer its format (Python strftime).

Final Output Format (6 lines, LaTeX-boxed):

1. Suitability → \boxed{\text{True}} or \boxed{\text{False}}
2. Timestamp column → \boxed{\text{column_name}} or \boxed{\text{ambiguous}}
3. Target column → \boxed{\text{column_name}} or \boxed{\text{ambiguous}}
4. Exogenous features → \boxed{\{ "col1", ... \}} or \boxed{\{ \}}
5. Column types → \boxed{\{ "col1": "timestamp", "col2": "numerical" \}}
6. Time format → \boxed{\%Y-\%m-\%d \%H:\%M:\%S} or \boxed{\text{ambiguous}}

1026
 1027 The following prompt identifies grouping entities (e.g., users, products, or regions) that structure the
 1028 dataset for time-CF tasks.

1029 **Entity (Group) Identification Prompt**

1030
 1031 You are given a dataset for time series forecasting. The dataset includes a timestamp column, a tar-
 1032 get column to be predicted, and possibly multiple other columns representing categorical or numeric
 1033 features. Your task is to identify which column(s) represent the entity (or group ID) — that is, the
 1034 column(s) that differentiate multiple independent time series within the dataset.

1035 Please analyze the column names and a sample of the data (including at least the first few rows), and
 1036 answer the following:

- 1036 1. Which column(s) should be used to distinguish different time series entities?
 - 1037 2. Briefly explain why those column(s) were selected as entity identifiers.
 - 1038 3. If no entity column is needed because the dataset represents a single time series, say so
 1039 explicitly.
- 1040
 1041
Dataset Description
 1042 {description}
 1043
 1044
 1045
Additional identification suggestions (optional)
 1046 {entity_identification_suggestions}
 1047
 1048
Sample Data (first 30 rows)
 1049 {sample_str}
 1050
 1051
 1052
Column statistics (distinct counts, top value frequency, example value patterns)
 1053 {column_stats_str}
 1054
 1055
 1056
Output format (exactly 2 lines, LaTeX-boxed, nothing else):
- 1057 1. Entity Columns → $\boxed{\{ \text{col_name1}, \text{col_name2}, \dots \}}$ or
 $\boxed{\{ \}} \text{ if none}$
 - 1058 2. Justification → $\boxed{\text{<Your explanation here>}}$

1062
 1063
 1064
 1065
 1066
 1067
 1068
 1069
 1070
 1071
 1072
 1073
 1074
 1075
 1076
 1077
 1078
 1079

1080
 1081 The following prompt decides whether resampling is needed for the dataset and, if so, specifies the
 1082 appropriate resampling strategy.
 1083

1084 **Resampling Decision Prompt**

1085 You are an expert data scientist assisting with time series preprocessing.
 1086 Your goal is to decide whether a given time series dataset needs **resampling** (i.e., converting irregular
 1087 or overly fine-grained timestamps to a fixed frequency like daily/hourly).

1088 **Task Description**

1089 You are given:

- 1090 1. A **brief description** of the dataset and task.
- 1091 2. The **first 30 rows** of the dataset (including timestamps and relevant columns).
 - 1092 • For each time value, up to 5 rows are shown.
 - 1093 • If a time value had more than 5 rows, it is marked with a comment.
- 1094 3. The **target column** for forecasting or analysis: `{target_col}`.

1095 Please analyze:

- 1096 • Whether the time column appears **irregular, too granular, or dense**.
- 1097 • Whether each row represents a **meaningful unit** (e.g., per-day summary) or a **low-level log**
 (e.g., events).
- 1098 • Whether **resampling** could make the series easier to model.
- 1099 • If resampling is needed, recommend:
 - 1100 – The **resampling rule** (e.g., 1min, 5min, 1H, 1D).
 - 1101 – The **aggregation function** for the target column (`{target_col}`): choose from
 "mean", "sum", "count", or other common aggregations.
- 1102 • If resampling is not needed, it may be because the data is evenly spaced or each row is
 meaningful as-is.

1103 **Dataset Description**

1104 {description}

1105 ---

1106 **Sample Data (first 30 rows)**

1107 {sample_str}

1108 ---

1109 **Output format (exactly 4 lines, LaTeX-boxed, nothing else):**

- 1110 1. Should resample → `\boxed{\text{True}}` or `\boxed{\text{False}}`
- 1111 2. Reason → `\boxed{\text{<One-sentence explanation>}}`
- 1112 3. Suggested rule → `\boxed{\text{<1min/5min/1H/1D or null>}}`
- 1113 4. Target aggregation → `\boxed{\text{<mean/sum/count/... for '\{target_col\}'>}}`

1114
 1115
 1116
 1117
 1118
 1119
 1120
 1121
 1122
 1123
 1124
 1125
 1126
 1127
 1128
 1129
 1130
 1131
 1132
 1133

1134
1135

H REFERENCE CODE FOR INSTRUCTION-FOLLOWING EVALUATION

1136
1137Below we include the reference implementation used to evaluate instruction-following tasks in our
benchmark.

1138

1139

Reference Code for Instruction-Following Evaluation

1140

1141

```

import json
import os
import sqlite3
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression, LinearRegression, Lasso
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor
from sklearn.neighbors import KNeighborsClassifier, KNeighborsRegressor
from sklearn.neural_network import MLPClassifier, MLPRegressor
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import LinearSVC
from sklearn.multioutput import MultiOutputClassifier
from sklearn.metrics import accuracy_score, classification_report, mean_squared_error,
r2_score

# Function to load and join tables from a SQLite file
def load_and_join(sqlite_path):
    conn = sqlite3.connect(sqlite_path)
    tables = pd.read_sql_query("SELECT name FROM sqlite_master WHERE type='table';",
                               conn)[['name']].tolist()
    df = None
    for table in tables:
        df_tab = pd.read_sql_query(f"SELECT * FROM '{table}'", conn)
        if 'row_id' not in df_tab.columns:
            continue
        if df is None:
            df = df_tab
        else:
            df = df.merge(df_tab, on='row_id', how='inner')
    conn.close()
    return df

def train_predict_model(train_df, eval_df, feature_cols, model_type,
                       column_type_inference, target_cols=['answer'], imputer_type="most_frequent",
                       problem_type="classification", random_state=42):
    print(f" MACHINE LEARNING PIPELINE")
    print("=" * 60)

    # Check if target column exists in training data
    for target_col in target_cols:
        if target_col not in train_df.columns:
            print(f"Target column '{target_col}' not found in training data!")
            print(f"Available columns in train_df: {list(train_df.columns)}")
            return None

    # Check if all feature columns exist in both datasets
    missing_features_train = [col for col in feature_cols if col not in train_df.
                               columns]
    missing_features_eval = [col for col in feature_cols if col not in eval_df.columns]

    if missing_features_train:
        print(f"Missing feature columns in train_df: {missing_features_train}")
        return None

    if missing_features_eval:
        print(f"Missing feature columns in eval_df: {missing_features_eval}")
        return None

    formatted_targets = ", ".join("'{col}'".format(col) for col in target_cols)
    print(f"Target column {formatted_targets} found in training data")
    print(f" Training dataset shape: {train_df.shape}")
    print(f" Evaluation dataset shape: {eval_df.shape}")

    # Prepare training features and target

```

```

1188
1189     X_train = train_df[feature_cols].copy()
1190     y_train = train_df[target_cols].copy()
1191
1192     # Prepare evaluation features (and target if it exists)
1193     X_eval = eval_df[feature_cols].copy()
1194
1195     # Check if target column exists in eval_df for evaluation
1196     has_eval_target = True
1197     for target_coloumn in target_cols:
1198         if target_coloumn not in eval_df.columns:
1199             has_eval_target = False
1200     if has_eval_target:
1201         y_eval = eval_df[target_cols].copy()
1202         print(f"Target column found in evaluation data - will compute metrics")
1203     else:
1204         y_eval = None
1205         print(f" Target column not found in evaluation data - will only make
1206               predictions")
1207
1208     # Check for null targets in training data
1209     null_targets_train = y_train.isnull().sum().sum()  # use two sum to get the total
1210             null number
1211     if null_targets_train > 0:
1212         print(f" Found {null_targets_train} null targets in training data - removing
1213             these rows")
1214         valid_indices = ~y_train.isnull().any(axis=1)  # make sure no null target row
1215         X_train = X_train[valid_indices]
1216         y_train = y_train[valid_indices]
1217
1218     print(f" Final training data: {X_train.shape[0]} rows")
1219     print(f" Final evaluation data: {X_eval.shape[0]} rows")
1220
1221     # Separate numeric and categorical features
1222     numeric_features = []
1223     categorical_features = []
1224
1225     for col in feature_cols:
1226         # Check data type in training data
1227         if column_type_inference[col].lower() == "numerical":
1228             numeric_features.append(col)
1229         elif column_type_inference[col].lower() == "categorical":
1230             categorical_features.append(col)
1231
1232     print(f" Numeric features ({len(numeric_features)}): {numeric_features}")
1233     print(f" Categorical features ({len(categorical_features)}): {categorical_features}")
1234     assert len(numeric_features) + len(categorical_features) == len(feature_cols)
1235
1236     # Create preprocessing pipeline
1237     transformers = []
1238
1239     if numeric_features:
1240         transformers.append('num', Pipeline([
1241             ('imputer', SimpleImputer(strategy=imputer_type)),
1242             ('scaler', StandardScaler())
1243         ]), numeric_features)
1244
1245     if categorical_features:
1246         transformers.append('cat', Pipeline([
1247             ('imputer', SimpleImputer(strategy="most_frequent")),
1248             ('onehot', OneHotEncoder(handle_unknown='ignore', sparse_output=False))
1249         ]), categorical_features)
1250
1251     preprocessor = ColumnTransformer(transformers=transformers)
1252
1253     # Choose model based on problem type
1254     if problem_type.lower() == "classification":
1255         if model_type == "LogisticRegression":
1256             model = LogisticRegression(random_state=random_state)
1257             if len(target_cols) > 1:
1258                 model = MultiOutputClassifier(model)
1259         elif model_type == "DecisionTreeClassifier":
1260             model = DecisionTreeClassifier(random_state=random_state)
1261         elif model_type == "GaussianNB":
1262             model = GaussianNB()
1263             if len(target_cols) > 1:
1264                 model = MultiOutputClassifier(model)
1265         elif model_type == "LinearSVC":
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2689
2690
2691
2692
2693
2694
2695
2696
2697
2698
2699
2700
2701
2702
2703
2704
2705
2706
2707
2708
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2749
2750
2751
2752
2753
2754
2755
2756
2757
2758
2759
2760
2761
2762
2763
2764
2765
2766
2767
2768
2769
2770
2771
2772
2773
2774
2775
2776
2777
2778
2779
2780
2781
2782
2783
2784
2785
2786
2787
2788
2789
2790
2791
2792
2793
2794
2795
2796
2797
2798
2799
2800
2801
2802
2803
2804
2805
2806
2807
2808
2809
2810
2811
2812
2813
2814
2815
2816
2817
2818
2819
2820
2821
2822
2823
2824
2825
2826
2827
2828
2829
2830
2831
2832
2833
2834
2835
2836
2837
2838
2839
2840
2841
2842
2843
2844
2845
2846
2847
2848
2849
2850
2851
2852
2853
2854
2855
2856
2857
2858
2859
2860
2861
2862
2863
2864
2865
2866
2867
2868
2869
2870
2871
2872
2873
2874
2875
2876
2877
2878
2879
2880
2881
2882
2883
2884
2885
2886
2887
2888
2889
2890
2891
2892
2893
2894
2895
2896
2897
2898
2899
2900
2901
2902
2903
2904
2905
2906
2907
2908
2909
2910
2911
2912
2913
2914
2915
2916
2917
2918
2919
2920
2921
2922
2923
2924
2925
2926
2927
2928
2929
2930
2931
2932
2933
2934
2935
2936
2937
2938
2939
2940
2941
2942
2943
2944
2945
2946
2947
2948
2949
2950
2951
2952
2953
2954
2955
2956
2957
2958
2959
2960
2961
2962
2963
2964
2965
2966
2967
2968
2969
2970
2971
2972
2973
2974
2975
2976
2977
2978
2979
2980
2981
2982
2983
2984
2985
2986
2987
2988
2989
2990
2991
2992
2993
2994
2995
2996
2997
2998
2999
2999

```

```

1242
1243     model = LinearSVC(random_state=random_state)
1244     if len(target_cols) > 1:
1245         model = MultiOutputClassifier(model)
1246     elif model_type == "MLPClassifier":
1247         model = MLPClassifier(random_state=random_state)
1248     else:
1249         raise ValueError(f"Invalid model type: {model_type}")
1250     print(f" Using {model_type} for classification")
1251
1252     elif problem_type.lower() == "regression":
1253         if model_type == "LinearRegression":
1254             model = LinearRegression()
1255         elif model_type == "DecisionTreeRegressor":
1256             model = DecisionTreeRegressor(random_state=random_state)
1257         elif model_type == "Ridge":
1258             model = Ridge(random_state=random_state)
1259         elif model_type == "Lasso":
1260             model = Lasso(random_state=random_state)
1261         elif model_type == "MLPRegressor":
1262             model = MLPRegressor(random_state=random_state)
1263         else:
1264             raise ValueError(f"Invalid model type: {model_type}")
1265
1266         print(f" Using {model_type} for regression")
1267
1268     # Create full pipeline
1269     ml_pipeline = Pipeline([
1270         ('preprocessor', preprocessor),
1271         ('model', model)
1272     ])
1273
1274     # Train the model
1275     print(f"TRAINING MODEL...")
1276     try:
1277         ml_pipeline.fit(X_train, y_train)
1278         print(f" Model trained successfully")
1279     except Exception as e:
1280         print(f" Error during training: {e}")
1281     return None
1282
1283     # Make predictions
1284     print(f"MAKING PREDICTIONS...")
1285     try:
1286         y_pred = ml_pipeline.predict(X_eval)
1287         print(f"Predictions completed")
1288     except Exception as e:
1289         print(f"Error during prediction: {e}")
1290     return None
1291
1292     # Evaluate model (only if we have evaluation targets)
1293     if has_eval_target and y_eval is not None:
1294         print(f"ODEL EVALUATION")
1295         print("=" * 30)
1296
1297         # Remove rows with null targets in evaluation for metrics
1298         valid_eval_mask = y_eval.notna()
1299         y_eval_clean = y_eval[valid_eval_mask]
1300         y_pred_clean = y_pred[valid_eval_mask]
1301
1302         if len(y_eval_clean) == 0:
1303             print(" No valid evaluation targets found - skipping evaluation metrics")
1304         else:
1305             if problem_type.lower() == "classification":
1306                 accuracy = accuracy_score(y_eval_clean, y_pred_clean)
1307                 print(f" Accuracy: {accuracy:.4f}")
1308                 print(f"Classification Report:")
1309                 print(classification_report(y_eval_clean, y_pred_clean))
1310
1311                 # Show sample predictions
1312                 print(f"Sample Predictions:")
1313                 for i in range(min(10, len(y_eval_clean))):
1314                     actual = y_eval_clean.iloc[i]
1315                     predicted = y_pred_clean[i]
1316                     status = "right" if actual == predicted else "wrong"
1317                     print(f" {status} Row {i}: Actual={actual}, Predicted={predicted}")
1318
1319             else:
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2689
2690
2691
2692
2693
2694
2695
2696
2697
2698
2699
2700
2701
2702
2703
2704
2705
2706
2707
2708
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2749
2750
2751
2752
2753
2754
2755
2756
2757
2758
2759
2760
2761
2762
2763
2764
2765
2766
2767
2768
2769
2770
2771
2772
2773
2774
2775
2776
2777
2778
2779
2780
2781
2782
2783
2784
2785
2786
2787
2788
2789
2790
2791
2792
2793
2794
2795
2796
2797
2798
2799
2800
2801
2802
2803
2804
2805
2806
2807
2808
2809
2810
2811
2812
2813
2814
2815
2816
2817
2818
2819
2820
2821
2822
2823
2824
2825
2826
2827
2828
2829
2830
2831
2832
2833
2834
2835
2836
2837
2838
2839
2840
2841
2842
2843
2844
2845
2846
2847
2848
2849
2850
2851
2852
2853
2854
2855
2856
2857
2858
2859
2860
2861
2862
2863
2864
2865
2866
2867
2868
2869
2870
2871
2872
2873
2874
2875
2876
2877
2878
2879
2880
2881
2882
2883
2884
2885
2886
2887
2888
2889
2890
2891
2892
2893
2894
2895
2896
2897
2898
2899
2900
2901
2902
2903
2904
2905
2906
2907
2908
2909
2910
2911
2912
2913
2914
2915
2916
2917
2918
2919
2920
2921
2922
2923
2924
2925
2926
2927
2928
2929
2930
2931
2932
2933
2934
2935
2936
2937
2938
2939
2940
2941
2942
2943
2944
2945
2946
2947
2948
2949
2950
2951
2952
2953
2954
2955
2956
2957
2958
2959
2960
2961
2962
2963
2964
2965
2966
2967
2968
2969
2970
2971
2972
2973
2974
2975
2976
2977
2978
2979
2980
2981
2982
2983
2984
2985
2986
2987
2988
2989
2990
2991
2992
2993
2994
2995
2996
2997
2998
2999
2999

```

```

1296
1297     mse = mean_squared_error(y_eval_clean, y_pred_clean)
1298     r2 = r2_score(y_eval_clean, y_pred_clean)
1299     rmse = np.sqrt(mse)
1300
1301     print(f" R^2 Score: {r2:.4f}")
1302     print(f" RMSE: {rmse:.4f}")
1303     print(f" MSE: {mse:.4f}")
1304
1305     # Show sample predictions
1306     print(f"Sample Predictions:")
1307     for i in range(min(10, len(y_eval_clean))):
1308         actual = y_eval_clean.iloc[i]
1309         predicted = y_pred_clean[i]
1310         diff = abs(actual - predicted)
1311         print(f" Row {i}: Actual={actual:.3f}, Predicted={predicted:.3f}, Diff={diff:.3f}")
1312     else:
1313         print(f"EVALUATION SKIPPED - No target column in evaluation data")
1314         print(f" Generated {len(y_pred)} predictions")
1315
1316     y_pred_df = pd.DataFrame(y_pred, columns=y_train.columns)
1317     y_pred_df.insert(0, 'row_id', eval_df["row_id"].values)
1318     return {
1319         'pipeline': ml_pipeline,
1320         'predictions': y_pred_df,
1321         'eval_indices': X_eval.index,
1322         'problem_type': problem_type,
1323         'X_train': X_train,
1324         'y_train': y_train,
1325         'X_eval': X_eval,
1326         'y_eval': y_eval if has_eval_target else None,
1327         'has_eval_target': has_eval_target,
1328         "numeric_features": numeric_features,
1329         "categorical_features": categorical_features,
1330     }
1331
1332     feature_cols = $feature_cols
1333     model_type = "$model_type"
1334     column_type_inference = $column_type_inference
1335     target_cols = $target_cols
1336     imputer_type = "$imputer_type"
1337     problem_type = "$problem_type"
1338     random_state = $random_state
1339     save_file_type = "$save_file_type"
1340
1341     if save_file_type == 'sqlite':
1342         conn = sqlite3.connect("train_v1_no_err.sqlite")
1343         train_df = pd.read_sql("SELECT * FROM train_set", conn)
1344         train_df = train_df.replace({None: np.nan})
1345         conn.close()
1346
1347         eval_df = load_and_join("val_v1.sqlite")
1348         eval_df = eval_df.replace({None: np.nan})
1349     elif save_file_type == 'csv':
1350         train_df = pd.read_csv('train_v1_no_err.csv', keep_default_na=False, na_values
1351                               =[""])
1352         eval_df = pd.read_csv('val_v1.csv', keep_default_na=False, na_values=[""])
1353     elif save_file_type == 'parquet':
1354         train_df = pd.read_parquet('train_v1_no_err.parquet')
1355         train_df = train_df.replace({None: np.nan})
1356         eval_df = pd.read_parquet('val_v1.parquet')
1357         eval_df = eval_df.replace({None: np.nan})
1358
1359     result = train_predict_model(
1360         train_df = train_df,
1361         eval_df = eval_df,
1362         feature_cols = feature_cols,
1363         model_type = model_type,
1364         column_type_inference=column_type_inference,
1365         target_cols=target_cols,
1366         imputer_type=imputer_type,
1367         problem_type=problem_type,
1368         random_state=random_state
1369     )
1370
1371     result['predictions'].to_csv('simulated_pred_local.csv', index=False)
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2298
2299
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2388
2389
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2398
2399
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2438
2439
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2448
2449
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2478
2479
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2489
2490
2491
2492
2493
2494
2495
2496
2497
2497
2498
2498
2499
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2538
2539
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2548
2549
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2578
2579
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2589
2590
2591
2592
2593
2594
2595
2596
2597
2597
2598
2598
2599
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2638
2639
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2648
2649
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2678
2679
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2689
2689
2690
2691
2692
2693
2694
2695
2696
2697
2697
2698
2698
2699
2699
2700
2701
2702
2703
2704
2705
2706
2707
2708
2709
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2738
2739
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2748
2749
2749
2750
2751
2752
2753
2754
2755
2756
2757
2758
2759
2759
2760
2761
2762
2763
2764
2765
2766
2767
2768
2769
2769
2770
2771
2772
2773
2774
2775
2776
2777
2778
2778
2779
2779
2780
2781
2782
2783
2784
2785
2786
2787
2788
2789
2789
2790
2791
2792
2793
2794
2795
2796
2797
2797
2798
2798
2799
2799
2800
2801
2802
2803
2804
2805
2806
2807
2808
2809
2809
2810
2811
2812
2813
2814
2815
2816
2817
2818
2819
2819
2820
2821
2822
2823
2824
2825
2826
2827
2828
2829
2829
2830
2831
2832
2833
2834
2835
2836
2837
2838
2838
2839
2839
2840
2841
2842
2843
2844
2845
2846
2847
2848
2848
2849
2849
2850
2851
2852
2853
2854
2855
2856
2857
2858
2859
2859
2860
2861
2862
2863
2864
2865
2866
2867
2868
2869
2869
2870
2871
2872
2873
2874
2875
2876
2877
2878
2878
2879
2879
2880
2881
2882
2883
2884
2885
2886
2887
2888
2888
2889
2889
2890
2891
2892
2893
2894
2895
2896
2897
2897
2898
2898
2899
2899
2900
2901
2902
2903
2904
2905
2906
2907
2908
2909
2909
2910
2911
2912
2913
2914
2915
2916
2917
2918
2919
2919
2920
2921
2922
2923
2924
2925
2926
2927
2928
2929
2929
2930
2931
2932
2933
2934
2935
2936
2937
2938
2938
2939
2939
2940
2941
2942
2943
2944
2945
2946
2947
2948
2948
2949
2949
2950
2951
2952
2953
2954
2955
2956
2957
2958
2959
2959
2960
2961
2962
2963
2964
2965
2966
2967
2968
2969
2969
2970
2971
2972
2973
2974
2975
2976
2977
2978
2978
2979
2979
2980
2981
2982
2983
2984
2985
2986
2987
2988
2989
2989
2990
2991
2992
2993
2994
2995
2996
2997
2997
2998
2998
2999
2999
3000
3001
3002
3003
3004
3005
3006
3007
3008
3009
3009
3010
3011
3012
3013
3014
3015
3016
3017
3018
3019
3019
3020
3021
3022
3023
3024
3025
3026
3027
3028
3029
3029
3030
3031
3032
3033
3034
3035
3036
3037
3038
3038
3039
3039
3040
3041
3042
3043
3044
3045
3046
3047
3048
3048
3049
3049
3050
3051
3052
3053
3054
3055
3056
3057
3058
3059
3059
3060
3061
3062
3063
3064
3065
3066
3067
3068
3069
3069
3070
3071
3072
3073
3074
3075
3076
3077
3078
3078
3079
3079
3080
3081
3082
3083
3084
3085
3086
3087
3088
3088
3089
3089
3090
3091
3092
3093
3094
3095
3096
3097
3097
3098
3098
3099
3099
3100
3101
3102
3103
3104
3105
3106
3107
3108
3109
3110
3111
3112
3113
3114
3115
3116
3117
3118
3119
3120
3121
3122
3123
3124
3125
3126
3127
3128
3129
3130
3131
3132
3133
3134
3135
3136
3137
3138
3139
3140
3141
3142
3143
3144
3145
3146
3147
3148
3149
3150
3151
3152
3153
3154
3155
3156
3157
3158
3159
3160
3161
3162
3163
3164
3165
3166
3167
3168
3169
3170
3171
3172
3173
3174
3175
3176
3177
3178
3179
3179
3180
3181
3182
3183
3184
3185
3186
3187
3188
3189
3189
3190
3191
3192
3193
3194
3195
3196
3197
3198
3199
3199
3200
3201
3202
3203
3204
3205
3206
3207
3208
3209
3209
3210
3211
3212
3213
3214
3215
3216
3217
3218
3219
3219
3220
3221
3222
3223
3224
3225
3226
3227
3228
3229
3229
3230
3231
3232
3233
3234
3235
3236
3237
3238
3239
3239
3240
3241
3242
3243
3244
3245
3246
3247
3248
3248
3249
3249
3250
3251
3252
3253
3254
3255
3256
3257
3258
3259
3259
3260
3261
3262
3263
3264
3265
3266
3267
3268
3269
3269
3270
3271
3272
3273
3274
3275
3276
3277
3278
3278
3279
3279
3280
3281
3282
3283
3284
3285
3286
3287
3288
3289
3289
3290
3291
3292
3293
3294
3295
3296
3297
3297
3298
3298
3299
3299
3300
3301
3302
3303
3304
3305
3306
3307
3308
3309
3309
3310
3311
3312
3313
3314
3315
3316
3317
3318
3319
3319
3320
3321
3322
3323
3324
3325
3326
3327
3328
3329
3329
3330
3331
3332
3333
3334
3335
3336
3337
3338
3339
3339
3340
3341
3342
3343
3344
3345
3346
3347
3348
3348
3349
3349
3350
3351
3352
3353
3354
3355
3356
3357
3358
3359
3359
3360
3361
3362
3363
3364
3365
3366
3367
3368
3369
3369
3370
3371
3372
3373
3374
3375
3376
3377
3378
3378
3379
3379
3380
3381
3382
3383
3384
3385
3386
3387
3388
3388
3389
3389
3390
3391
3392
3393
3394
3395
3396
3397
3397
3398
3398
3399
3399
3400
3401
3402
3403
3404
3405
3406
3407
3408
3409
3409
3410
3411
3412
3413
3414
3415
3416
3417
3418
3419
3419
3420
3421
3422
3423
3424
3425
3426
3427
3428
3429
3429
3430
3431
3432
3433
3434
3435
3436
3437
3438
3439
3439
3440
3441
3442
3443
3444
3445
3446
3447
3448
3448
3449
3449
3450
3451
3452
3453
3454
3455
3456
3457
3458
3
```

1350 **I TOOL SCHEMA AND TASK EXAMPLES**
13511352 The following schema defines the details of our code executor tool.
1353

```

1354 Tool Schema: python-executor
1355
1356 {
1357     "type": "function",
1358     "name": "python_executor",
1359     "description": "Execute a Python script in an isolated HTTP sandbox with a 200-second
1360         time limit. Each run is single-shot and stateless (no REPL, no persistent
1361         environment between runs). You may upload input files via 'files_to_load' and
1362         retrieve results via 'files_to_save'. The maximum file size for both upload and
1363         download is 200 MB. The tool returns the full program output, including both
1364         stdout and stderr. Use explicit 'print(...)' statements to capture values in the
1365         output. This tool can be invoked up to 3 times per conversation.",
1366     "parameters": {
1367         "type": "object",
1368         "properties": {
1369             "code": {
1370                 "type": "string",
1371                 "description": "The Python code to execute."
1372             },
1373             "files_to_load": {
1374                 "type": "array",
1375                 "items": {
1376                     "type": "string"
1377                 },
1378                 "description": "List of input file paths to upload prior to execution (e.g. [\"
1379                     input1.csv\", \"config.json\"])."
1380             },
1381             "files_to_save": {
1382                 "type": "array",
1383                 "items": {
1384                     "type": "string"
1385                 },
1386                 "description": "List of output file paths to download after execution (e.g. [\"
1387                     results.csv\", \"log.txt\"])."
1388             },
1389             "required": [
1390                 "code",
1391                 "files_to_load",
1392                 "files_to_save"
1393             ],
1394             "additionalProperties": false
1395         },
1396         "strict": true
1397     }
1398
1399
1400
1401
1402
1403

```

1404 The following provides a example of IF task.
 1405

1406 Instruction Following Task Example

1408 Question

1409 Please complete the task as described below without asking any follow-up questions or requesting ad-
 1410 ditional information. Proceed under the assumption that all required information is provided. You are
 1411 given access to a training Parquet file named `train_v1.parquet`, which contains a single table,
 1412 and a metadata file `metadata.txt` that describes the original dataset and each of its columns. Your
 1413 task is to perform classification using this data and predict the target column `Personality` for the
 1414 validation set located at `val_v1.parquet`. Unless stated otherwise, you should use default parame-
 1415 ters for all steps including model training and preprocessing. First, load the training file directly. Then,
 1416 filter the training dataset using the expected ranges while retaining any rows that have missing values in
 1417 the relevant columns, excluding only those rows where a non-missing value violates its expected range.
 1418 The expected ranges are as follows in the specified order: number of close friends must be between 0.0
 1419 and 15.0; presence of stage fright must be either “No” or “Yes”; social media post frequency must be
 1420 between 0.0 and 10.0; frequency of going outside must be between 0.0 and 7.0; feeling drained after
 1421 socializing must be either “No” or “Yes”; frequency of social events must be between 0.0 and 10.0;
 1422 and hours spent alone daily must be between 0.0 and 11.0. After filtering, select only the features listed
 1423 in their original order: number of close friends, frequency of social events, presence of stage fright,
 1424 feeling drained after socializing, and hours spent alone daily. The numeric features, to be used in the
 1425 specified order, are number of close friends, frequency of social events, and hours spent alone daily,
 1426 and the categorical features, also to be used in the specified order, are presence of stage fright and feel-
 1427 ing drained after socializing. Handle missing values by imputing numeric features with the mean and
 1428 categorical features with the most frequent value. Preprocess the data by applying a standard scaler to
 1429 the numeric features and one-hot encoding to the categorical features with `handle.unknown` set to
 1430 `ignore` and `sparse_output` set to `False`. Train a single `LogisticRegression` model using
 1431 `scikit-learn` with `random_state=86`. Finally, make predictions on the validation set and save the
 1432 results to a CSV file at `prediction.csv`, including the column `row_id` as provided in the original
 1433 `val_v1.parquet` and the corresponding predictions aligned with each `row_id` so that performance
 1434 can be computed correctly.

1435 Metadata

1436 Overview

1437 *Dive into the Extrovert vs. Introvert Personality Traits Dataset, a rich collection of behavioral and
 1438 social data designed to explore the spectrum of human personality. This dataset captures key indicators
 1439 of extroversion and introversion, making it a valuable resource for psychologists, data scientists, and
 1440 researchers studying social behavior, personality prediction, or data preprocessing techniques.*

1441 Context

1442 *Personality traits like extroversion and introversion shape how individuals interact with their social
 1443 environments. This dataset provides insights into behaviors such as time spent alone, social event
 1444 attendance, and social media engagement, enabling applications in psychology, sociology, marketing,
 1445 and machine learning. Whether you’re predicting personality types or analyzing social patterns, this
 1446 dataset is your gateway to uncovering fascinating insights.*

1447 Dataset Details

1448 **Size:** The dataset contains 2,900 rows and 8 columns.

1449 Features:

- 1450 • `Time_spent_Alone`: Hours spent alone daily (0–11).
- 1451 • `Stage_fear`: Presence of stage fright (Yes/No).
- 1452 • `Social_event_attendance`: Frequency of social events (0–10).
- 1453 • `Going_outside`: Frequency of going outside (0–7).
- 1454 • `Drained_after_socializing`: Feeling drained after socializing (Yes/No).
- 1455 • `Friends_circle_size`: Number of close friends (0–15).
- 1456 • `Post_frequency`: Social media post frequency (0–10).
- 1457 • `Personality`: Target variable (Extrovert/Introvert).

1458 **Data Quality:** Includes some missing values, ideal for practicing imputation and preprocessing.

1459 **Format:** Single CSV file, compatible with Python, R, and other tools.

1460 **Data Quality Notes**

1458
1459 • Contains missing values in columns like `Time_spent_Alone` and `Going_outside`, of-
1460 ferring opportunities for data cleaning practice.
1461 • Balanced classes ensure robust model training.
1462 • Binary categorical variables simplify encoding tasks.
1463
1464 **Potential Use Cases**
1465 • Build machine learning models to predict personality types.
1466 • Analyze correlations between social behaviors and personality traits.
1467 • Explore social media engagement patterns.
1468 • Practice data preprocessing techniques like imputation and encoding.
1469 • Create visualizations to uncover behavioral trends.
1470
1471 **==== About this file ====**
1472 About this file This dataset contains 2,900 entries with 8 features related to social behavior and per-
1473 sonality traits, designed to explore and classify individuals as Extroverts or Introverts.
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511

1512 The following provides an example of ML Modeling task.
 1513

1514 **ML Modeling Task Example**

1515 **Question**

1516 Please complete the task as described below without asking any follow-up questions or requesting
 1517 additional information. Your task is to achieve good performance while balancing training time
 1518 and accuracy in the sandbox. You are provided with a processed dataset, along with a metadata
 1519 file `metadata.txt` that describes the original dataset and each of its columns. Load data in
 1520 `train_v2.parquet` as your full training set. Once that's done, using only CPU resources, train
 1521 a classification model on this training data. Then use `val_v2.parquet` to generate a prediction for
 1522 `Internet_Access` for each `row_id`. The model should output a file named `prediction.csv`.
 1523 The file must contain the column `row_id` (as provided in the original `val_v2.parquet` and the
 1524 corresponding predictions. Each prediction should be aligned with its `row_id` so that performance can
 1525 be computed correctly.

1526 **Metadata**

1527 **AI Tool Usage by Indian College Students 2025**

1528 This unique dataset, collected via a May 2025 survey, captures how 496 Indian college students use AI
 1529 tools (e.g., ChatGPT, Gemini, Copilot) in academics. It includes 16 attributes like AI tool usage, trust,
 1530 impact on grades, and internet access, ideal for education analytics and machine learning.

1531 **Columns**

- 1532 • `Student_Name`: Anonymized student name.
- 1533 • `College_Name`: College attended.
- 1534 • `Stream`: Academic discipline (e.g., Engineering, Arts).
- 1535 • `Year_of_Study`: Year of study (1–4).
- 1536 • `AI_Tools_Used`: Tools used (e.g., ChatGPT, Gemini).
- 1537 • `Daily_Usage_Hours`: Hours spent daily on AI tools.
- 1538 • `Use_Cases`: Purposes (e.g., Assignments, Exam Prep).
- 1539 • `Trust_in_AI_Tools`: Trust level (1–5).
- 1540 • `Impact_on_Grades`: Grade impact (-3 to +3).
- 1541 • `Do_Professors_Allow_Use`: Professor approval (Yes/No).
- 1542 • `Preferred_AI_Tool`: Preferred tool.
- 1543 • `Awareness_Level`: AI awareness (1–10).
- 1544 • `Willing_to_Pay_for_Access`: Willingness to pay (Yes/No).
- 1545 • `State`: Indian state.
- 1546 • `Device_Used`: Device (e.g., Laptop, Mobile).
- 1547 • `Internet_Access`: Access quality (Poor/Medium/High).

1548 Use Cases - Predict academic performance using AI tool usage. - Analyze trust in AI across streams
 1549 or regions. - Cluster students by usage patterns. - Study digital divide via 'Internet_Access'.
 1550

1551 Source: Collected via Google Forms survey in May 2025, ensuring diverse representation across India.
 1552 Note: First dataset of its kind on Kaggle!

1553

1554

1555

1556

1557

1558

1559

1560

1561

1562

1563

1564

1565

1566
1567

The following provides an example of Time Series Canonical Forecasting Task.

1568

Time Series Canonical Forecasting Task Example

1569

1570

Question

1571

Please complete the task as described below without asking any follow-up questions or requesting additional information. Your task is to achieve good performance while balancing time and accuracy in the sandbox environment. You are provided with a processed dataset, along with a metadata file `metadata.txt` that describes the original dataset and each of its columns. Load the file `train.csv` as your complete training data. This file contains the raw, non-resampled time series data. Once that's done, using only CPU resources, train time series analysis model(s) on the training data. Then, based on the file `val_v2.csv`, forecast the target column `Close`. Generate predictions exactly for each `row_id` present in the validation data. Output a file named `prediction.csv`. The file must contain the column `row_id` (as provided in the original `val_v2.csv`) and the corresponding predictions. Each prediction should be aligned with its `row_id` so that performance can be computed correctly.

1572

1573

1574

1575

1576

1577

1578

1579

Metadata

1580

1581

1582

1583

1584

What's Inside

1585

1586

1587

Why This Matters

1588

1589

1590

Apple is not just a company, it's a movement. Its stock price reflects not only financial performance, but the world's response to innovation—launches, leadership changes, economic cycles, and the occasional “one more thing.”

1591

Possibilities

1592

1593

1594

- Visualize long-term growth and volatility - Model trends, moving averages, or momentum - Forecast future prices with machine learning - Detect the impact of major product launches or events - Explore relationships between volume and price action

1595

Inspiration

1596

1597

1598

As you explore this data, don't just look for patterns—look for stories. See how moments of genius and risk-taking ripple through the numbers. Use this dataset to inspire your own creativity, your own analysis, your own ‘insanely great’ discoveries.

1599

1600

1601

Whether you're here to build a predictive model, craft beautiful visualizations, or simply marvel at the journey, remember: The people who are crazy enough to think they can change the world with data... are the ones who do.

1602

==== About this file ====

1603

About this file This file contains historical daily stock price data for Apple Inc. Each row represents one trading day and includes key financial metrics that track Apple's performance on the stock market.

1604

==== Columns & descriptions ====

1605

Date: The calendar date for the trading record (format: YYYY-MM-DD). Close: The price of Apple's stock at the end of the trading day.

1606

1607

1608

1609

1610

1611

1612

1613

1614

1615

1616

1617

1618

1619

1620 **J REJECTION SAMPLING IMPLEMENTATION DETAILS**
16211622 We sample up to $K=8$ candidate trajectories per task. Each trajectory records: (i) `final_score`
1623 and (ii) end-to-end wall-clock `time`. For IF tasks, `final_score` is exact match $\in \{0, 1\}$; for
1624 predictive tasks, `final_score` is a normalized metric such as macro-F1 or clipped R^2 .
16251626 **J.1 VALIDITY AND DIVERSITY CONDITIONS**
16271628 **Validity.** A trajectory is considered *valid* if:
1629

- 1630
- For IF tasks: `final_score` = 1.
 - For predictive tasks: `final_score` \geq type-specific threshold:
1631 class-MM: 0.8, reg-MM: 0.7, time-XF: 0.6, time-CF: 0.3.
-
- 1633

1634 **Diversity.** A task is considered *diverse* if:
1635

- 1636
- For IF tasks: among the K trials, at least one `final_score` = 1 and at least one
1637 `final_score` = 0.
 - For predictive tasks: the variance of the K scores satisfies
1639 $\text{Var}(S_i) \geq \text{threshold}$, class-MM/reg-MM: 0.15, time-XF: 0.15, time-CF: 0.1.
-
- 1641

1642 **J.2 REJECTION SAMPLING STRATEGIES**
16431644 **FV (Fastest-Valid).** For every task that has at least one valid trajectory:
1645

- 1646
- IF tasks: keep the single fastest valid trajectory.
 - Predictive tasks: keep the trajectory with the highest `final_score`.
-
- 1647

1648 **AV (All-Valid).** For every task:
1649

- 1650
- Keep all valid trajectories (as defined above).
-
- 1651

1652 **BV (Best-Valid).** For every *diverse* task:
1653

- 1654
- IF tasks: keep the single fastest valid trajectory.
 - Predictive tasks: keep the trajectory with the highest `final_score`.
-
- 1656

1657 Thus BV applies the same selection rule as FV, but restricted to diverse tasks only.
16581659 **DV (Duo-Valid).** For every *diverse* task:
1660

- 1661
- IF tasks: keep the two fastest valid trajectories (or one if fewer exist).
 - Predictive tasks: keep the top-2 trajectories by score, restricted to those with $s(t) > \bar{s}_i$
1663 (above mean).
-
- 1664

1665 **NOTES**
1666

- 1667
- FV applies to *all tasks* with valid traces; BV and DV apply only to *diverse tasks*.
 - AV is the only strategy that may return multiple valid trajectories even for non-diverse
1669 tasks.
 - FV/BV always select at most one trajectory; DV at most two; AV can return more.
 - This design ensures a balance between efficiency (FV), diversity (AV), quality (BV), and
1672 complementary coverage (DV).
-
- 1673

1674 K TASK DOMAIN CLASSIFICATION METHODOLOGY

1675
 1676 To assess the diversity of DARE-bench and verify its coverage across real-world scenarios, we clas-
 1677 sified each task into a primary domain (e.g., Finance, Health, Technology). Given the scale of the
 1678 benchmark (6,300 tasks), manual classification was infeasible. Therefore, we employed an auto-
 1679 mated LLM-based classification pipeline utilizing the rich metadata associated with each Kaggle-
 1680 derived dataset.

1681 **Metadata Usage.** The classification relies on four key metadata fields:

- 1683 • **Title:** The official name of the dataset.
- 1684 • **Subtitle:** A short phrase summarizing the dataset content.
- 1685 • **Description:** The full natural language description of the dataset context.
- 1686 • **Keywords:** User-provided tags from the original Kaggle source.

1687 **Classification Taxonomy.** To ensuring consistency, we defined a controlled vocabulary of allowed
 1688 domains based on common industry verticals: *finance, health, business, technology, automotive,*
 1689 *education, environment, and others.*

1690 **Prompt Design.** We constructed a strict prompt to instruct the LLM to identify the single best
 1691 domain label. The prompt enforces a hierarchical reasoning logic: it prioritizes explicit domain
 1692 terms found in the user-provided keywords before inferring the domain from the title or description.
 1693 The full prompt template is provided below:

1694 Domain Classification Prompt

1695 **Instruction:** Identify the single best domain for a task using the provided metadata.

1696 **Input Data:**

- 1701 • Title: {title}
- 1702 • Subtitle: {subtitle}
- 1703 • Description: {description}
- 1704 • Keywords: {keywords}

1705 **Example Domains:** [agriculture, finance, health, business, technology, automotive, education, envi-
 1706 ronment, other]

1707 **Reasoning Steps:**

- 1708 1. **Keyword Check:** First, strictly check the provided ‘keywords’ list for any explicit domain
 1709 word (e.g., ‘finance’, ‘health’). If a match is found from the allowed list, select it immedi-
 1710 ately.
- 1711 2. **Inference:** If no direct domain appears in the keywords, infer the most appropriate domain
 1712 based on the semantic context of the ‘title’, ‘subtitle’, and ‘description’.
- 1713 3. **Output Formatting:** Output exactly ONE lowercase word from the allowed domains list.
 1714 Do not output punctuation, explanations, or spaces. If the domain is uncertain or does not fit
 1715 the specific categories, output ‘other’.

1716 **Output:**

1717
 1718
 1719
 1720
 1721
 1722
 1723
 1724
 1725
 1726
 1727