

AUTOKAGGLE: A MULTI-AGENT FRAMEWORK FOR AUTONOMOUS DATA SCIENCE COMPETITIONS

Anonymous authors

Paper under double-blind review

ABSTRACT

Data science tasks involving tabular data present complex challenges that require sophisticated problem-solving approaches. We propose AutoKaggle, a powerful and user-centric framework that assists data scientists in completing daily data pipelines through a collaborative multi-agent system. AutoKaggle implements an iterative development process that combines code execution, debugging, and comprehensive unit testing to ensure code correctness and logic consistency. The framework offers highly customizable workflows, allowing users to intervene at each phase, thus integrating automated intelligence with human expertise. Our universal data science toolkit, comprising validated functions for data cleaning, feature engineering, and modeling, forms the foundation of this solution, enhancing productivity by streamlining common tasks. We selected 8 Kaggle competitions to simulate data processing workflows in real-world application scenarios. Evaluation results demonstrate that AutoKaggle achieves a validation submission rate of 0.85 and a comprehensive score of 0.82 in typical data science pipelines, fully proving its effectiveness and practicality in handling complex data science tasks.¹

1 INTRODUCTION

In recent years, with the rapid development of large language models (LLMs) (OpenAI, 2022; 2023), automated data science has gradually become possible. LLM-based agents have shown great potential in the data domain, as they can automatically understand, analyze, and process data (Hasan et al., 2023; Lucas, 2023; Zhang et al., 2024a), thereby promoting the democratization and widespread application of data science.

However, existing research still has significant shortcomings in addressing complex data science problems. Many studies are limited to simple, one-step data analysis tasks (Zhang et al., 2024c; Hu et al., 2024), which are far from the actual application scenarios of data science. While recent work (Jing et al., 2024) attempts to evaluate data science capabilities through more comprehensive tasks, it still focuses on relatively constrained scenarios that represent only portions of a complete data science pipeline. Other research relies on pre-built knowledge bases (Guo et al., 2024), raising the barrier to use and limiting the flexibility and adaptability of solutions. Moreover, current research focuses excessively on improving task completion rates and optimizing performance metrics, while neglecting the interpretability and transparency of intermediate decision-making steps in logically complex data science tasks. This neglect not only affects users' understanding of solutions but also diminishes their credibility and practicality in real-world applications.

To address these issues, we propose AutoKaggle, a universal multi-agent framework that provides data scientists with end-to-end processing solutions for tabular data, helping them efficiently complete daily data pipelines and enhance productivity. AutoKaggle has the following features:

(i) **Phase-based Workflow and Multi-agent Collaboration.** AutoKaggle employs a phase-based workflow and multi-agent collaboration system. It divides the data science competition process into six key phases: background understanding, preliminary exploratory data analysis, data cleaning (DC), in-depth exploratory data analysis, feature engineering (FE), and model-building, -validation, and -prediction (MBVP). To execute these phases, five specialized agents (Reader, Planner,

¹All code and data are available: <https://anonymous.4open.science/r/AutoKaggle-B8D2>.

054 Developer, Reviewer, and Summarizer) work collaboratively to execute these phases, from
 055 problem analysis to report generation.
 056

057 **(ii) Iterative Debugging and Unit Testing.** AutoKaggle ensures code quality through iterative de-
 058 bugging and unit testing. The Developer employs three main tools (code execution, debugging,
 059 and unit testing) to verify both syntactic correctness and logical consistency.

060 **(iii) Machine Learning Tools Library.** AutoKaggle integrates a comprehensive machine learning-
 061 tools library covering data cleaning, feature engineering, and model-building, -validation, and
 062 -prediction. The library includes expert-written code snippets and custom tools, enhancing code
 063 generation efficiency and quality. By combining predefined tools with self-generated code, AutoK-
 064 aggle handles complex tasks while reducing reliance on LLMs for domain-specific knowledge.

065 **(iv) Comprehensive Reporting.** AutoKaggle generates detailed reports after each phase and at the
 066 competition’s conclusion, showcasing its decision-making process, key findings, actions, and rea-
 067 soning. This feature makes the data processing workflows transparent, increasing user trust in Au-
 068 toKaggle.

069 AutoKaggle provides a universal and comprehensive solution for a wide variety of data science tasks.
 070 By simply providing a task overview, it can automatically complete the entire process from develop-
 071 ment to testing, making it exceptionally easy to use. AutoKaggle is highly adaptable, allowing users
 072 to customize it according to their specific needs. Moreover, it offers clear interpretability throughout
 073 the automated data science process, enhancing users’ understanding and trust in the system.

074 We chose competitions from the Kaggle platform to evaluate our framework. Kaggle data science
 075 competitions simulate the real challenges faced by data scientists, covering the complete process
 076 from data cleaning to model deployment. These competitions require participants to execute a series
 077 of complex and interdependent tasks. These include: data cleaning and preprocessing, exploratory
 078 data analysis, feature engineering, and modeling. Each step demands professional knowledge and
 079 meticulous planning, often necessitating multiple iterations. This complexity makes Kaggle an ideal
 080 platform for assessing the effectiveness of data science automation tools. In the 8 Kaggle data
 081 science competitions we evaluated, AutoKaggle achieved 0.85 in valid submission rate and 0.82 in
 082 comprehensive score. We summarize our contributions as follows:

- We propose AutoKaggle, a novel multi-agent framework for Kaggle data science competitions, achieving high task completion rates and competitive performance above the average human level in our evaluations.
- We introduce a phase-based workflow integrated with multi-agent collaboration, incorporating iterative debugging and unit testing, which systematically addresses the complexities of data science tasks and ensures robust, correct code generation.
- We develop a machine learning tools library and integrate it into our framework, enhancing code generation efficiency and quality for complex data science tasks.
- We implement a comprehensive reporting system that provides detailed insights into the decision-making process at each phase, making AutoKaggle both a solution provider and an educational tool for data science competitions, thereby contributing to the democratization of data science skills.

098 2 AUTOKAGGLE

100 2.1 OVERALL FRAMEWORK

102 In this section, we introduce AutoKaggle, a fully automated, robust, and user-friendly framework
 103 designed to produce directly submittable prediction results using only the original Kaggle data.
 104 Given the diversity of data science problems, the range of potential solutions, and the need for
 105 precise reasoning and real-time understanding of data changes, effectively handling complex data
 106 science tasks on Kaggle is challenging. Our technical design addresses two primary issues: *(i)* how
 107 to decompose and systematically manage complex data science tasks; and *(ii)* how to efficiently
 solve these tasks using LLMs and multi-agent collaboration.

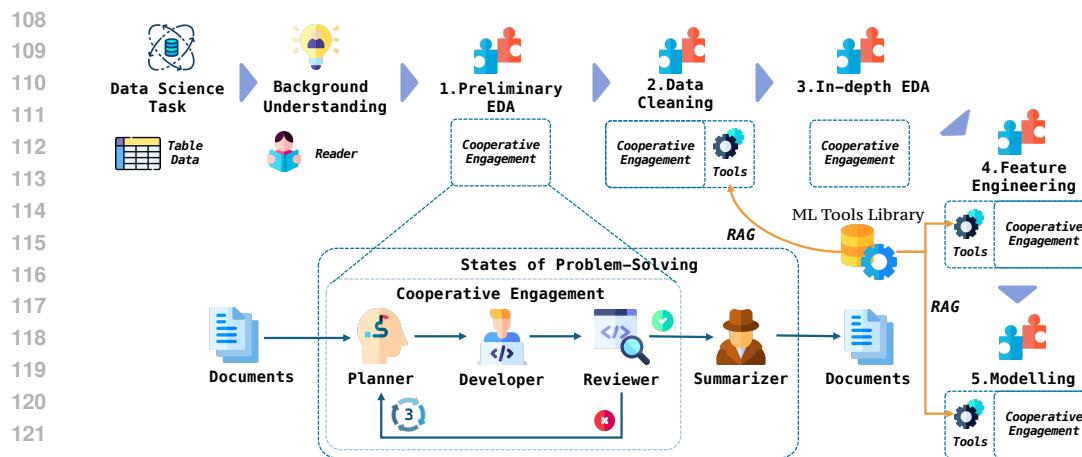


Figure 1: Overview of AutoKaggle. AutoKaggle integrates a phase-based workflow with specialized agents (Reader, Planner, Developer, Reviewer, and Summarizer), iterative debugging and unit testing, a comprehensive machine learning tools library, and detailed reporting.

The core concept of AutoKaggle is phase-based multi-agent reasoning. This method leverages LLMs to reason and solve tasks within a structured workflow, addressing different facets of the data science process through the collaboration of multiple agents. AutoKaggle comprises two main components: a phase-based workflow and a multi-agent system, which complement each other, as shown in Figure 1.

Phase-based Workflow. The data science process is divided into six key phases: understanding the background, preliminary exploratory data analysis, data cleaning, in-depth exploratory data analysis, feature engineering, and model-building, -validation, and -prediction. Data cleaning, feature engineering, and model-building, -validation, and -prediction are fundamental processes required for any data science competition. We designed two additional data analysis phases to provide essential information and insights for data cleaning and feature engineering, respectively. Given that our initial input is only an overview of a Kaggle data science competition and the raw dataset, we added a background understanding phase to analyze various aspects of the competition background, objectives, file composition, and data overview from the raw input. This structured approach ensures that all aspects of the problem are systematically and comprehensively addressed, with different phases decoupled from each other. It allows thorough unit testing at each phase to ensure correctness and prevent errors from propagating to subsequent phases.

Multi-agent System. The system consists of five specialized agents: Reader, Planner, Developer, Reviewer, and Summarizer. Each agent is designed to perform specific tasks within the workflow. They collaborate to analyze the problem, develop strategies, implement solutions, evaluate results, and generate comprehensive reports. Detailed setup and interaction processes of agents are described in Appendix D.1.

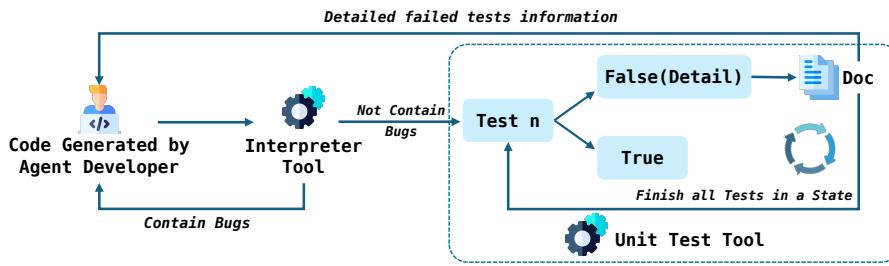


Figure 2: Iterative debugging and testing.

We summarize the pseudo-code of AutoKaggle in Algorithm 1. Let \mathcal{C} represent the competition, \mathcal{D} the dataset, and $\Phi = \{\phi_1, \phi_2, \dots, \phi_6\}$ the set of all phases in the competition workflow. For each phase ϕ_i , a specific set of agents \mathcal{A}_{ϕ_i} is assigned to perform various tasks. The key agents include Planner, Developer, Reviewer, and Summarizer.

2.2 DEVELOPMENT BASED ON ITERATIVE DEBUGGING AND TESTING

In AutoKaggle, the Developer adopts a development approach based on iterative error correction and testing. It ensures the robustness and correctness of generated code through iterative execution, debugging, and testing.

Figure 2 shows the overall process of iterative debugging and testing. Specifically, the Developer first generates code based on the current state s_t , the plan P_{ϕ_i} created by the Planner, and the historical context \mathcal{H} : $C_{\phi_i} = \text{GenerateCode}(s_t, P_{\phi_i}, \mathcal{H})$. C_{ϕ_i} is the generated code for phase ϕ_i , and $\text{GenerateCode}(\cdot)$ represents the code generation function executed by the Developer. The historical context \mathcal{H} includes previous phases' code, outputs, and other relevant information from other agents' activities.

After the initial code generation, it enters an iterative debugging and testing process. This process can be described by Algorithm 2.

Developer utilize three primary tools: code execution, code debugging, and unit testing.

(i) Code Execution. The Code Execution tool runs the generated code and captures any runtime errors. When an error is detected, the system restores a file to record the error messages.

(ii) Code Debugging. The Code Debugging tool analyzes error messages and attempts to fix the code. It utilizes error messages along with the current code and historical context to generate fixes: $C'_{\phi_i} = \text{DebugCode}(C_{\phi_i}, E_{\phi_i}, \mathcal{H})$. C'_{ϕ_i} is the debugged version of the code.

Following previous work (Tyen et al., 2024), we designed the debugging process into three main steps: error localization, error correction, and merging of correct and corrected code segments. We set a maximum of 5 attempts for the Developer to self-correct errors. Additionally, we've introduced an assistance mechanism. We record all error messages encountered during the debugging process. When the number of correction attempts reaches 3, the Developer evaluates the feasibility of continuing based on historical information. If past error messages are similar, it suggests that the Developer might lack the ability to resolve this particular error, and continuing might lead to a loop. In such cases, we allow the Developer to exit the correction process and regenerate the code from scratch.

(iii) Unit Testing. Unit testing runs predefined tests to ensure code meets requirements. For each phase ϕ_i , a set of unit tests T_{ϕ_i} is defined: $T_{\phi_i} = \{t_1, t_2, \dots, t_k\}$. The unit testing process can be represented as: $R_{\phi_i} = \text{ExecuteUnitTests}(C_{\phi_i}, T_{\phi_i})$. R_{ϕ_i} is the set of test results, with each result $r_j \in \{0, 1\}$ indicating whether the corresponding test passed (1) or failed (0).

In complex and accuracy-demanding tasks like Kaggle data science competitions, merely ensuring that the code runs without errors is not enough. These competitions often involve intricate data processing and sophisticated algorithms, where hidden logical errors can significantly affect the final results. Therefore, it is necessary to design meticulous unit tests that not only verify the correctness of the code but also ensure it meets the expected logical and performance standards. Otherwise, hidden errors may accumulate through successive phases, making the completion of each subsequent phase increasingly difficult. For example, unnoticed logical defects during the data cleaning phase may lead to poor feature extraction, thereby affecting the model building in subsequent phases.

To mitigate these risks, unit tests for each phase must be carefully designed to cover a wide range of scenarios, including edge cases and potential failure points. This involves not only checking the correctness of the output but also ensuring that the intermediate steps conform to the expected logic. For instance, in the data cleaning phase, unit tests should verify whether missing values are handled correctly, outliers are appropriately managed, and data transformations are accurately applied.

By implementing comprehensive unit tests, we can catch and correct errors early in the development process, preventing them from propagating to later phases. This systematic testing approach ensures

216 that the code at each phase is not only error-free but also functionally correct and aligned with the
 217 overall project goals.
 218

219 In conclusion, the iterative debugging and testing method employed by Developer ensures the
 220 generation of robust, error-free, and effective code for each phase of the competition. By employing
 221 advanced error handling, iterative debugging, and comprehensive unit testing, the system can adapt
 222 to various challenges and consistently produce high-quality code outputs.

223 2.3 MACHINE LEARNING TOOLS LIBRARY 224

225 Generating machine learning code from scratch using LLMs can be challenging due to the intri-
 226 cacies of various tasks. These models need to encompass specialized knowledge across a range
 227 of processes, from data processing and feature engineering to model-building, -validation, and -
 228 prediction. In many cases, leveraging expert-crafted machine learning tools is more efficient than
 229 relying solely on LLM-generated code. This is because LLMs often lack domain-specific expertise,
 230 potentially leading to suboptimal or inaccurate code. Furthermore, when tasked with complex oper-
 231 ations, the generated code may suffer from syntactical or logical errors, increasing the likelihood of
 232 failures.

233 Our machine learning library is categorized into three core toolsets: data cleaning, feature engineer-
 234 ing, and model-building, -validation, and -prediction, each serving a specific role in the workflow.
 235 The data cleaning toolkit comprises seven tools, including FillMissingValues, RemoveColumns
 236 WithMissingData, DetectAndHandleOutliersZscore, DetectAndHandleOutliersIqr, RemoveDuplic-
 237 ates, ConvertDataTypes and FormatDatetime, all designed to ensure clean, consistent, and reli-
 238 able data preparation. The feature engineering module encompasses eleven tools aimed at enhanc-
 239 ing model performance, such as OneHotEncode, FrequencyEncode, CorrelationFeatureSelection,
 240 and ScaleFeatures, employing various techniques like correlation analysis and feature scaling to
 241 optimize data representation. The model-building, -validation, and -prediction category provides
 242 TrainAndValidationAndSelectTheBestModel to support the full model development lifecycle, in-
 243 cluding model selection, training, evaluation, prediction, ensemble integration, and hyperparam-
 244 eter optimization, facilitating robust model deployment and effective performance. Each tool comes
 245 with comprehensive explanations, input/output specifications, anomaly detection, and error handling
 246 guidance.

246 This comprehensive library is crucial for efficient multi-agent collaboration in tackling complex
 247 Kaggle competitions. Each tool provides standardized, reliable functionality, enabling AutoKaggle
 248 to seamlessly share and process data, enhance feature quality, and optimize model performance,
 249 ultimately improving overall workflow efficiency and ensuring coordinated, high-quality solutions
 250 in a competitive environment. Moreover, our machine learning library reduces the burden on Au-
 251 toKaggle in detailed programming tasks, enabling them to focus more on higher-level task planning
 252 and code design. This shift of focus allows AutoKaggle to navigate complex tasks more effectively,
 253 ultimately improving their overall performance. More details of our machine learning tools can be
 254 found in Appendix D.3.

255 3 EXPERIMENTS

256 3.1 EXPERIMENTAL SETUP

259 **Task Selection.** We select eight Kaggle competitions that predominantly use tabular datasets, fo-
 260 cusing on classification and regression tasks. These competitions are categorized into two types:
classic Kaggle and *Recent Kaggle*. Classic Kaggle competitions are those that begin before October
 261 2023 with at least 500 participants, whereas Recent Kaggle competitions begin in 2024 or later. As
 262 our analysis relies on GPT-4o, which is trained on data available until October 2023, it possibly
 263 includes information about Classic Kaggle competitions, thereby posing a risk of data leakage. To
 264 evaluate the generalization capabilities of AutoKaggle, we therefore focus on competitions initiated
 265 after 2024. Additionally, we classify these competitions into three difficulty levels: easy, medium,
 266 and hard. For each dataset, we access the corresponding competition’s homepage on Kaggle, ex-
 267 tract content from the overview and data description sections, and compile this information into a
 268 file named overview.txt. This file, along with the original competition data files, forms the primary
 269 input for AutoKaggle. More details of our datasets can be found in Appendix C.

Notably, we do not incorporate the nine tabular datasets from Mle-Bench (Hong et al., 2024) due to their substantial size, which would significantly increase computational runtime. Resource limitations prevent us from adhering to Mle-Bench’s experimental setup, which specifies a 24-hour participation window per agent and a 9-hour code execution timeout.

Table 1: Made submission, valid submission and comprehensive score on 8 Kaggle tasks. Each experiment is repeated with 5 trials. The best performances on individual tasks are underlined, and the best performances across all tasks are bolded.

Metric	Setting / Task	Classic				Recent				Avg.
		Task 1	Task 2	Task 3	Task 4	Task 5	Task 6	Task 7	Task 8	
Made Submission	AutoKaggle gpt-4o	<u>1</u>	<u>0.80</u>	<u>0.80</u>	<u>1</u>	0.80	<u>0.80</u>	<u>0.80</u>	<u>0.80</u>	0.85
	AutoKaggle o1-mini	<u>1</u>	0.60	0.60	<u>1</u>	0.60	<u>0.80</u>	0.60	0.60	0.73
	AIDE gpt-4o	<u>1</u>	0.40	0.20	0.60	<u>1</u>	<u>0.80</u>	<u>0.80</u>	0	0.60
Valid Submission	AutoKaggle gpt-4o	<u>1</u>	<u>0.80</u>	<u>0.80</u>	<u>1</u>	<u>0.80</u>	0.60	<u>0.80</u>	<u>0.80</u>	0.83
	AutoKaggle o1-mini	<u>1</u>	0.60	0.60	<u>1</u>	0.60	0.60	0.60	0.60	0.70
	AIDE gpt-4o	<u>1</u>	0.40	0.20	0.40	<u>1</u>	0.80	<u>0.80</u>	0	0.58
Comprehensive Score	AutoKaggle gpt-4o	0.888	<u>0.786</u>	<u>0.831</u>	0.862	0.810	0.728	<u>0.848</u>	<u>0.812</u>	0.821
	AutoKaggle o1-mini	0.879	0.680	<u>0.729</u>	<u>0.863</u>	0.709	0.735	0.742	<u>0.735</u>	0.759
	AIDE gpt-4o	0.872	0.597	0.542	0.561	0.918	<u>0.793</u>	0.848	0	0.641

Evaluation metric. We evaluate the capability of the AutoKaggle from four perspectives: Made Submission, Valid Submission, Average Normalized Performance Score and Comprehensive Score. The first two metrics refer to Mle-bench and are primarily used to assess the ability to generate a submission.csv file. The last two metrics come from Data Interpreter (Chan et al., 2024), we made modifications to adapt them to the evaluation of our framework.

(i) **Made Submission (MS).** Made Submission refers to the percentage of times a submission.csv file is generated.

(ii) **Valid Submission (VS).** Valid Submission indicates the percentage of those submission.csv files that are valid—meaning they can be successfully submitted to the Kaggle website, produce results without errors, and have no issues related to data scale or category mismatches.

(iii) **Comprehensive Score (CS).** In the evaluations, performance metrics are divided into two categories: bounded metrics, which range from 0 to 1 where higher values indicate better performance, and unbounded metrics, where lower values denote superior performance. To normalize these different types of metrics, we utilize the normalized performance score (NPS), defined as follows:

$$\text{NPS} = \begin{cases} \frac{1}{1+s}, & \text{if } s \text{ is smaller the better} \\ s, & \text{otherwise.} \end{cases} \quad (1)$$

For multiple trials of a task, we calculate the Average Normalized Performance Score (ANPS) as the average of the successful attempts:

$$\text{ANPS} = \frac{1}{T_s} \sum_{t=1}^{T_s} \text{NPS}_t \quad (2)$$

Table 2: Ablation study on machine learning tools. Evaluated with completion rate and comprehensive score. Best performance are underlined.

		Task 1	Task 2	Task 3	Task 5	Avg.
VS	No Tools	0.80	0.60	0.50	0.40	0.58
	DC Tools	0.80	0.70	<u>1.00</u>	<u>1.00</u>	0.88
	DC & FE Tools	0.80	0.60	0.60	0.60	0.65
	All Tools	<u>1.00</u>	<u>0.80</u>	0.80	0.80	0.85
CS	No Tools	0.781	0.697	0.666	0.602	0.687
	DC Tools	0.781	0.721	<u>0.928</u>	<u>0.909</u>	0.835
	DC & FE Tools	0.787	0.684	0.735	0.713	0.730
	All Tools	<u>0.888</u>	<u>0.786</u>	0.831	0.810	0.829

324 where T_s represents the total number of successful attempts for a task, and NPS_t is the NPS value
 325 for the t -th attempt.

326 To comprehensively evaluate both the pass rate and the average performance, we define the Com-
 327 prehensive Score (CS) as the average of VS and ANPS:

$$329 \quad CS = 0.5 \times VS + 0.5 \times ANPS \quad (3)$$

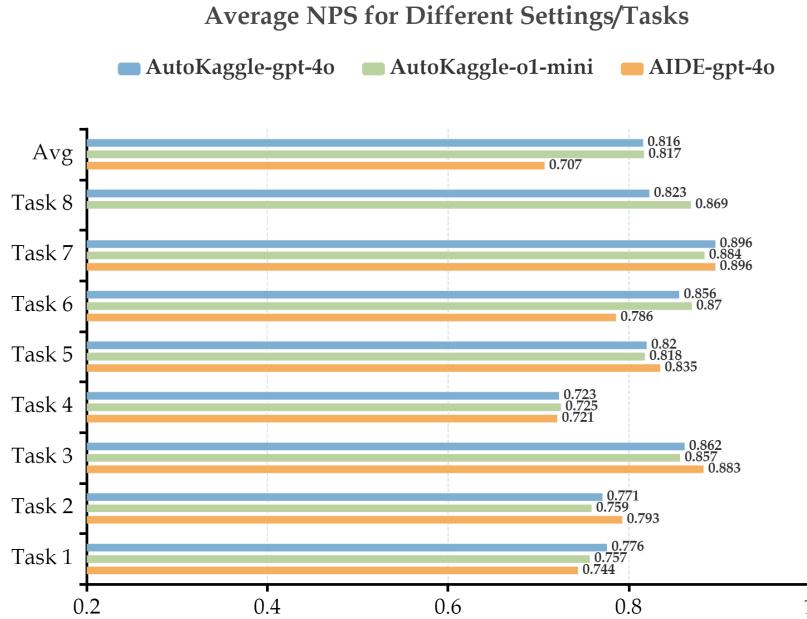
330
Experiment Details. We evaluated AutoKaggle’s performance based on both GPT-4o and o1-mini
 331 models. Notably, different models were assigned to specific agents based on their functional re-
 332 quirements. The Reader, Reviewer, and Summarizer, which perform tasks requiring minimal logical
 333 reasoning and coding capabilities, were implemented using the GPT-4o-mini model.
 334 The Planner, responsible for task decomposition and planning that demands sophisticated logi-
 335 cal reasoning, operates on either the GPT-4o or o1-mini model. Although the Developer’s tasks
 336 traditionally necessitate advanced logical reasoning and coding skills, the Planner’s effective task
 337 decomposition methodology has moderated these requirements, therefore it is based on GPT-4o
 338 model.

339 In our experiments, Each task undergoes five trials, with each phase in the workflow allowing for a
 340 maximum of three iterations. During an iteration, the Developer may debug the code up to five
 341 times. If unsuccessful, they proceed with the same phase, deriving insights and adjusting strategies
 342 based on previous attempts. Failure to resolve issues after three iterations is considered a definitive
 343 failure.

344
Baseline. We employ AIDE (Schmidt et al., 2024) as our baseline, which is the best-performing
 345 framework in Mle-bench evaluation results. We use AIDE’s default settings, only modifying
 346 agent.base.model to the GPT-4o model.

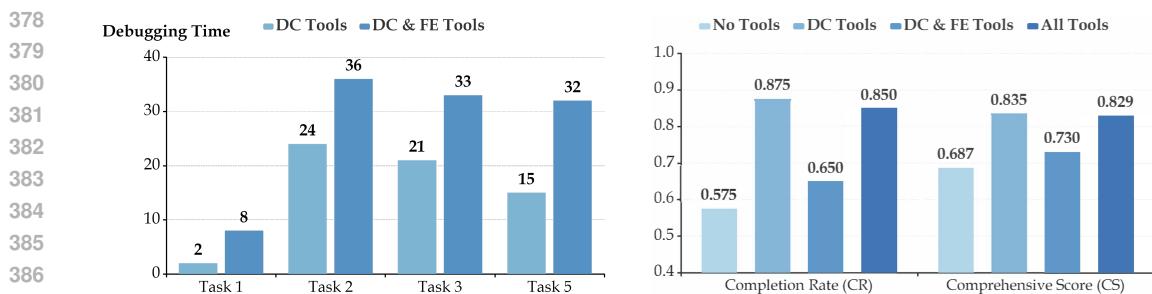
348 3.2 MAIN RESULTS

350 The comprehensive performance of AutoKaggle across 8 Kaggle data science competitions is pre-
 351 sented in Table 1. In order to facilitate understanding, we uniformly name the eight tasks as task
 352 1-8. The real task names and detailed dataset information are available in Appendix C.



375
 376 Figure 3: Average normalized performance score for different settings/tasks.

377
Made submission and Valid submission. We first evaluated the success rate of valid submis-
 378 sion.csv file generation across different experimental configurations. The AutoKaggle framework,
 379 implemented with GPT-4o, demonstrated superior performance with an average valid submission

Figure 4: **Left.** Debugging time and **Right.** Average performance in competitions.

rate of 83% across all 8 Kaggle tasks, surpassing the AIDE framework by 28%. These results underscore the robustness of our framework in executing comprehensive data science workflows. While the AIDE framework successfully processed Tasks 1-7, which involved single-variable classification or regression on tabular data, it failed to generate valid submissions for Task 8, a multi-variable classification problem. This differential performance demonstrates our framework’s versatility in handling diverse tabular data tasks.

Another interesting observation is that within the AutoKaggle framework, the GPT-4o model achieved better results than the o1-mini model, despite the latter’s purported superior reasoning capabilities. This performance difference emerged solely from varying the model used in the Planner component. We hypothesize that this counterintuitive result stems from o1-mini’s tendency toward excessive planning complexity, which proves disadvantageous in our streamlined, phase-based workflow architecture. This same consideration influenced our decision to maintain GPT-4o as the Developer’s base model, as our experiments indicated that an o1-mini-based Developer would significantly increase code verbosity, expanding 100-line solutions to approximately 500 lines through the introduction of superfluous components such as logging systems.

Comprehensive Score. Subsequently, we compared the overall performance of different settings across 8 Kaggle tasks. AutoKaggle with GPT-4o achieved the highest comprehensive score in 5 tasks and demonstrated the best overall performance. Figure 3 illustrates the comparison of different settings based on the average normalized performance score metric, where AutoKaggle with o1-mini achieved the highest overall score. This indicates that although the o1-mini-based Planner generated overly complex plans that increased development difficulty, successfully executing these plans according to specifications led to superior performance outcomes.

3.3 ABLATION STUDY

Apart from the modules involved in the ablation study, all other experimental settings are identical to those in the formal experiment.

Study on Machine Learning Tools. To evaluate the effectiveness of the machine learning tools module and the impact of tools across different phases on the results, we conduct ablation experiments. We begin without any tools and progressively add them at each phase until all machine learning tools are implemented. The results are presented in Table 2. Notably, the completion rate increases by 30% with the use of data cleaning phase tools, and by 27.5% when all tools are utilized, compared to the scenario with no tools. However, the completion rate exhibits a decline during the feature engineering phase, particularly in the house prices and academic success competitions. This decline can be attributed to the relatively large number of features involved, alongside the complexity and high encapsulation of the tools used in this phase, which necessitate the addition and removal of features, thereby complicating their usage. Furthermore, this complexity poses challenges for Developers in debugging erroneous code. As illustrated in Figure 4 (a), the frequency of debugging instances is greater when employing tools from the feature engineering phase.

Figure 4 (b) provides a clearer comparison, demonstrating that while the best normalized performance scores across four scenarios are similar, the completion rate significantly increases with the use of the tool. This suggests that although the machine learning tool library we develop does not substantially elevate the solution’s upper limit, it functions as a more stable tool that enhances AutoKaggle’s completion rate. This outcome aligns with expectations, as the machine learning tool

library is a redevelopment based on widely used libraries such as pandas and scikit-learn. It does not introduce new functionalities but instead combines and re-packages existing ones, incorporating error handling and manual testing to ensure compatibility with our framework.

Study on Unit Tests. To evaluate the effectiveness of the unit tests module, we conduct ablation experiments. The results are presented in Table 3. In the absence of unit tests, the completion rate significantly decreases, making it nearly impossible to complete the tasks. This emphasizes that for tasks like data science, which demand high levels of precision and logic, it is not enough for each phase of the code to merely execute without errors. Comprehensive unit testing is required to ensure that the code is logical and achieves the objectives of each phase.

Study on Debugging Times. We conduct ablation experiments to investigate the impact of the number of allowed debugging times on the results. The experimental setup permits five code debugging attempts within each phase, with each phase being executable up to three times. Consequently, we analyze scenarios with allowable corrections set at 0, 5, and 10. The results are shown in Figure 5. It can be observed that when AutoKaggle is required to pass without any errors, there is only one successful record on the Titanic task. Allowing five debugging attempts significantly improves the completion rate, and further increases in allowable debugging attempts lead to rises in all metrics. This demonstrates the efficacy of our code debugging module. However, the performance when the number of allowable debugging attempts is set to 10 and 15, suggesting that the agent's self-correction abilities are limited. There are complex errors that it cannot resolve independently, and further increasing the number of allowable debugging attempts does not address these errors. See more details in Section B.

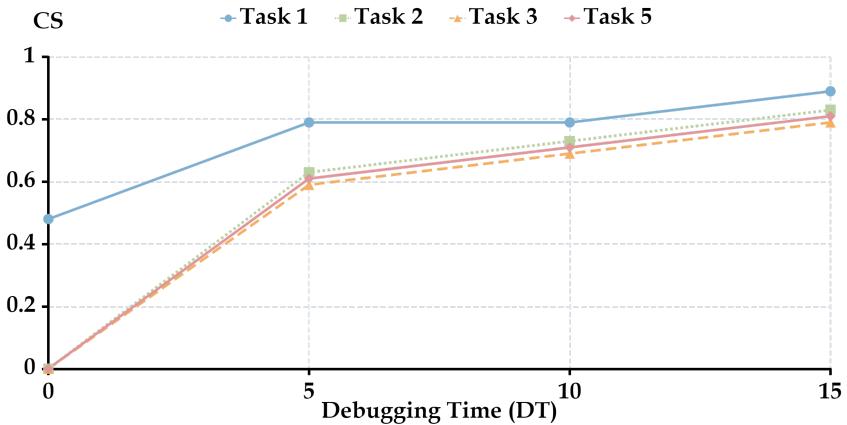


Figure 5: Comprehensive Score across different debugging times.

Study on Competition Date. To further evaluate the generalization capabilities of our AutoKaggle framework, we conducted an analysis stratified by competition dates. Tasks 1-4 corresponded to competitions potentially included in the training data of models such as GPT-4o and O1-mini, while tasks 5-8 were derived from competitions launched in the current year. This temporal stratification enabled us to assess the framework's performance on out-of-distribution tasks. For classic Kaggle tasks, AutoKaggle with GPT-4o achieved a valid submission rate of 0.90 and a comprehensive score of 0.842. On recent tasks, these metrics were 0.75 and 0.800 respectively, demonstrating only marginal performance degradation. These results indicate that our task decoupling approach and

Table 3: Ablation study on unit tests. Better performance are underlined.

		Task 1	Task 2	Task 3	Task 5	Avg.
CR	w/o Unit Tests	0.20	0	0.20	0	0.10
	w/ Unit Tests	<u>1.00</u>	<u>0.80</u>	<u>0.80</u>	<u>0.80</u>	<u>0.85</u>
CS	w/o Unit Tests	0.478	0	0.482	0	0.240
	w/ Unit Tests	<u>0.888</u>	<u>0.831</u>	<u>0.786</u>	<u>0.810</u>	<u>0.829</u>

486 predefined execution pathways enable effective handling of novel competitions, even in scenarios
 487 where the underlying model lacks prior exposure to the domain.
 488

489 4 RELATED WORK

490 A concise framework of agents consists of brain, perception, and action modules (Xi et al., 2023).
 491 The perception module processes external information, the brain plans based on that information,
 492 and the action module executes these plans (Xi et al., 2023; Zhou et al., 2023). LLMs, acting as brain
 493 modules, exhibit impressive zero-shot abilities and are applied in fields like data science and music
 494 composition (Brown et al., 2020; Hong et al., 2024; Deng et al., 2024). While the chain-of-thought
 495 method enhances reasoning (Wei et al., 2023), it still faces challenges related to hallucinations and
 496 unfaithfulness (Turpin et al., 2023), potentially due to internal representations (Yao et al., 2023).
 497 The ReAct paradigm addresses this by integrating thoughts and actions, refining outputs through
 498 interaction with external environments (Yao et al., 2023; Madaan et al., 2023; Shinn et al., 2023;
 499 Zhou et al., 2024).

500 While an individual agent can achieve basic natural language processing (NLP) tasks, real-world
 501 tasks have higher complexities. In human societies, people chunk complex tasks into simple sub-
 502 tasks that different people can easily handle. **Inspired by this division of labor principle, multi-agent**
 503 **systems enhance performance (Talebirad & Nadiri, 2023) using cooperative interactions (Xi et al.,**
 504 **2023; Li et al., 2023) to achieve shared goals.** Another interaction method is adversarial interactions
 505 (Lewis et al., 2017), where several agents compete with each other for better results, or one agent
 506 critiques and reviews the generation of another agent (Gou et al., 2024).

507 Previous studies have similarly focused on addressing problems in the data science domain, but
 508 many of these approaches suffer from limited scalability due to heavy reliance on pre-constructed
 509 expert knowledge bases (Guo et al., 2024) or the need for historical data as experience pools (Zhang
 510 et al., 2024a). Recently, the AIDE (Schmidt et al., 2024) framework demonstrated strong perfor-
 511 mance in Mle-Bench(Chan et al., 2024). However, its solutions face challenges such as insufficient
 512 process transparency and significant deviations from human logical reasoning, limiting their inter-
 513 pretability and generalizability.

514 In comparison, AutoKaggle adopt hierarchical systems (Hong et al., 2024; Zhang et al., 2024b; Chi
 515 et al., 2024) to complete tasks such as task understanding, feature engineering, and model building.
 516 In each hierarchy, separately design two agents for the code planning and code generation respec-
 517 tively (Hong et al., 2024), and use unit tests (Zhang et al., 2024b) to verify the quality of code
 518 generation. Beyond self-debugging by autonomous multi-agents, human-in-the-loop (Hong et al.,
 519 2024; Zhang et al., 2024b) mechanisms also provide oversight and corrections to LLM outputs,
 520 reducing hallucinations in each hierarchy.

521 In summary, multi-agent systems and LLM-based agents have demonstrated significant potential
 522 across domains such as NLP and data science. While individual agents excel in basic tasks, inte-
 523 grating multiple agents is crucial for tackling complex real-world challenges. By combining task-
 524 specific agents with human-in-the-loop mechanisms and unit testing, these systems improve code
 525 quality and address issues like hallucinations. Our framework, AutoKaggle, advances these efforts
 526 by integrating LLM-based reasoning with multi-agent collaboration, ensuring adaptability, correct-
 527 ness, and user control in data science competitions.

528 5 CONCLUSION

529 In this paper, we introduce AutoKaggle, a robust framework designed to leverage phase-based work-
 530 flows and multi-agent collaboration for solving complex Kaggle data science competitions. Au-
 531 toKaggle employs an iterative development process, incorporating thorough code debugging, unit
 532 testing, and a specialized machine learning tools library to address the intricate requirements of data
 533 science tasks. Our framework enhances reliability and automation in managing sophisticated data
 534 workflows, while maintaining user control through customizable processes. Extensive evaluations
 535 across various Kaggle competitions demonstrate AutoKaggle’s effectiveness, marking a significant
 536 advancement in AI-assisted data science problem-solving and expanding the capabilities of LLM-
 537 based systems in tackling real-world challenges.

540 REFERENCES
541

- 542 Ryan Holbrook Addison Howard, Ashley Chow. Spaceship titanic, 2022. URL <https://kaggle.com/competitions/spaceship-titanic>.
- 543
- 544 DataCanary Anna Montoya. House prices - advanced regression tech-
545 niques, 2016. URL <https://kaggle.com/competitions/house-prices-advanced-regression-techniques>.
- 546
- 547 Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhari-
548 wal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal,
549 Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M.
550 Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz
551 Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec
552 Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020. URL
553 <https://arxiv.org/abs/2005.14165>.
- 554
- 555 Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio
556 Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, et al. Mle-bench: Evaluating machine learn-
557 ing agents on machine learning engineering. *arXiv preprint arXiv:2410.07095*, 2024.
- 558
- 559 Yizhou Chi, Yizhang Lin, Sirui Hong, Duyi Pan, Yaying Fei, Guanghao Mei, Bangbang Liu, Tianqi
560 Pang, Jacky Kwok, Ceyao Zhang, Bang Liu, and Chenglin Wu. Sela: Tree-search enhanced
561 llm agents for automated machine learning, 2024. URL <https://arxiv.org/abs/2410.17238>.
- 562
- 563 Will Cukierski. Titanic - machine learning from disaster, 2012. URL <https://kaggle.com/competitions/titanic>.
- 564
- 565 Qixin Deng, Qikai Yang, Ruibin Yuan, Yipeng Huang, Yi Wang, Xubo Liu, Zeyue Tian, Jiahao
566 Pan, Ge Zhang, Hanfeng Lin, Yizhi Li, Yinghao Ma, Jie Fu, Chenghua Lin, Emmanouil Benetos,
567 Wenwu Wang, Guangyu Xia, Wei Xue, and Yike Guo. Composerx: Multi-agent symbolic music
568 composition with llms, 2024. URL <https://arxiv.org/abs/2404.18081>.
- 569
- 570 Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen.
571 Critic: Large language models can self-correct with tool-interactive critiquing, 2024. URL
572 <https://arxiv.org/abs/2305.11738>.
- 573
- 574 Siyuan Guo, Cheng Deng, Ying Wen, Hechang Chen, Yi Chang, and Jun Wang. Ds-agent: Auto-
575 mated data science by empowering large language models with case-based reasoning, 2024. URL
576 <https://arxiv.org/abs/2402.17453>.
- 577
- 578 Md Mahadi Hassan, Alex Knipper, and Shubhra Kanti Karmaker Santu. Chatgpt as your personal
579 data scientist, 2023. URL <https://arxiv.org/abs/2305.13657>.
- 580
- 581 Sirui Hong, Yizhang Lin, Bang Liu, Bangbang Liu, Biniao Wu, Danyang Li, Jiaqi Chen, Jiayi
582 Zhang, Jinlin Wang, Li Zhang, Lingyao Zhang, Min Yang, Mingchen Zhuge, Taicheng Guo, Tuo
583 Zhou, Wei Tao, Wenyi Wang, Xiangru Tang, Xiangtao Lu, Xiawu Zheng, Xinbing Liang, Yaying
584 Fei, Yuheng Cheng, Zongze Xu, and Chenglin Wu. Data interpreter: An llm agent for data
science, 2024. URL <https://arxiv.org/abs/2402.18679>.
- 585
- 586 Xueyu Hu, Ziyu Zhao, Shuang Wei, Ziwei Chai, Qianli Ma, Guoyin Wang, Xuwu Wang, Jing Su,
587 Jingjing Xu, Ming Zhu, Yao Cheng, Jianbo Yuan, Jiwei Li, Kun Kuang, Yang Yang, Hongxia
588 Yang, and Fei Wu. Infiagent-dabench: Evaluating agents on data analysis tasks, 2024. URL
589 <https://arxiv.org/abs/2401.05507>.
- 590
- 591 Liqiang Jing, Zhehui Huang, Xiaoyang Wang, Wenlin Yao, Wenhai Yu, Kaixin Ma, Hongming
592 Zhang, Xinya Du, and Dong Yu. Dsbench: How far are data science agents to becoming data
593 science experts?, 2024. URL <https://arxiv.org/abs/2409.07703>.
- 594
- 595 Wendy Kan. Ghouls, goblins, and ghosts... boo!, 2016. URL <https://kaggle.com/competitions/ghouls-goblins-and-ghosts-boo>.

- 594 Mike Lewis, Denis Yarats, Yann N. Dauphin, Devi Parikh, and Dhruv Batra. Deal or no deal? end-
 595 to-end learning for negotiation dialogues, 2017. URL <https://arxiv.org/abs/1706.05125>.
- 597 Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbulin, and Bernard Ghanem.
 598 Camel: Communicative agents for "mind" exploration of large language model society, 2023.
 599 URL <https://arxiv.org/abs/2303.17760>.
- 600 Killian Lucas. GitHub - KillianLucas/open-interpreter: A natural language interface for computers
 601 — github.com. <https://github.com/KillianLucas/open-interpreter>, 2023.
- 602 Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri
 603 Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad
 604 Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-
 605 refine: Iterative refinement with self-feedback, 2023. URL <https://arxiv.org/abs/2303.17651>.
- 606 OpenAI. Chatgpt: Optimizing language models for dialogue. <https://openai.com/blog/chatgpt>, 2022.
- 607 OpenAI. Gpt-4 technical report. <https://arxiv.org/abs/2303.08774>, 2023.
- 608 Dominik Schmidt, Zhengyao Jiang, and Yuxiang Wu. Introducing Weco AIDE. <https://www.weco.ai/blog/technical-report>, April 2024.
- 609 Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and
 610 Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning, 2023. URL
 611 <https://arxiv.org/abs/2303.11366>.
- 612 Yashar Talebirad and Amirhossein Nadiri. Multi-agent collaboration: Harnessing the power of
 613 intelligent llm agents, 2023. URL <https://arxiv.org/abs/2306.03314>.
- 614 Miles Turpin, Julian Michael, Ethan Perez, and Samuel R. Bowman. Language models don't always
 615 say what they think: Unfaithful explanations in chain-of-thought prompting, 2023. URL <https://arxiv.org/abs/2305.04388>.
- 616 Gladys Tyen, Hassan Mansoor, Victor Cărbune, Peter Chen, and Tony Mak. Llms cannot find
 617 reasoning errors, but can correct them given the error location, 2024. URL <https://arxiv.org/abs/2311.08516>.
- 618 Ashley Chow Walter Reade. Binary classification with a bank churn dataset, 2024a. URL <https://kaggle.com/competitions/playground-series-s4e1>.
- 619 Ashley Chow Walter Reade. Multi-class prediction of obesity risk, 2024b. URL <https://kaggle.com/competitions/playground-series-s4e2>.
- 620 Ashley Chow Walter Reade. Steel plate defect prediction, 2024c. URL <https://kaggle.com/competitions/playground-series-s4e3>.
- 621 Ashley Chow Walter Reade. Classification with an academic success dataset, 2024d. URL <https://kaggle.com/competitions/playground-series-s4e6>.
- 622 Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc
 623 Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models,
 624 2023. URL <https://arxiv.org/abs/2201.11903>.
- 625 Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Jun-
 626 zhe Wang, Senjie Jin, Enyu Zhou, Rui Zheng, Xiaoran Fan, Xiao Wang, Limao Xiong, Yuhao
 627 Zhou, Weiran Wang, Changhao Jiang, Yicheng Zou, Xiangyang Liu, Zhangyue Yin, Shihan Dou,
 628 Rongxiang Weng, Wensen Cheng, Qi Zhang, Wenjuan Qin, Yongyan Zheng, Xipeng Qiu, Xuan-
 629 jing Huang, and Tao Gui. The rise and potential of large language model based agents: A survey,
 630 2023. URL <https://arxiv.org/abs/2309.07864>.

- 648 Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao.
649 React: Synergizing reasoning and acting in language models, 2023. URL <https://arxiv.org/abs/2210.03629>.
650
651 Lei Zhang, Yuge Zhang, Kan Ren, Dongsheng Li, and Yuqing Yang. Mlcopilot: Unleashing the
652 power of large language models in solving machine learning tasks, 2024a. URL <https://arxiv.org/abs/2304.14979>.
653
654 Yaolun Zhang, Yinxu Pan, Yudong Wang, and Jie Cai. Pybench: Evaluating llm agent on various
655 real-world coding tasks, 2024b. URL <https://arxiv.org/abs/2407.16732>.
656
657 Yuge Zhang, Qiyang Jiang, Xingyu Han, Nan Chen, Yuqing Yang, and Kan Ren. Benchmarking
658 data science agents, 2024c. URL <https://arxiv.org/abs/2402.17168>.
659
660 Wangchunshu Zhou, Yuchen Eleanor Jiang, Long Li, Jialong Wu, Tiannan Wang, Shi Qiu, Jintian
661 Zhang, Jing Chen, Ruiwu Wu, Shuai Wang, Shiding Zhu, Jiuyu Chen, Wentao Zhang, Xiangru
662 Tang, Ningyu Zhang, Huajun Chen, Peng Cui, and Mrinmaya Sachan. Agents: An open-source
663 framework for autonomous language agents, 2023. URL <https://arxiv.org/abs/2309.07870>.
664
665 Wangchunshu Zhou, Yixin Ou, Shengwei Ding, Long Li, Jialong Wu, Tiannan Wang, Jiamin Chen,
666 Shuai Wang, Xiaohua Xu, Ningyu Zhang, Huajun Chen, and Yuchen Eleanor Jiang. Sym-
667 bolic learning enables self-evolving agents, 2024. URL <https://arxiv.org/abs/2406.18532>.
668
669
670
671
672
673
674
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 A ALGORITHM
703
704705 **Algorithm 1:** AutoKaggle Workflow

706 **Input :** Competition \mathcal{C} , Dataset \mathcal{D}
 707 **Output:** Solution \mathcal{S} , Comprehensive report \mathcal{R}

708 1 Initialize state \mathbf{s}_0 with first phase ϕ_1 : "Understand Background";
 709 2 $t \leftarrow 0$;
 710 3 $\Phi \leftarrow \{\phi_1, \phi_2, \dots, \phi_6\}$; /* Set of all phases */
 711 4 Define \mathcal{A}_ϕ for each $\phi \in \Phi$; /* Agents for each phase */
 712 5 **do**
 713 6 $\mathbf{s}_t \leftarrow \text{GetCurrentState}()$;
 714 7 $\phi_{\text{current}} \leftarrow \text{GetCurrentPhase}(\Phi)$;
 715 8 $\mathcal{A}_t \leftarrow \mathcal{A}_{\phi_{\text{current}}}$;
 716 9 **for** $a \in \mathcal{A}_t$ **do**
 717 10 **if** a is Planner **then**
 718 11 $r_a \leftarrow a.\text{execute}(\mathbf{s}_t)$;
 719 12 $\mathbf{s}_t \leftarrow \text{UpdateState}(\mathbf{s}_t, r_a)$;
 720 13 **if** UserInteractionEnabled() **then**
 721 14 $\mathbf{s}_t \leftarrow \text{UserReview}(\mathbf{s}_t)$; /* User Review plan */
 722 15 **else if** a is Developer **then**
 723 16 $r_a \leftarrow a.\text{execute}(\mathbf{s}_t)$;
 724 17 $\mathbf{s}_t \leftarrow \text{UpdateState}(\mathbf{s}_t, r_a)$;
 725 18 **if** NoErrors(r_a) **then**
 726 19 $T \leftarrow \text{ExecuteUnitTests}(\phi_{\text{current}})$;
 727 20 **if** $\neg \text{PassTests}(T)$ **then**
 728 21 $\mathbf{s}_t \leftarrow \text{Debug}(\mathbf{s}_t)$;
 729 22 **else**
 730 23 $r_a \leftarrow a.\text{execute}(\mathbf{s}_t)$;
 731 24 $\mathbf{s}_t \leftarrow \text{UpdateState}(\mathbf{s}_t, r_a)$;
 732 25 **if** AllAgentsCompleted(\mathcal{A}_t) **and** PassTests(T) **then**
 733 26 $\phi_{\text{current}} \leftarrow \text{NextPhase}(\Phi)$;
 734 27 $t \leftarrow t + 1$;
 735 28 **while** $\exists \phi \in \Phi : \text{not completed}(\phi)$;
 736 29 $\mathcal{S} \leftarrow \text{ExtractSolution}(\mathbf{s}_t)$;
 737 30 $\mathcal{R} \leftarrow \text{GenerateReport}(\mathbf{s}_t)$;

738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755

756

Algorithm 2: Development based on Iterative Debugging and Testing

757

Input : Initial code C_{ϕ_i} , Current state s_t , Plan P_{ϕ_i} , Historical context \mathcal{H} , Maximum tries max_tries , Error threshold $threshold$

759

Output: Debugged and tested code C'_{ϕ_i} , Execution flag $execution_flag$

760

```

1 round ← 0;
2 error_flag ← false;
3 execution_flag ← true;
4 retry_flag ← false;
5 error_history ← ∅;
6 while round < max_tries do
7   if round = 0 or retry_flag then
8     C_{\phi_i} ← GenerateCode(s_t, P_{\phi_i}, \mathcal{H});
9     error_history ← ∅;
10    retry_flag ← false;
11    error_flag, E_{\phi_i} ← RunCode(C_{\phi_i});
12    if error_flag then
13      error_history ← error_history ∪ {E_{\phi_i}};
14      if |error_history| >= threshold then
15        retry_flag ← EvaluateRetry(error_history);
16        if retry_flag then
17          continue;
18      C_{\phi_i} ← DebugCode(C_{\phi_i}, E_{\phi_i}, \mathcal{H});
19    else
20      R_{\phi_i} ← ExecuteUnitTests(C_{\phi_i}, T_{\phi_i});
21      if ∃r_j ∈ R_{\phi_i} : r_j = 0 then
22        C_{\phi_i} ← DebugTestFailures(C_{\phi_i}, R_{\phi_i}, \mathcal{H});
23      else
24        execution_flag ← true;
25        break;
26    round ← round + 1;
27  if round = max_tries then
28    execution_flag ← false;
29 return C_{\phi_i}, execution_flag

```

790

791

792

B ERROR ANALYSIS

793

In each subtask phase of AutoKaggle, errors may occur, with data cleaning and feature engineering experiencing the highest error rates at 25% and 22.5%, respectively. Notably, failures during the feature engineering phase result in direct competition failures in 31.25% of cases.

794

795

In the context of the proposed AutoKaggle framework, which aims to assist data scientists in solving complex tabular data challenges through a collaborative multi-agent system, Table 4 provides an overview of the different types of errors encountered during the iterative development process. AutoKaggle’s workflow includes code execution, debugging, and comprehensive unit testing, and the listed errors are indicative of the various challenges encountered while automating these stages. The most frequently observed errors are Value Errors (49 occurrences), related to mismatched input types or ranges, and Key Errors (44 occurrences), resulting from attempts to access non-existent dictionary keys. Additionally, Type Errors (25 occurrences) and Model Errors (8 occurrences) highlight operational issues due to data type mismatches or incorrect model configurations, respectively. The table also details other errors such as Timeout, FileNotFoundError, and Index Errors, each contributing to the debugging process. Understanding these error types is crucial for improving AutoKaggle’s robustness and aligning automated workflows with human interventions, ultimately enhancing productivity in typical data science pipelines.

810 In addition, we provide a detailed debugging process for developers. Below, we illustrate this using
 811 a FileNotFoundError as an example of the debugging workflow:
 812

- 813 • **Error Localization:** The developer initially encounters issues executing a Python script
 814 involving file-saving operations with libraries like Matplotlib and Pandas. The specific er-
 815 rror, FileNotFoundError, is traced to nonexistent directories or incorrect file paths. Through
 816 an iterative analysis, the problematic sections of the code are identified, focusing on the
 817 need to properly manage directory paths and handle filenames.
- 818 • **Error Correction:** To address these issues, several modifications are suggested. First,
 819 the importance of ensuring that directories exist before performing file operations is high-
 820 lighted by incorporating os.makedirs to create any missing directories. Additionally, a file-
 821 name sanitization approach is recommended to prevent errors related to invalid characters
 822 in file paths. A custom sanitize_filename function is introduced to ensure filenames contain
 823 only valid characters, thereby avoiding issues caused by special symbols or whitespace.
- 824 • **Merging Correct and Corrected Code Segments:** The final step involves merging the
 825 corrected segments back into the original code to create a seamless and robust solution.
 826 The revised script includes improvements such as verifying directory existence, creating
 827 necessary directories, and applying filename sanitization to ensure compatibility across
 828 different operating systems. The corrected code is delivered with a focus on enhancing
 829 reliability, particularly in file-saving processes, making it resilient against common pitfalls
 830 like missing directories or invalid filenames.

831 C DETAILED DATASET DESCRIPTION

832 Here is the detailed description of our dataset. Note that we use task labels to represent the differ-
 833 ent datasets. Task 1 refers to Titanic (Cukierski, 2012), Task 2 refers to Spaceship Titanic (Addi-
 834 son Howard, 2022), Task 3 refers to House Prices (Anna Montoya, 2016), Task 4 refers to Monsters
 835 (Kan, 2016), Task 5 refers to Academic Success (Walter Reade, 2024d), Task 6 refers to Bank Churn
 836 (Walter Reade, 2024a), Task 7 refers to Obesity Risk (Walter Reade, 2024b), and Task 8 refers to
 837 Plate Defect (Walter Reade, 2024c).

838 Our framework deliberately avoids selecting competitions with excessively large datasets. The rea-
 839 son for this is that larger datasets significantly extend the experimental runtime, making it impracti-
 840 cal to dedicate a machine to a single experiment for such prolonged periods.

841 Table 4: Error Types of AutoKaggle in the Problem-Solving Stage

842 Error Type (Count)	843 Description
844 Value Error (49)	Fail to match the expected type or range of the input values
845 Key Error (44)	Attempt to access a dictionary element using a key that does not exist
846 File Error (8)	Attempt to access a file that does not exist in the specified location
847 Model Error (8)	Incorrect setup in the parameters or structure of a model, leading to opera- tional failures
848 Type Error (25)	Mismatch between expected and actual data type, leading to operational failure
849 Timeout Error (6)	Failure to complete a process within the allocated time period
850 Index Error (3)	Attempt to access an element at an index that is outside the range of a list or array
851 Assertion Error (1)	An assertion condition in the code is not met, indicating an unmet expected constraint
852 Name Error (2)	Use of an undeclared variable that is not recognized by the system
853 Attribute Error (2)	Attempt to access an attribute or method that does not exist for an object
854 Indentation Error (1)	Incorrect indentation disrupts code structure, preventing proper parsing

First, we intentionally avoided selecting competitions with datasets that were too large, as larger datasets can significantly extend the experimental runtime, making it impractical to use a single machine for extended experiments. Second, we adhered to real-world competition settings by generating submission files and submitting them manually for evaluation. Simply splitting the training data would result in a test set with a distribution very similar to the training data, which could inflate performance metrics—similar to the difference often seen between validation scores and real test scores. Third, our dataset clearly identifies the contest type, i.e., tabular data. Fourth, since datasets for large language modeling include publicly available Kaggle contests, we selected only those released after 2024. Our framework requires agents to independently interpret contest tasks, understand the data, and determine appropriate optimization strategies without relying on predefined guidance.”

Table 5: Selected Kaggle tasks. For each task, we show its number, category, difficulty level, number of teams, train size and test size in dataset.

Category	No.	Task Name	Task	Level	Teams	Train	Test
Classic	1	Titanic	Classification	Medium	13994	891	418
	2	Spaceship Titanic	Classification	Easy	1720	8693	4277
	3	House Prices	Regression	Medium	4383	1460	1459
	4	Monsters	Classification	Easy	763	371	529
Recent	5	Academic Success	Regression	Medium	2684	76.5K	51K
	6	Bank Churn	Regression	Easy	3632	165K	110K
	7	Obesity Risk	Classification	Easy	3587	20.8K	13.8K
	8	Plate Defect	Regression	Medium	2199	19.2K	12.8K

D IMPLEMENTATION DETAILS

D.1 AGENT DETAILS

D.1.1 AGENT BASE

The base agent is a father class of other agents (Reader, Planner, Developer, Reviewer, and Summarizer) in the AutoKaggle. This agent can act with various tools for tasks related to data analysis, model evaluation, and document retrieval etc.

D.1.2 READER

Reader is designed for reading documents and summarizing information. It processes overview.txt in each competition, subsequently providing a well-organized summary of the competition’s background

Prompt of Agent Reader / Task Prompt

Role: reading documents and summarizing information
Description: The Reader only appears in the Understand Background phase, it reads the overview.txt file of the Kaggle competition, the sample data of both training and testing sets and summarizes it into a clearly structured competition.info.txt in markdown format.

918
919**Prompt of Agent Reader / Task Prompt**920
921

```
# CONTEXT #
{phases_in_context}
Currently, I am at phase: Background Understand.

#####
# TASK #
{task}

#####
# RESPONSE #
Let's work this out in a step by step way.

#####
# START ANALYSIS #
If you understand, please request the overview of this data science competition, and
data preview from me.
```

922

923
924

Please conduct a comprehensive analysis of the competition, focusing on the following aspects:

1. Competition Overview: Understand the background and context of the topic.
2. Files: Analyze each provided file, detailing its purpose and how it should be used in the competition.
3. Problem Definition: Clarify the problem's definition and requirements.
4. Data Information: Gather detailed information about the data, including its structure and contents.
 - 4.1 Data type:
 - 4.1.1. ID type: features that are unique identifiers for each data point, which will NOT be used in the model training.
 - 4.1.2. Numerical type: features that are numerical values.
 - 4.1.3. Categorical type: features that are categorical values.
 - 4.1.4 Datetime type: features that are datetime values.
 - 4.2 Detailed data description
5. Target Variable: Identify the target variable that needs to be predicted or optimized, which is provided in the training set but not in the test set.
6. Evaluation Metrics: Determine the evaluation metrics that will be used to assess the submissions.
7. Submission Format: Understand the required format for the final submission.
8. Other Key Aspects: Highlight any other important aspects that could influence the approach to the competition.

Ensure that the analysis is thorough, with a strong emphasis on :

1. Understanding the purpose and usage of each file provided.
2. Figuring out the target variable and evaluation metrics.
3. Classification of the features.

925
926927
928

929

D.1.3 PLANNER

930

931
932

Planner is designed for creating task plans and roadmaps. The agent's main function is to structure and organize tasks into executable plans, primarily by leveraging available tools and previously generated reports.

933
934**Prompt of Agent Planner / Task Prompt**

935

Role: creating task plans and roadmaps

Description: In the first execution, the Planner collects the competition information, the current state, and the user's rules to generate a new plan. This generation involves several rounds of interaction with a LLM to gather task details, reorganize data into structured formats (Markdown and JSON), and finalize a plan.

972
973**Prompt of Agent Planner / Task Prompt**

974

975

```
# CONTEXT #
{phases_in_context}
Currently, I am at phase: {phase_name}.
```

976

977

```
#####
# INFORMATION #
{background_info}
```

978

979

```
{state_info}
```

980

981

```
#####
# NOTE #
## PLANNING GUIDELINES ##
```

982

1. Limit the plan to a MAXIMUM of FOUR tasks.
2. Provide clear methods and constraints for each task.
3. Focus on critical steps specific to the current phase.
4. Prioritize methods and values mentioned in USER RULES.
5. Offer detailed plans without writing actual code.
6. ONLY focus on tasks relevant to this phase, avoiding those belonging to other phases. For example, during the in-depth EDA phase:
 - you CAN perform detailed univariate analysis on KEY features.
 - you CAN NOT modify any feature or modify data.

983

```
## DATA OUTPUT PREFERENCES ##
1. Prioritize TEXT format (print) for statistical information.
```

984

```
2. Print a description before outputting statistics.
```

985

```
3. Generate images only if text description is inadequate.
```

986

```
## METHODOLOGY REQUIREMENTS ##
1. Provide highly detailed methods, especially for data cleaning.
```

987

```
2. Specify actions for each feature without omissions.
```

988

```
## RESOURCE MANAGEMENT ##
1. Consider runtime and efficiency, particularly for:
```

989

- Data visualization
- Large dataset handling
- Complex algorithms

990

```
2. Limit generated images to a MAXIMUM of 10 for EDA.
```

991

```
3. Focus on critical visualizations with valuable insights.
```

992

```
## OPTIMIZATION EXAMPLE ##
When using seaborn or matplotlib for large datasets:
```

993

- Turn off unnecessary details (e.g., annot=False in heatmaps)

994

- Prioritize efficiency in plot generation

995

```
#####
# TASK #
{task}
```

996

```
#####
# RESPONSE #
Let's work this out in a step by step way.
```

997

```
#####
# START PLANNING #
Before you begin, please request the following documents from me, which contain
```

998

```
important information that will guide your planning:
```

999

1. Report and plan from the previous phase

1000

2. Available tools in this phase

1001

3. Sample data for analysis

1002

1003

1004

1005

1006

1007

1008

1009

1010

1011

1012

1013

1014

1015

1016

1017

1018

1019

Please design plan that is clear and specific to each FEATURE for the current development phase: {phase_name}.

The developer will execute tasks based on your plan.

I will provide you with INFORMATION, RESOURCE CONSTRAINTS, and previous reports and plans.

You can use the following reasoning pattern to design the plan:

1. Break down the task into smaller steps.
2. For each step, ask yourself and answer:
 - "What is the objective of this step?"
 - "What are the essential actions to achieve the objective?"
 - "What features are involved in each action?"
 - "Which tool can be used for each action? What are the parameters of the tool?"
 - "What are the expected output of each action? What is the impact of the action on the data?"
 - "What are the constraints of this step?"

1026 D.1.4 DEVELOPER
1027

1028 Developer is responsible for implementing and debugging code based on the structured plans
1029 generated by the Planner. The Developer's key function is to translate the high-level task roadmap
1030 into executable code, resolve any arising issues, and perform unit tests to ensure the functionality of
1031 the solution.

1032 **Prompt of Agent Developer / Task Prompt**
1033

1034 **Role:** write and implement code according to plan
1035 **Description:** The Developer first reviews the task plan
1036 and the relevant competition information. It can gathers
1037 code from previous phases when necessary and uses LLMs
1038 to generate new code. The Developer also cleans up any
1039 redundant code sections, writes functions, and ensures the
1040 code runs correctly by debugging and performing unit tests.
1041 It iterates through the process until the code passes all
1042 tests.

1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079

```

1080
1081 Prompt of Agent Developer / Task Prompt
1082
1083 # CONTEXT #
1084 {phases_in_context}
1085 Currently, I am at phase: {phase_name}.
1086 ######
1087 # INFORMATION #
1088 {background_info}
1089 {state_info}
1090 ######
1091 # PLAN #
1092 {plan}
1093 ######
1094 # TASK #
1095 {task}
1096 ######
1097 # RESPONSE: BLOCK (CODE & EXPLANATION) #
1098 TASK 1:
1099 THOUGHT PROCESS:
1100 [Explain your approach and reasoning]
1101 CODE:
1102 ```python
1103 [code]
1104 ```
1105 EXPLANATION:
1106 [Brief explanation of the code and its purpose]
1107
1108 TASK 2:
1109 [Repeat the above structure for each task/subtask]
1110 ...
1111 ######
1112 # START CODING #
1113 Before you begin, please request the following information from me:
1114 1. Code from previous phases
1115 2. All features of the data
1116 3. Available tools
1117
1118 Once you have this information, provide your complete response with code and
1119 explanations for all tasks in a single message.
1120
1121
1122 D.1.5 REVIEWER
1123 Reviewer is responsible for evaluating the performance of other agents in completing tasks and
1124 providing constructive feedback.
1125
1126
1127
1128
1129
1130
1131
1132
1133

```

1134
1135**Prompt of Agent Reviewer / Task Prompt**1136
1137
1138
1139
1140
1141
1142
1143

Role: assess agent performance and offer feedback
Description: The Reviewer agent evaluates the performance of multiple agents. In each evaluation phase, it merges suggestions and scores from different agents into a unified report. It interacts with a LLM to generate detailed feedback, iterating through rounds to assess task results, merging agent responses, and producing both final scores and constructive suggestions.

1144
1145
1146**Prompt of Agent Reviewer**

1147

```
# CONTEXT #
{phases_in_context}
Each phase involves collaboration between multiple agents. You are currently
evaluating the performance of agents in Phase: {phase_name}.

#####
# TASK #
Your task is to assess the performance of several agents in completing Phase: {
    phase_name}.
I will provide descriptions of each agent, the tasks they performed, and the outcomes
of those tasks.
Please assign a score from 1 to 5 for each agent, with 1 indicating very poor
performance and 5 indicating excellent performance.
Additionally, provide specific suggestions for improving each agent's performance, if
applicable.
If an agent's performance is satisfactory, no suggestions are necessary.

#####
# RESPONSE: JSON FORMAT #
Let's work this out in a step by step way.

#####
# START EVALUATION #
If you are ready, please request from me the role, description, input, task and
execution result of the agent to be evaluated.
```

1166
11671168
11691170
1171**D.1.6 SUMMARIZER**1172
1173
1174

Summarizer is responsible for generating summaries, designing questions, and reorganizing both questions and answers to produce structured reports based on the competition phases.

1175
1176**Prompt of Agent Summarizer / Task Prompt**1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187

Role: assess agent performance and offer feedback
Description: The agent Summarizer works through various phases, each focusing on a specific task like choosing relevant images, designing key questions, answering phase-related questions, and organizing the responses into a structured report. Each phase involves interaction with provided inputs such as competition information, the planner's plan, and the reviewer's evaluation to synthesize the most relevant insights.

1188
1189 **Prompt of Agent Summarizer**
1190
1191 # TASK #
1192 Please reorganize the answers that you have given in the previous step, and synthesize
1193 them into a report.
1194 #####
1195 # RESPONSE: MARKDOWN FORMAT #
1196 ``'markdown
1197 # REPORT
1198 ## QUESTIONS AND ANSWERS
1199 ### Question 1
1200 What files did you process? Which files were generated? Answer with detailed file path
1201 .
1202 ### Answer 1
1203 [answer to question 1]
1204
1205 ### Question 2
1206 Which features were involved in this phase? What changes did they undergo? If any
1207 feature types were modified, answer which features are modified and how they are
1208 modified. If any features were deleted or created, answer which features are
1209 deleted or created and provide detailed explanations. (This is a FIXED question
1210 for each phase.)
1211 ### Answer 2
1212 [answer to question 2]
1213
1214 ### Question 3
1215 [repeat question 3]
1216 ### Answer 3
1217 [answer to question 3]
1218
1219 ### Question 4
1220 [repeat question 4]
1221 ### Answer 4
1222 [answer to question 4]
1223
1224 ### Question 5
1225 [repeat question 5]
1226 ### Answer 5
1227 [answer to question 5]
1228
1229 ### Question 6
1230 [repeat question 6]
1231 ### Answer 6
1232 [answer to question 6]
1233 `.
1234 #####
1235 # START REORGANIZE QUESTIONS #

1224
1225 **D.2 UNIT TESTS**

1226
1227 In data science competitions, code generated by agents may be executable in the Python interpreter,
1228 but this execution does not guarantee correctness. To ensure that data dependencies are properly
1229 handled, a Unit Test Tool is necessary. In our research, where the framework operates iteratively,
1230 we aim to separate tasks corresponding to different states in data science competitions. Each phase
1231 builds upon the results of the previous one, making it crucial to confirm that logic remains sound,
1232 data processing is accurate, and information transfers seamlessly from one state to the next. Our
1233 Unit Test Tool plays a key role in supporting the self-refine phase of LLM agents.

1234
1235 We developed unit tests (in the accompanying Table 6) based on issues identified during the execu-
1236 tion of weak baseline, strong baseline and our AutoKaggle. If the code fails to run in the Python
1237 interpreter, an error message is relayed to the agent Reviewer. If the code passes this initial stage,
1238 it progresses to the Unit Test Tool, where all required tests are executed in a loop. If a test fails, the
1239 reason is logged as short-term memory and passed to the next review state. The review and planning
1240 stages work in an adversarial interaction: the review phase compiles the reasons for failed unit tests,
1241 while the planner addresses these failures in subsequent iterations.

1242
 1243 Table 6: Overview of unit tests for state DC, FE, and MBVP. These unit tests handle to detect
 1244 missing values, outliers, duplicates, and other data consistency issues.
 1245

State	Unit test name	Unit test description
State DC	test_document_exist	Test if cleaned_train.csv and cleaned_test.csv data exist.
	test_no_duplicate_cleaned_train	Test if there are any duplicate rows in the cleaned_train.csv.
	test_no_duplicate_cleaned_test	Test if there are any duplicate rows in the cleaned_test.csv.
	test_readable_cleaned_train	Test if the cleaned_train.csv is readable.
	test_readable_cleaned_test	Test if the cleaned_test.csv is readable.
	test_cleaned_train_no_missing_values	Test if the cleaned_train.csv contains missing value.
	test_cleaned_test_no_missing_values	Test if the cleaned_test.csv contains missing value.
	test_cleaned_train_no_duplicated_features	Test if the cleaned_train.csv contains duplicate features.
State FE	test_cleaned_test_no_duplicated_features	Test if the cleaned_test.csv contains duplicate features.
	test_cleaned_difference_train_test_columns	Test if the cleaned_train.csv and cleaned_test.csv have the same features except for target variable.
	test_cleaned_train_no_missing_target	Test if the target variable is in cleaned_train.csv.
	test_document_exist	Test if processed_train.csv and processed_test.csv data exist.
	test_processed_train_feature_number	Test if the feature engineering phase is performed well in processed_train.csv.
	test_processed_test_feature_number	Test if the feature engineering phase is performed well in processed_test.csv.
	test_file_size	Test if processed data is larger than a threshold.
	test_processed_train_no_duplicated_features	Test if the processed_train.csv contains duplicate features.
State MBVP	test_processed_test_no_duplicated_features	Test if the processed_test.csv contains duplicate features.
	test_processed_difference_train_test_columns	Test if the processed_train.csv and processed_test.csv have the same features except for target varibale.
	test_processed_train_no_missing_target	Test if the target variable is in processed_train.csv.
	test_document_exist	Test if a submission file exists.
	test_no_duplicate_submission	Test if there are any duplicate rows in the submission file.
	test_readable_submission	test if the submission file is readable.
	test_file_num_submission	Test if the submission file and sample_submission.csv have the same number of rows.
	test_column_names_submission	Test if the submission file and sample_submission.csv have the same column names.
	test_submission_validity	1) Test if the submission file and sample_submission.csv have the same data index. 2) Test if the submission file and sample_submission.csv have the same numerical range.

1296 D.3 MACHINE LEARNING TOOLS DETAILS
12971298 Table 7: Overview of Tools for state DC, FE, and MBVP. This table presents various tools catego-
1299 rized by their functionality.
1300

1301 State	1302 Tool name	1303 Tool description
1304 State DC	1305 FillMissingValues	1306 Fills missing values or removes columns with 1307 missing values based on a threshold.
	1308 RemoveColumns WithMissingData	1309 Removes columns containing missing values from a 1310 DataFrame based on a threshold.
	1311 DetectAndHandleOutliersZscore	1312 Detects and handles outliers in specified columns using 1313 the Z-score method.
	1314 DetectAndHandleOutliersIqr	1315 Detects and handles outliers in specified columns using 1316 the Interquartile Range (IQR) method.
	1317 RemoveDuplicates	1318 Removes duplicate rows from a DataFrame.
	1319 ConvertDataTypes	1320 Converts the data type of specified columns in a 1321 DataFrame.
1322 State FE	1323 FormatDatetime	1324 Formats datetime columns to a specified format.
	1325 OneHotEncode	1326 Performs one-hot encoding on specified categorical 1327 columns.
	1328 LabelEncode	1329 Performs label encoding on specified categorical 1330 columns.
	1331 FrequencyEncode	1332 Performs frequency encoding on specified categorical 1333 columns.
	1334 TargetEncode	1335 Performs target encoding on specified categorical 1336 columns.
	1337 CorrelationFeatureSelection	1338 Performs feature selection based on correlation analy- 1339 sis.
1340 State MBVP	1341 VarianceFeatureSelection	1342 Performs feature selection based on variance analysis.
	1343 ScaleFeatures	1344 Scales numerical features in the specified columns of a 1345 DataFrame.
	1346 PerformPca	1347 Performs Principal Component Analysis (PCA) on the 1348 specified columns of a DataFrame.
	1349 PerformRfe	1350 Performs Recursive Feature Elimination (RFE) on the 1351 specified columns of a DataFrame.
	1352 CreatePolynomialFeatures	1353 Creates polynomial features from specified columns of 1354 a DataFrame.
	1355 CreateFeatureCombinations	1356 Creates feature combinations from specified columns of 1357 a DataFrame.
<hr/>		
1358 TrainAndValidation 1359 AndSelectTheBestModel		
1360 Trains, evaluates, and selects the best machine learning 1361 model based on the training data and labels, returning 1362 the best performing model along with the performance 1363 scores of each model and their best hyperparameters.		

1350
 1351 **Examples of Tool Schema.** In this paper, we provide two schema formats for each machine learning
 1352 tool: JSON and Markdown. Here, we take the FillMissingValues tool as an example and provide
 1353 schemas in both formats.
 1354

1355 Markdown-formatted tool schema for FillMissingValues

1356 **Description:** Fill missing values in specified columns of a DataFrame. This tool can handle
 1357 both numerical and categorical features by using different filling methods.

1358 **Applicable Situations:** Handle missing values in various types of features.

1359 **Parameters:**

1360 • **data:**

1361 – **Type:** pd.DataFrame

1362 – **Description:** A pandas DataFrame object representing the dataset.

1363 • **columns:**

1364 – **Type:** string | array

1365 – **Description:** The name(s) of the column(s) where missing values should be
 1366 filled.

1367 • **method:**

1368 – **Type:** string

1369 – **Description:** The method to use for filling missing values.

1370 – **Enum:** auto | mean | median | mode | constant

1371 – **Default:** auto

1372 • **fill_value:**

1373 – **Type:** number | string | null

1374 – **Description:** The value to use when method is constant.

1375 – **Default:** None

1376 **Required:** data, columns

1377 **Result:** Successfully fill missing values in the specified column(s) of data.

1378 **Notes:**

- 1379 • The `auto` method uses mean for numeric columns and mode for non-numeric
 1380 columns.
- 1381 • Using `mean` or `median` on non-numeric columns will raise an error.
- 1382 • The `mode` method uses the most frequent value, which may not always be appro-
 1383 priate.
- 1384 • Filling missing values can introduce bias, especially if the data is not missing com-
 1385 pletely at random.
- 1386 • Consider the impact of filling missing values on your analysis and model perfor-
 1387 mance.

1388
 1389
 1390
 1391
 1392
 1393
 1394
 1395
 1396
 1397
 1398
 1399
 1400
 1401
 1402
 1403

1404
 1405 JSON-formatted tool schema for FillMissingValues
 1406
 1407 {
 1408 "name": "fill_missing_values",
 1409 "description": "Fill missing values in specified columns
 of a DataFrame. This tool can handle both numerical and
 categorical features by using different filling methods
 .",
 1410 "applicable_situations": "handle missing values in
 various types of features",
 1411 "parameters": {
 1412 "data": {
 1413 "type": "pd.DataFrame",
 1414 "description": "A pandas DataFrame object
 representing the dataset."
 1415 },
 1416 "columns": {
 1417 "type": ["string", "array"],
 1418 "items": {
 1419 "type": "string"
 1420 },
 1421 "description": "The name(s) of the column(s)
 where missing values should be filled."
 1422 },
 1423 "method": {
 1424 "type": "string",
 1425 "description": "The method to use for filling
 missing values.",
 1426 "enum": ["auto", "mean", "median", "mode", "constant"],
 1427 "default": "auto"
 1428 },
 1429 "fill_value": {
 1430 "type": ["number", "string", "null"],
 1431 "description": "The value to use when method is 'constant'.",
 1432 "default": null
 1433 },
 1434 "required": ["data", "columns"],
 1435 "result": "Successfully fill missing values in the
 specified column(s) of data",
 1436 "additionalProperties": false,
 1437 "notes": [
 1438 "The 'auto' method uses mean for numeric columns and
 mode for non-numeric columns.",
 1439 "Using 'mean' or 'median' on non-numeric columns will
 raise an error.",
 1440 "The 'mode' method uses the most frequent value,
 which may not always be appropriate.",
 1441 "Replacing missing values can introduce bias,
 especially if the data is not missing completely at
 random.",
 1442 "Consider the impact of filling missing values on
 your analysis and model performance."
 1443]
 1444 }
 1445 }

1458
 1459 **Tool use.** During execution, we extract the machine learning tools specified in the plan generated
 1460 by Planner and use them as queries to search the entire documentation of machine learning tools.
 1461 Since the plan includes multiple tools, we retrieve several tools based on their similarity to the
 1462 queries. The Developer then uses the retrieved tools to carry out the task.
 1463

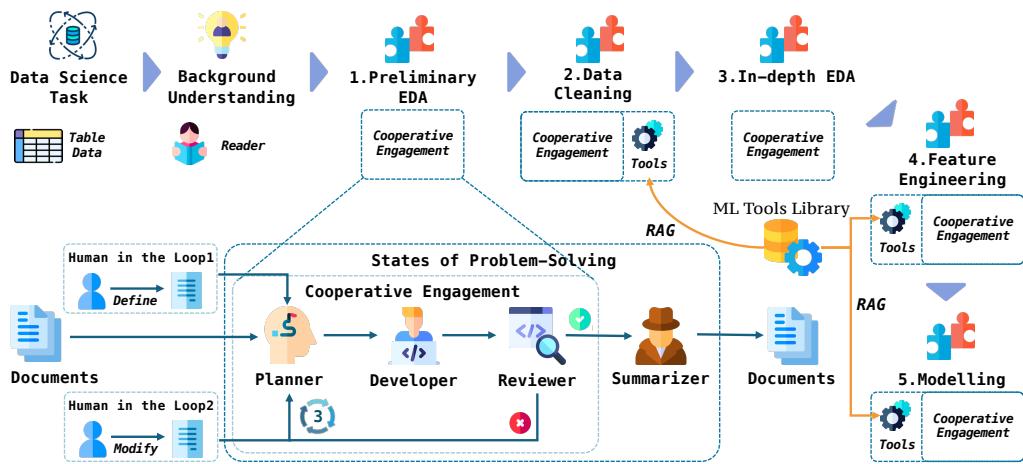
1463 D.4 TOOL UTILIZATION

1465 In the multi-agent framework designed for autonomous data science tasks, tools serve not only as
 1466 automation resources but also as integral components of the workflow. The framework enables
 1467 agents to dynamically access and execute tools as they transition through various problem-solving
 1468 states, ensuring adaptability and efficiency.

1469 The tool utilization process in this framework is structured around a systematic approach. Tool in-
 1470 formation is first stored in the system’s Memory, which is implemented as a vector database. This
 1471 Memory holds detailed explanations regarding each tool’s functionality, usage, and context. A con-
 1472 figuration file is used to map specific tools to the states in which they are required, allowing agents
 1473 to reference and identify the appropriate tools at each stage of the problem-solving process. To de-
 1474 termine which tools are required in each state, the table 7 provides an overview of tools categorized
 1475 by their functionality. As an agent moves into a particular state, it consults the configuration file to
 1476 determine the relevant tools. From the figure 1 shown, the agent subsequently queries the Memory
 1477 to retrieve detailed explanations for the tool’s use, and finally, executes the tool with precision based
 1478 on the retrieved information.

1479 This dynamic interaction between the Memory, configuration file, and agents facilitates seamless
 1480 tool integration, empowering agents to operate autonomously while maintaining flexibility and en-
 1481 suring accurate tool application throughout the autonomous process.

1482 D.5 USER INTERACTION



1501 Figure 6: Two human-in-the-loop process.
 1502

1503 We have additionally designed two Human-in-the-loop modules for the model. Note that we as-
 1504 sessed only the performance of autonomous multi-agents in all of the evaluations we wrote previ-
 1505 ously, ensuring no human intervention to maintain the fairness and objectivity of our assessment.
 1506 Figure 6 illustrates these two Human-in-the-Loop methods. Before the Planner formulates a
 1507 plan, human can interact with the command line. The input consists of meticulously manually
 1508 crafted rules, each one carefully cataloged in a handbook. Memory module subsequently retrieved
 1509 these predefined rules, integrating this human-driven knowledge in prompt engineering to guide the
 1510 Planner’s next steps. After generating the plan, humans can review and refine the Planner’s
 1511 output. They inspect areas where the logical flow seems inconsistent, focusing particularly on points
 where the output diverges from reality to address hallucination issues.

1512
1513

E CASE STUDY: TITANIC

1514
1515

E.1 BACKGROUND UNDERSTANDING

1516
1517
1518
1519

In this step, the system employs a LLM (GPT-4o) to extract and summarize the key information from the Titanic Kaggle competition. Upon completion of this process, a markdown file is automatically generated containing essential competition details, which include the competition overview, dataset information, and evaluation metrics. Below is an excerpt of the generated markdown file:

1520
1521

Background Understanding

1522

```
# Competition Information
## 1. Competition Overview
The Titanic competition challenges participants to build a predictive model to
determine which passengers survived the shipwreck of the RMS Titanic on April 15,
1912, using passenger data such as name, age, gender, and socio-economic class.

## 2. Files
- **train.csv**: Contains data for 891 passengers, including whether they survived (
    ground truth).
- **test.csv**: Includes data for 418 passengers, but without survival outcomes-these
need to be predicted.
- **sample_submission.csv**: A template showing the required format for submissions.

## 3. Problem Definition
The objective is to predict the survival of passengers based on various features in
the dataset. Participants build a model using the training set to learn from the
data and apply this model to make predictions on the test set.

.....
```

1523

```
## 6. Evaluation Metrics
The evaluation metric is **accuracy**: the percentage of correct predictions made by
the model.
```

1524

```
## 7. Submission Format
Submissions must be in CSV format with exactly 418 entries plus a header row,
containing:
- 'PassengerId': Sorted in any order.
- 'Survived': Binary predictions (1 for survived, 0 for deceased).
```

1525

```
## 8. Other Key Aspects
```

1526

- The competition runs indefinitely with a rolling leaderboard.
- Kaggle provides a community forum for discussion and collaboration.
- It's beneficial to explore shared notebooks for additional insights and approaches.

1527

1528

1529

1530

1531

1532

1533

1534

1535

1536

1537

1538

1539

1540

1541

1542

1543

1544

1545

1546

1547

E.2 PRELIMINARY EXPLORATORY DATA ANALYSIS

1548

In this state, an autonomous exploratory analysis is conducted to understand the Titanic dataset. The LLM will plan a process: 1) the dataset is loaded and inspected to reveal its structure, including data types, basic statistics, and missing values. A univariate analysis follows, where the distributions of key numerical features like age and fare are explored, as well as the frequencies of categorical features such as passenger class and gender. 2) A bivariate analysis is performed to examine the relationship between each feature and the target variable, survival. This step involves visualizing the survival rates across different categories and examining how numerical features vary between survivors and non-survivors. 3) A correlation analysis investigates relationships between numerical features, visualized through a correlation matrix and additional pairwise comparisons. This analysis provides a deeper understanding of feature interactions, highlighting those that may have the most significant impact on survival. Below is an excerpt of the generated file:

1549

1550

1551

1552

1553

1554

1555

1556

1557

1558

1559

1560

1561

1562

1563

1564

1565

1566

Preliminary Exploratory Data Analysis Plan

1567

1568

1569

1570

1571

1572

Great, thank you for the information. Let's move forward with a detailed plan for the Preliminary Exploratory Data Analysis phase. The main objectives of this phase are to understand the data structure, examine distributions, and identify any initial patterns or anomalies. We will break down the task into four key steps:

1573

1. Understanding Data Structure

1574

Objective:

To get a high-level overview of the data types, missing values, and basic statistics.

1575

Actions:

1. **Load the Data**:

- Use Pandas to read the 'train.csv' and 'test.csv' files.
- Tools: 'pd.read_csv()'
- Expected Output: DataFrames for train and test data.

2. **Inspect Data Types and Missing Values**:

.....

3. **Basic Statistical Summary**:

.....

1576

2. Univariate Analysis of Numerical Features

1577

Objective:

To understand the distribution of each numerical feature.

1578

Actions:

1. **Histograms for Numerical Features**:

.....

2. **Box Plots for Numerical Features**:

.....

1579

3. Analysis of Categorical Features

1580

Objective:

To examine the distribution and frequency of each categorical feature.

1581

Actions:

1. **Bar Plots for Categorical Features**:

.....

2. **Value Counts**:

.....

1582

4. Initial Insights and Summary

1583

Objective:

To summarize the initial findings and identify any immediate data quality issues or patterns.

1584

Actions:

1. **Summarize Key Findings**:

.....

2. **Identify Next Steps**:

.....

1585

Constraints:

- **Efficiency**: Ensure that visualizations and computations are efficient and reproducible.
- **Clarity**: Provide clear and concise visualizations and summaries to aid understanding.

1586

Expected Outputs:

1. DataFrames for train and test sets.
2. Summary of data types and missing values.
3. Basic statistical summaries.
4. Histograms and box plots for numerical features.
5. Bar plots and frequency distributions for categorical features.
6. A detailed summary of initial insights and recommended next steps.

1587

By following this plan, you'll gain a comprehensive preliminary understanding of your data, setting a solid foundation for subsequent phases.

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 Preliminary Exploratory Data Analysis Code
1622
1623 import os
1624 import pandas as pd
1625 import matplotlib.pyplot as plt
1626 import seaborn as sns
1627
1628 # File paths
1629 train_file_path = '/mnt/d/PythonProjects/AutoKaggleMaster/multi_agents/competition/
1630     titanic/train.csv'
1631 test_file_path = '/mnt/d/PythonProjects/AutoKaggleMaster/multi_agents/competition/
1632     titanic/test.csv'
1633
1634 # Loading datasets
1635 train_df = pd.read_csv(train_file_path)
1636 test_df = pd.read_csv(test_file_path)
1637
1638 # Output dataset shapes
1639 print(f"Train dataset shape: {train_df.shape}")
1640 print(f"Test dataset shape: {test_df.shape}")
1641
1642 # Inspect data types and missing values
1643 print("Missing values in train dataset:")
1644 print(train_df.isnull().sum())
1645 print("\nMissing values in test dataset:")
1646 print(test_df.isnull().sum())
1647
1648 # Basic statistical summary for numerical features
1649 print("Statistical summary for numerical features in train dataset:")
1650 print(train_df.describe())
1651
1652 # Plot histograms for numerical features
1653 numerical_features = ['Age', 'Fare', 'SibSp', 'Parch']
1654 for feature in numerical_features:
1655     plt.figure(figsize=(8, 4))
1656     sns.histplot(train_df[feature], bins=30, kde=True)
1657     plt.title(f'Histogram of {feature}')
1658     plt.xlabel(feature)
1659     plt.ylabel('Frequency')
1660     plt.savefig(f'/mnt/d/PythonProjects/AutoKaggleMaster/multi_agents/competition/
1661         titanic/pre_eda/images/histogram_{feature}.png')
1662     plt.close()
1663
1664 # Plot bar charts for categorical features
1665 categorical_features = ['Pclass', 'Sex', 'Embarked', 'Survived']
1666 for feature in categorical_features:
1667     plt.figure(figsize=(8, 4))
1668     sns.countplot(x=train_df[feature])
1669     plt.title(f'Bar chart of {feature}')
1670     plt.xlabel(feature)
1671     plt.ylabel('Count')
1672     plt.savefig(f'/mnt/d/PythonProjects/AutoKaggleMaster/multi_agents/competition/
1673         titanic/pre_eda/images/bar_chart_{feature}.png')
1674     plt.close()
1675
1676 # Summarize key findings
1677 key_findings = """
1678 Key Findings from Preliminary EDA:
1679 1. Missing Values:
1680     - Age has 177 missing values in the train dataset.
1681     - Cabin has significant missing values.
1682     - Embarked has 2 missing values in the train dataset.
1683
1684 2. Numerical Features:
1685     - Age distribution shows a right skew.
1686     - Fare distribution is highly skewed.
1687
1688 3. Categorical Features:
1689     - More males than females in the train dataset.
1690     - Most passengers embarked from port 'S' (Southampton).
1691     - Survival rate is around 38% in the train dataset.
1692 """
1693
1694 print(key_findings)

```

```

1674
1675 Preliminary EDA
1676
1677 Train Dataset Head:
1678      PassengerId  Survived  Pclass  ...   Fare Cabin Embarked
1679      0            1         0     3  ...  7.2500   NaN    S
1680      1            2         1     1  ...  71.2833  C85    C
1681      2            3         1     3  ...  7.9250   NaN    S
1682      3            4         1     1  ...  53.1000  C123   S
1683      4            5         0     3  ...  8.0500   NaN    S
1684
1685 [5 rows x 12 columns]
1686
1687 Test Dataset Head:
1688      PassengerId  Pclass  ... Cabin Embarked
1689      0            892     3  ...  NaN    Q
1690      1            893     3  ...  NaN    S
1691      2            894     2  ...  NaN    Q
1692      3            895     3  ...  NaN    S
1693      4            896     3  ...  NaN    S
1694
1695 [5 rows x 11 columns]
1696
1697 .....
1698
1699 Missing Values in Test Dataset:
1700      PassengerId      0
1701      Pclass          0
1702      Name            0
1703      Sex             0
1704      Age            86
1705      SibSp           0
1706      Parch           0
1707      Ticket          0
1708      Fare            1
1709      Cabin           327
1710      Embarked        0
1711      dtype: int64
1712
1713 Frequency of Pclass:
1714      Pclass
1715      3    491
1716      1    216
1717      2    184
1718      Name: count, dtype: int64
1719
1720 Frequency of Sex:
1721      Sex
1722      male    577
1723      female   314
1724      Name: count, dtype: int64
1725
1726 Frequency of Embarked:
1727      Embarked
1728      S    644
1729      C    168
1730      Q    77
1731      Name: count, dtype: int64
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
3000
3001
3002
3003
3004
3005
3006
3007
3008
3009
3010
3011
3012
3013
3014
3015
3016
3017
3018
3019
3020
3021
3022
3023
3024
3025
3026
3027
3028
3029
3030
3031
3032
3033
3034
3035
3036
3037
3038
3039
3040
3041
3042
3043
3044
3045
3046
3047
3048
3049
3050
3051
3052
3053
3054
3055
3056
3057
3058
3059
3060
3061
3062
3063
3064
3065
3066
3067
3068
3069
3070
3071
3072
3073
3074
3075
3076
3077
3078
3079
3080
3081
3082
3083
3084
3085
3086
3087
3088
3089
3090
3091
3092
3093
3094
3095
3096
3097
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
3139
3140
3141
3142
3143
3144
3145
3146
3147
3148
3149
3149
3150
3151
3152
3153
3154
3155
3156
3157
3158
3159
3159
3160
3161
3162
3163
3164
3165
3166
3167
3168
3169
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
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
3280
3281
3282
3283
3284
3285
3286
3286
3287
3288
3289
3289
3290
3291
3292
3293
3294
3295
3296
3296
3297
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
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
3380
3381
3382
3383
3384
3385
3386
3386
3387
3388
3389
3389
3390
3391
3392
3393
3394
3395
3396
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
3449
3449
3450
3451
3452
3453
3454
3455
3456
3457
3458
3459
3459
3460
3461
3462
3463
3464
3465
3466
3467
3468
3469
3469
3470
3471
3472
3473
3474
3475
3476
3477
3478
3478
3479
3480
3481
3482
3483
3484
3485
3486
3487
3488
3489
3489
3490
3491
3492
3493
3494
3495
3496
3497
3498
3498
3499
3499
3500
3501
3502
3503
3504
3505
3506
3507
3508
3509
3509
3510
3511
3512
3513
3514
3515
3516
3517
3518
3519
3519
3520
3521
3522
3523
3524
3525
3526
3527
3528
3529
3529
3530
3531
3532
3533
3534
3535
3536
3537
3538
3539
3539
3540
3541
3542
3543
3544
3545
3546
3547
3548
3549
3549
3550
3551
3552
3553
3554
3555
3556
3557
3558
3559
3559
3560
3561
3562
3563
3564
3565
3566
3567
3568
3569
3569
3570
3571
3572
3573
3574
3575
3576
3577
3578
3578
3579
3580
3581
3582
3583
3584
3585
3586
3586
3587
3588
3589
3589
3590
3591
3592
3593
3594
3595
3596
3597
3598
3598
3599
3599
3600
3601
3602
3603
3604
3605
3606
3607
3608
3609
3609
3610
3611
3612
3613
3614
3615
3616
3617
3618
3619
3619
3620
3621
3622
3623
3624
3625
3626
3627
3628
3629
3629
3630
3631
3632
3633
3634
3635
3636
3637
3638
3639
3639
3640
3641
3642
3643
3644
3645
3646
3647
3648
3649
3649
3650
3651
3652
3653
3654
3655
3656
3657
3658
3659
3659
3660
3661
3662
3663
3664
3665
3666
3667
3668
3669
3669
3670
3671
3672
3673
3674
3675
3676
3677
3678
3678
3679
3680
3681
3682
3683
3684
3685
3686
3687
3688
3689
3689
3690
3691
3692
3693
3694
3695
3696
3697
3698
3699
3699
3700
3701
3702
3703
3704
3705
3706
3707
3708
3709
3709
3710
3711
3712
3713
3714
3715
3716
3717
3718
3719
3720
3721
3722
3723
3724
3725
3726
3727
3728
3729
3729
3730
3731
3732
3733
3734
3735
3736
3737
3738
3739
3739
3740
3741
3742
3743
3744
3745
3746
3747
3748
3749
3749
3750
3751
3752
3753
3754
3755
3756
3757
3758
3759
3759
3760
3761
3762
3763
3764
3765
3766
3767
3768
3769
3769
3770
3771
3772
3773
377
```

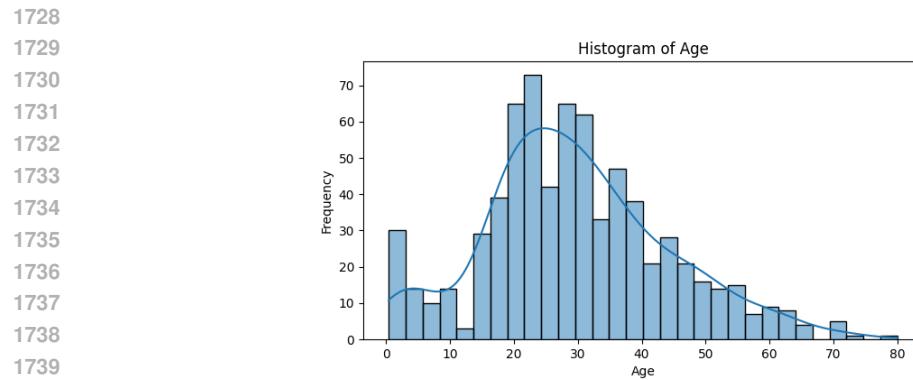


Figure 7: The histogram of age before outliers are processed

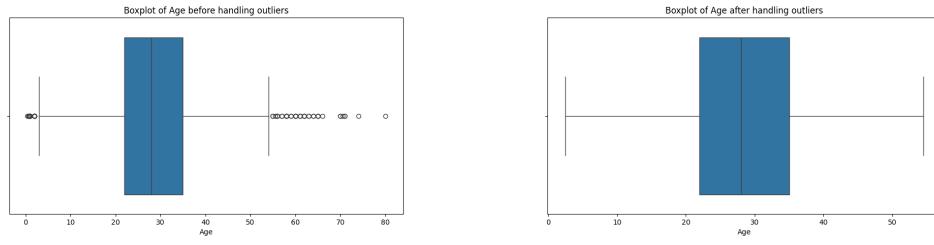


Figure 8: Comparison of age box plots before and after treatment of outliers. The image on the left is before the outliers are processed and the one on the right is after the process is done

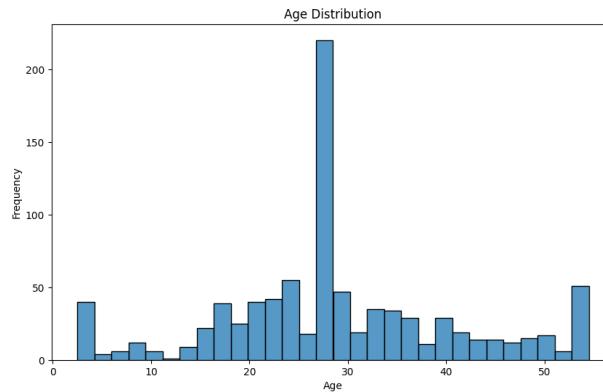


Figure 9: The histogram of age after outliers are processed

```

1782
1783 Data Cleaning Code
1784
1785 import sys
1786 import os
1787 import numpy as np
1788 import pandas as pd
1789 import matplotlib.pyplot as plt
1790 import seaborn as sns
1791
1792 sys.path.extend(['.', '..', '../..', '../../..', '/../../..', '/multi_agents',
1793                 'multi_agents/tools', 'multi_agents/prompts'])
1794 sys.path.append(os.path.dirname(os.path.abspath(__file__)))
1795 from tools.ml_tools import *
1796
1797 def generated_code_function():
1798     # Load datasets
1799     train_path = '/mnt/d/PythonProjects/AutoKaggleMaster/multi_agents/competition/
1800     titanic/train.csv'
1801     test_path = '/mnt/d/PythonProjects/AutoKaggleMaster/multi_agents/competition/
1802     titanic/test.csv'
1803     train_df = pd.read_csv(train_path)
1804     test_df = pd.read_csv(test_path)
1805
1806     # Handle missing values
1807     for df in [train_df, test_df]:
1808         df = fill_missing_values(df, columns=['Age', 'Fare'], method='median')
1809         df = fill_missing_values(df, columns=['Embarked'], method='mode')
1810         df = remove_columns_with_missing_data(df, columns=['Cabin'])
1811
1812     # Convert data types
1813     for df in [train_df, test_df]:
1814         df = convert_data_types(df, columns=['PassengerId', 'Pclass'],
1815                                 target_type='str')
1816     train_df = convert_data_types(train_df, columns=['Survived'],
1817                                 target_type='str')
1818
1819     # Plot outliers and handle using IQR method
1820     def plot_outliers(data, columns, suffix):
1821         output_dir = '/mnt/d/PythonProjects/AutoKaggleMaster/multi_agents/competition/
1822         titanic/data_cleaning/images/'
1823         os.makedirs(output_dir, exist_ok=True)
1824         for column in columns:
1825             plt.figure(figsize=(10, 5))
1826             sns.boxplot(x=data[column])
1827             plt.title(f'Boxplot of {column} {suffix}')
1828             plt.savefig(f'{output_dir}{column}_{suffix}.png')
1829             plt.close()
1830
1831         columns_with_outliers = ['Age', 'Fare']
1832         plot_outliers(train_df, columns_with_outliers, 'before_outliers')
1833
1834         for df in [train_df, test_df]:
1835             df = detect_and_handle_outliers_iqr(df, columns=columns_with_outliers,
1836                                                 factor=1.5, method='clip')
1837
1838         plot_outliers(train_df, columns_with_outliers, 'after_outliers')
1839
1840     # Save cleaned datasets
1841     train_df.to_csv('/mnt/d/PythonProjects/AutoKaggleMaster/multi_agents/competition/
1842     titanic/cleaned_train.csv', index=False)
1843     test_df.to_csv('/mnt/d/PythonProjects/AutoKaggleMaster/multi_agents/competition/
1844     titanic/cleaned_test.csv', index=False)
1845
1846 if __name__ == "__main__":
1847     generated_code_function()
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

```

```

1836
1837             Data Cleaning Result
1838
1839             Missing values in train dataset before handling:
1840             PassengerId      0
1841             ...
1842             Sex            0
1843             Age           177
1844             ...
1845             Cabin          687
1846             Embarked        2
1847             dtype: int64
1848
1849             Missing values in test dataset before handling:
1850             PassengerId      0
1851             ...
1852             Age            86
1853             ...
1854             Fare           1
1855             Cabin          327
1856             Embarked        0
1857             dtype: int64
1858
1859             Missing values in train dataset after handling:
1860             Age            0
1861             Embarked        0
1862             ...
1863             ...
1864             SibSp          0
1865             Ticket         0
1866             dtype: int64
1867
1868             Missing values in test dataset after handling:
1869             Age            0
1870             Embarked        0
1871             ...
1872             ...
1873             SibSp          0
1874             Ticket         0
1875             dtype: int64
1876
1877             Data types in train dataset after conversion:
1878             Age            float64
1879             Embarked        object
1880             Fare           float64
1881             Name           object
1882             Parch          int64
1883             PassengerId    object
1884             Pclass          object
1885             Sex            object
1886             SibSp          int64
1887             Survived        object
1888             Ticket         object
1889             dtype: object
1890             Cleaned training data saved to /mnt/d/PythonProjects/AutoKaggleMaster/multi_agents/
1891                 competition/titanic/cleaned_train.csv
1892             Cleaned test data saved to /mnt/d/PythonProjects/AutoKaggleMaster/multi_agents/
1893                 competition/titanic/cleaned_test.csv
1894
1895
1896
1897
1898
1899

```

E.4 IN-DEPTH EXPLORATORY DATA ANALYSIS

In this state, the AutoKaggle delves further into the Titanic dataset. 1) The process begins with an extended univariate analysis to explore the distribution of both numerical and categorical features.

1890 Key statistical summaries are generated for numerical features such as age and fare, while bar charts
1891 and frequency tables are created for categorical features like passenger class and gender. 2) A bi-
1892 variate analysis investigates the relationship between individual features and the survival outcome.
1893 Box plots and violin plots are used to analyze how numerical features vary between survivors and
1894 non-survivors, while count plots are generated for categorical features to visualize survival rates
1895 across different groups. 3) A correlation analysis is conducted to explore the relationships between
1896 numerical features, visualized through a correlation matrix and heatmap. This helps to identify any
1897 strong correlations between features and the target variable, survival. 4) A multivariate analysis is
1898 performed to explore interactions between key features such as passenger class, gender, and age,
1899 in relation to survival. Visualizations, such as stacked bar charts and facet grids, are used to high-
1900 light these complex interactions, providing deeper insights into the data.Below is an excerpt of the
1901 generated file:
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

In-depth Exploratory Data Analysis Plan

Great! Based on the information provided, we will proceed with a detailed plan for the **In-depth Exploratory Data Analysis (EDA)** phase. The aim is to perform a thorough analysis of each feature to uncover deeper insights that can inform feature engineering and model building.

Plan for In-Depth Exploratory Data Analysis

Task 1: Detailed Univariate Analysis
Objective: Understand the distribution and characteristics of each individual feature.

1. **Age:**
 - **Actions:**
 - Plot a histogram to observe the distribution.
 - Use a boxplot to identify any remaining outliers.
 - **Tools & Parameters:**
 - `seaborn.histplot(data=cleaned_train, x='Age', bins=30)`
 - `seaborn.boxplot(data=cleaned_train, x='Age')`
 - **Expected Output:**
 - Histogram showing the age distribution.
 - Boxplot indicating the presence of outliers.
 - **Constraints:**
 - Ensure plots are clear and readable.

2. **Fare:**
 -

3. **Categorical Features (Pclass, Sex, Embarked):**
 -

Task 2: Bivariate Analysis

Objective: Explore relationships between pairs of features, especially with the target variable 'Survived'.

1. **Survived vs. Age:**
 -

2. **Survived vs. Fare:**
 -

3. **Survived vs. Categorical Features (Pclass, Sex, Embarked):**
 -

Task 3: Multivariate Analysis

Objective: Understand interactions between multiple features and their combined effect on the target variable.

1. **Pairplot:**
 -

2. **Correlation Heatmap:**
 -

Task 4: Insights Summary

Objective: Summarize key findings from the in-depth EDA to inform the next phases.

1. **Actions:**
 -
2. **Tools & Parameters:**
 -
3. **Expected Output:**
 -
4. **Constraints:**
 -

By following this detailed plan, you will be able to gain a thorough understanding of the dataset, which will be invaluable for the subsequent phases of feature engineering and model building.

1990
1991
1992
1993
1994
1995
1996
1997

```

1998
1999
2000
2001 import pandas as pd
2002 import matplotlib.pyplot as plt
2003 import numpy as np
2004
2005 # Load the cleaned training data
2006 cleaned_train = pd.read_csv('cleaned_train.csv')
2007 # Task 1: Univariate analysis on the Age and Fare features
2008
2009 # Histogram and Boxplot for Age and Fare
2010 for feature in ['Age', 'Fare']:
2011     plt.figure(figsize=(10, 6))
2012     cleaned_train[feature].hist(bins=30)
2013     plt.title(f'{feature} Distribution')
2014     plt.xlabel(feature)
2015     plt.ylabel('Frequency')
2016     plt.savefig(f'deep_eda/images/{feature.lower()}_histogram.png')
2017     plt.close()
2018
2019     plt.figure(figsize=(10, 6))
2020     cleaned_train[feature].plot(kind='box')
2021     plt.title(f'Boxplot of {feature}')
2022     plt.xlabel(feature)
2023     plt.savefig(f'deep_eda/images/{feature.lower()}_boxplot.png')
2024     plt.close()
2025
2026 print("Univariate analysis on Age and Fare features completed.")
2027
2028 # Task 2: Univariate analysis on categorical features
2029 # Countplot for Pclass, Sex, Embarked
2030 for feature in ['Pclass', 'Sex', 'Embarked']:
2031     cleaned_train[feature].value_counts().plot(kind='bar', figsize=(10, 6), title=f'{feature} Distribution')
2032     plt.xlabel(feature)
2033     plt.ylabel('Frequency')
2034     plt.savefig(f'deep_eda/images/{feature.lower()}_countplot.png')
2035     plt.close()
2036
2037 # Task 3: Bivariate analysis of Survived vs. Age, Fare, categorical features
2038 # Violin plot for Age and Fare by Survived
2039 for feature in ['Age', 'Fare']:
2040     plt.figure(figsize=(10, 6))
2041     cleaned_train.boxplot(column=feature, by='Survived')
2042     plt.title(f'Survival by {feature}')
2043     plt.xlabel('Survived')
2044     plt.ylabel(feature)
2045     plt.savefig(f'deep_eda/images/survived_vs_{feature.lower()}.png')
2046     plt.close()
2047
2048 # Countplot for categorical features by Survived
2049 for feature in ['Pclass', 'Sex', 'Embarked']:
2050     pd.crosstab(cleaned_train[feature], cleaned_train['Survived']).plot(kind='bar',
2051         stacked=True, figsize=(10, 6))
2052     plt.title(f'Survival by {feature}')
2053     plt.xlabel(feature)
2054     plt.ylabel('Count')
2055     plt.savefig(f'deep_eda/images/survived_vs_{feature.lower()}.png')
2056     plt.close()
2057
2058 # Task 4: Multivariate analysis using a correlation heatmap
2059 # Correlation heatmap
2060 numeric_df = cleaned_train.select_dtypes(include=[np.number])
2061 plt.figure(figsize=(10, 8))
2062 plt.matshow(numeric_df.corr(), cmap='coolwarm', fignum=1)
2063 plt.title('Correlation Heatmap')
2064 plt.savefig('deep_eda/images/correlation_heatmap.png')
2065 plt.close()
2066
2067 # Task 5: Summarize key insights from the EDA
2068 summary = """
2069 ....
2070 """
2071
2072 # Save the summary to a text file
2073 with open('deep_eda/eda_summary.txt', 'w') as file:
2074     file.write(summary)

```

```

2052
2053     In-deep EDA
2054
2055     Summary statistics for Age:
2056     count    891.000000
2057     mean     29.039282
2058     std      12.072074
2059     min      2.500000
2060     25%     22.000000
2061     50%     28.000000
2062     75%     35.000000
2063     max      54.500000
2064     Name: Age, dtype: float64
2065
2066     .....
2067
2068     Survival rate by Pclass:
2069     Pclass
2070     1    0.629630
2071     2    0.472826
2072     3    0.242363
2073     Name: Survived, dtype: float64
2074     Survival rate by Sex:
2075     Sex
2076     female   0.742038
2077     male     0.188908
2078     Name: Survived, dtype: float64
2079     Survival rate by Embarked:
2080     Embarked
2081     C    0.553571
2082     Q    0.389610
2083     S    0.339009
2084     Name: Survived, dtype: float64
2085     Correlation matrix:
2086
2087     Age      SibSp      Parch      Fare      Survived
2088     Age      1.000000 -0.239601 -0.178959  0.144544 -0.060622
2089     SibSp   -0.239601  1.000000  0.414838  0.332021 -0.035322
2090     Parch   -0.178959  0.414838  1.000000  0.292616  0.081629
2091     Fare    0.144544  0.332021  0.292616  1.000000  0.317430
2092     Survived -0.060622 -0.035322  0.081629  0.317430  1.000000
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105

```

2106
2107 **Feature Engineering Plan**
2108
2109 Great! Let's design a detailed plan for the Feature Engineering phase, focusing on the
2110 current features and the available tools. We'll ensure that each task is clear,
2111 specific, and includes the necessary steps to achieve the desired outcome.
2112 #### Feature Engineering Plan
2113 ##### Task 1: Create New Features
2114 **Objective:** Enhance the dataset by creating new features that could capture
2115 additional information relevant to predicting survival.
2116 1. **Family Size:**
2117 - **Action:** Create a new feature 'FamilySize' by combining 'SibSp' and 'Parch'.
2118 - **Method:** 'FamilySize = SibSp + Parch + 1' (including the passenger themselves)
2119 - **Impact:** Captures the total number of family members traveling together, which
2120 can influence survival chances.
2121 2. **IsAlone:**
2122
2123 3. **Age Bins:**
2124
2125 4. **Fare per Person:**
2126
2127 ##### Task 2: Encode Categorical Features
2128 **Objective:** Convert categorical features into numerical format suitable for machine
2129 learning models.
2130 1. **Sex:**
2131
2132 2. **Embarked:**
2133
2134 ##### Task 3: Handle Ticket and Cabin Features
2135 **Objective:** Extract useful information from 'Ticket' and 'Cabin' features, which
2136 are currently in text format.
2137 1. **Ticket:**
2138
2139 2. **Cabin:**
2140
2141 ##### Task 4: Scale Numerical Features
2142 **Objective:** Standardize numerical features to ensure they are on a comparable scale
2143 , improving model performance.
2144 1. **Numerical Features:**
2145
2146 ### Summary of Expected Output
2147 - **New Features Added:** 'FamilySize', 'IsAlone', 'AgeBins', 'FarePerPerson'
2148 - **Encoded Features:** 'Sex' (label encoded), 'Embarked' (one-hot encoded)
2149 - **Processed Features:** 'TicketPrefix', 'CabinDeck'
2150 - **Scaled Features:** 'Age', 'Fare', 'FarePerPerson'
2151
2152 ### Constraints and Considerations
2153 - **Runtime Efficiency:** Ensure feature creation and encoding steps are optimized for
2154 performance.
2155 - **Handling Missing Values:** Address any missing values in the newly created
2156 features appropriately.
2157 - **Avoid Data Leakage:** Perform encoding and scaling separately on train and test
2158 sets to prevent data leakage.
2159
2160 By following these tasks, you will transform the cleaned data into a more informative
2161 and model-ready format ('processed_train.csv' and 'processed_test.csv'). This plan
2162 ensures that the most critical steps of feature engineering are covered,
2163 enhancing the predictive power of your model for the Titanic competition.

```

2160
2161             Feature Engineering Code
2162
2163
2164     import pandas as pd
2165
2166     # Load the cleaned datasets
2167     train_df = pd.read_csv('cleaned_train.csv')
2168     test_df = pd.read_csv('cleaned_test.csv')
2169
2170     # Create copies of the dataframes
2171     train_df_copy = train_df.copy()
2172     test_df_copy = test_df.copy()
2173
2174     # STEP 1: Create New Features
2175     train_df_copy['FamilySize'] = train_df_copy['SibSp'] + train_df_copy['Parch'] + 1
2176     test_df_copy['FamilySize'] = test_df_copy['SibSp'] + test_df_copy['Parch'] + 1
2177
2178     train_df_copy['IsAlone'] = (train_df_copy['FamilySize'] == 1).astype(int)
2179     test_df_copy['IsAlone'] = (test_df_copy['FamilySize'] == 1).astype(int)
2180
2181     age_bins = [0, 12, 18, 35, 60, 120]
2182     age_labels = ['Child', 'Teen', 'Adult', 'Senior', 'Elder']
2183     train_df_copy['AgeBins'] = pd.cut(train_df_copy['Age'], bins=age_bins, labels=
2184         age_labels, right=False)
2185     test_df_copy['AgeBins'] = pd.cut(test_df_copy['Age'], bins=age_bins, labels=age_labels
2186         , right=False)
2187
2188     train_df_copy['FarePerPerson'] = train_df_copy['Fare'] / train_df_copy['FamilySize']
2189     test_df_copy['FarePerPerson'] = test_df_copy['Fare'] / test_df_copy['FamilySize']
2190
2191     # Save the datasets with new features
2192     train_df_copy.to_csv('processed_train.csv', index=False)
2193     test_df_copy.to_csv('processed_test.csv', index=False)
2194
2195     # Load the processed datasets
2196     train_df = pd.read_csv('processed_train.csv')
2197     test_df = pd.read_csv('processed_test.csv')
2198
2199     # Define functions to extract ticket prefix and cabin deck
2200     def extract_ticket_prefix(ticket):
2201         parts = ticket.split()
2202         return parts[0] if not parts[0].isdigit() else 'None'
2203
2204     def extract_cabin_deck(cabin):
2205         return cabin[0] if pd.notna(cabin) else 'Unknown'
2206
2207     # Extract TicketPrefix and CabinDeck
2208     train_df['TicketPrefix'] = train_df['Ticket'].apply(extract_ticket_prefix)
2209     test_df['TicketPrefix'] = test_df['Ticket'].apply(extract_ticket_prefix)
2210
2211     train_df['CabinDeck'] = train_df['Cabin'].apply(extract_cabin_deck) if 'Cabin' in
2212         train_df.columns else 'Unknown'
2213     test_df['CabinDeck'] = test_df['Cabin'].apply(extract_cabin_deck) if 'Cabin' in
2214         test_df.columns else 'Unknown'
2215
2216     # Save the datasets with extracted features
2217     train_df.to_csv('processed_train.csv', index=False)
2218     test_df.to_csv('processed_test.csv', index=False)

```

E.6 MODEL BUILDING, VALIDATION, AND PREDICTION

In this phase, we conduct a comprehensive analysis of the Titanic passenger dataset with the aim of predicting passengers' survival probabilities. Initially, the data undergo preprocessing that included filling missing values, deleting columns with high missingness, and handling outliers. Subsequent feature engineering efforts introduce new attributes such as family size, solitary travel, age groupings, and fare per person, and involved encoding for gender and embarkation points. Furthermore, a random forest model is employed, optimized via grid search, and evaluated using cross-validation. Predictions are then made on the test set, and a submission file is prepared.

```

2214
2215 Model Building, Validation, and Prediction Plan
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

    #### Detailed Plan

    ##### Task 1: Prepare Training Data
    **Objective:** Separate the target variable and remove non-numeric columns.

    1. **Separate Target Column**
        - **Action:** Extract the 'Survived' column from 'processed_train.csv' as 'y'.
        - **Tool:** pandas
        - **Expected Output:** 'y' as a separate pandas Series containing the target variable.
        - **Code Example:**
            ```python
 import pandas as pd
 train_data = pd.read_csv('/mnt/d/PythonProjects/AutoKaggleMaster/multi_agents/competition/titanic/processed_train.csv')
 y = train_data['Survived']
            ```

    2. **Remove Non-Numeric Columns**
        - **Action:** Identify and drop non-numeric columns from the training set.
        - **Features Involved:** 'Name', 'Ticket', 'TicketPrefix', 'CabinDeck'
        - **Tool:** pandas
        - **Expected Output:** 'X_train' as a DataFrame containing only numeric columns.
        - **Code Example:**
            ```python
 X_train = train_data.drop(columns=['Survived', 'Name', 'Ticket', 'TicketPrefix', 'CabinDeck'])
            ```

    ##### Task 2: Prepare Test Data
    **Objective:** Ensure the test data has the same structure as the training data.

    1. **Remove Non-Numeric Columns**
        .....

    ##### Task 3: Train and Validate the Model
    **Objective:** Train and validate a machine learning model using the prepared datasets

    1. **Model Selection and Training**
        .....

    2. **Hyperparameter Tuning**
        .....

    ##### Task 4: Make Predictions
    **Objective:** Use the trained model to make predictions on the test dataset.

    1. **Prediction**
        .....

    ##### Task 5: Prepare Submission File
    **Objective:** Prepare the submission file in the required format.

    1. **Submission File Preparation**
        .....

    #### Final Notes
    - Ensure that all steps are executed in sequence to maintain data integrity.
    - Validate each step with print statements or logs to confirm successful execution.
    - Use cross-validation to ensure model robustness and avoid overfitting.

    Feel free to execute the tasks based on this detailed plan. If you have any questions or need further adjustments, don't hesitate to ask.

```

Model Building, Validation, and Prediction Code

```
2268
2269
2270
2271 import pandas as pd
2272 # Load the processed training data
2273 processed_train_path = '/mnt/d/PythonProjects/AutoKaggleMaster/multi_agents/
2274     competition/titanic/processed_train.csv'
2275 train_df = pd.read_csv(processed_train_path)
2276
2277 # Separate the target variable
2278 y = train_df['Survived']
2279
2280 # Identify non-numeric columns to be dropped
2281 columns_to_drop = ['Survived', 'Name', 'Ticket', 'TicketPrefix', 'CabinDeck', 'AgeBins
2282     ']
2283
2284 # Drop the target variable and non-numeric columns
2285 X_train = train_df.drop(columns=columns_to_drop)
2286
2287
2288 # Load the processed test data
2289 processed_test_path = '/mnt/d/PythonProjects/AutoKaggleMaster/multi_agents/competition
2290     /titanic/processed_test.csv'
2291 test_df = pd.read_csv(processed_test_path)
2292
2293 # Drop non-numeric columns to match X_train
2294 X_test = test_df.drop(columns=['Name', 'Ticket', 'TicketPrefix', 'CabinDeck', 'AgeBins
2295     '])
2296
2297 # Ensure the column order matches X_train
2298 X_test = X_test[X_train.columns]
2299
2300 print(f"Test data processed. X_test shape: {X_test.shape}")
2301
2302
2303 from sklearn.ensemble import RandomForestClassifier
2304 from sklearn.model_selection import GridSearchCV, cross_val_score
2305
2306 # Define the model and parameter grid
2307 model = RandomForestClassifier(random_state=42)
2308 param_grid = {
2309     'n_estimators': [100, 200, 300],
2310     'max_depth': [None, 10, 20, 30]
2311 }
2312
2313 # Use GridSearchCV for hyperparameter tuning
2314 grid_search = GridSearchCV(model, param_grid, cv=5, scoring='accuracy')
2315 grid_search.fit(X_train, y)
2316
2317 # Get the best model
2318 best_model = grid_search.best_estimator_
2319
2320 # Validate the model using cross-validation
2321 cv_scores = cross_val_score(best_model, X_train, y, cv=5, scoring='accuracy')
2322
2323
2324 # Make predictions on the test data
2325 predictions = best_model.predict(X_test)
2326
2327 # Convert predictions to a DataFrame
2328 predictions_df = pd.DataFrame({
2329     'PassengerId': test_df['PassengerId'],
2330     'Survived': predictions
2331 })
2332
2333 print(f"Predictions made. Sample predictions:\n{predictions_df.head()}")
2334
2335
2336 # Prepare the submission file
2337 submission_path = '/mnt/d/PythonProjects/AutoKaggleMaster/multi_agents/competition/
2338     titanic/submission.csv'
2339 predictions_df.to_csv(submission_path, index=False)
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
3000
3001
3002
3003
3004
3005
3006
3007
3008
3009
3010
3011
3012
3013
3014
3015
3016
3017
3018
3019
3020
3021
3022
3023
3024
3025
3026
3027
3028
3029
3030
3031
3032
3033
3034
3035
3036
3037
3038
3039
3040
3041
3042
3043
3044
3045
3046
3047
3048
3049
3050
3051
3052
3053
3054
3055
3056
3057
3058
3059
3060
3061
3062
3063
3064
3065
3066
3067
3068
3069
3070
3071
3072
3073
3074
3075
3076
3077
3078
3079
3080
3081
3082
3083
3084
3085
3086
3087
3088
3089
3090
3091
3092
3093
3094
3095
3096
3097
3098
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
3180
3181
3182
3183
3184
3185
3186
3187
3188
3189
3190
3191
3192
3193
3194
3195
3196
3197
3198
3199
3200
3201
3202
3203
3204
3205
3206
3207
3208
3209
3210
3211
3212
3213
3214
3215
3216
3217
3218
3219
3220
3221
3222
3223
3224
3225
3226
3227
3228
3229
3230
3231
3232
3233
3234
3235
3236
3237
3238
3239
3240
3241
3242
3243
3244
3245
3246
3247
3248
3249
3250
3251
3252
3253
3254
3255
3256
3257
3258
3259
3260
3261
3262
3263
3264
3265
3266
3267
3268
3269
3270
3271
3272
3273
3274
3275
3276
3277
3278
3279
3280
3281
3282
3283
3284
3285
3286
3287
3288
3289
3290
3291
3292
3293
3294
3295
3296
3297
3298
3299
3300
3301
3302
3303
3304
3305
3306
3307
3308
3309
3310
3311
3312
3313
3314
3315
3316
3317
3318
3319
3320
3321
3322
3323
3324
3325
3326
3327
3328
3329
3330
3331
3332
3333
3334
3335
3336
3337
3338
3339
3340
3341
3342
3343
3344
3345
3346
3347
3348
3349
3350
3351
3352
3353
3354
3355
3356
3357
3358
3359
3360
3361
3362
3363
3364
3365
3366
3367
3368
3369
3370
3371
3372
3373
3374
3375
3376
3377
3378
3379
3380
3381
3382
3383
3384
3385
3386
3387
3388
3389
3390
3391
3392
3393
3394
3395
3396
3397
3398
3399
3400
3401
3402
3403
3404
3405
3406
3407
3408
3409
3410
3411
3412
3413
3414
3415
3416
3417
3418
3419
3420
3421
3422
3423
3424
3425
3426
3427
3428
3429
3430
3431
3432
3433
3434
3435
3436
3437
3438
3439
3440
3441
3442
3443
3444
3445
3446
3447
3448
3449
3450
3451
3452
3453
3454
3455
3456
3457
3458
3459
3460
3461
3462
3463
3464
3465
3466
3467
3468
3469
3470
3471
3472
3473
3474
3475
3476
3477
3478
3479
3480
3481
3482
3483
3484
3485
3486
3487
3488
3489
3490
3491
3492
3493
3494
3495
3496
3497
3498
3499
3500
3501
3502
3503
3504
3505
3506
3507
3508
3509
3510
3511
3512
3513
3514
3515
3516
3517
3518
3519
3520
3521
3522
3523
3524
3525
3526
3527
3528
3529
3530
3531
3532
3533
3534
3535
3536
3537
3538
3539
3540
3541
3542
3543
3544
3545
3546
3547
3548
3549
3550
3551
3552
3553
3554
3555
3556
3557
3558
3559
3560
3561
3562
3563
3564
3565
3566
3567
3568
3569
3570
3571
3572
3573
3574
3575
3576
3577
3578
3579
3580
3581
3582
3583
3584
3585
3586
3587
3588
3589
3590
3591
3592
3593
3594
3595
3596
3597
3598
3599
3600
3601
3602
3603
3604
3605
3606
3607
3608
3609
3610
3611
3612
3613
3614
3615
3616
3617
3618
3619
3620
3621
3622
3623
3624
3625
3626
3627
3628
3629
3630
3631
3632
3633
3634
3635
3636
3637
3638
3639
3640
3641
3642
3643
3644
3645
3646
3647
3648
3649
3650
3651
3652
3653
3654
3655
3656
3657
3658
3659
3660
3661
3662
3663
3664
3665
3666
3667
3668
3669
3670
3671
3672
3673
3674
3675
3676
3677
3678
3679
3680
3681
3682
3683
3684
3685
3686
3687
3688
3689
3690
3691
3692
3693
3694
3695
3696
3697
3698
3699
3700
3701
3702
3703
3704
3705
3706
3707
3708
3709
3710
3711
3712
3713
3714
3715
3716
3717
3718
3719
3720
3721
3722
3723
3724
3725
3726
3727
3728
3729
3730
3731
3732
3733
3734
3735
3736
3737
3738
3739
3740
3741
3742
3743
3744
3745
3746
3747
3748
3749
3750
3751
3752
3753
3754
3755
3756
3757
3758
3759
3760
3761
3762
3763
3764
3765
3766
3767
3768
3769
3770
3771
3772
3773
3774
3775
3776
3777
3778
3779
3780
3781
3782
3783
3784
3785
3786
3787
3788
3789
3790
3791
3792
3793
3794
3795
3796
3797
3798
3799
3800
3801
3802
3803
3804
3805
3806
3807
3808
3809
3810
3811
3812
3813
3814
3815
3816
3817
3818
3819
3820
3821
3822
3823
3824
3825
3826
3827
3828
3829
3830
3831
3832
3833
3834
3835
3836
3837
3838
3839
3840
3841
3842
3843
3844
3845
3846
3847
3848
3849
3850
3851
3852
3853
3854
3855
3856
3857
3858
3859
3860
3861
3862
3863
3864
3865
3866
3867
3868
3869
3870
3871
3872
3873
3874
3875
3876
3877
3878
3879
3880
3881
3882
3883
3884
3885
3886
3887
3888
3889
3890
3891
3892
3893
3894
3895
3896
3897
3898
3899
3900
3901
3902
3903
3904
3905
3906
3907
3908
3909
3910
3911
3912
3913
3914
3915
3916
3917
3918
3919
3920
3921
3922
3923
3924
3925
3926
3927
3928
3929
3930
3931
3932
3933
3934
3935
3936
3937
3938
3939
3940
3941
3942
3943
3944
3945
3946
3947
3948
3949
3950
3951
3952
3953
3954
3955
3956
3957
3958
3959
3960
3961
3962
3963
3964
3965
3966
3967
3968
3969
3970
3971
3972
3973
3974
3975
3976
3977
3978
3979
3980
3981
3982
3983
3984
3985
3986
3987
3988
3989
3990
3991
3992
3993
3994
3995
3996
3997
3998
3999
4000
4001
4002
4003
4004
4005
4006
4007
4008
4009
4010
4011
4012
4013
4014
4015
4016
4017
4018
4019
4020
4021
4022
4023
4024
4025
4026
4027
4028
4029
4030
4031
4032
4033
4034
4035
4036
4037
4038
4039
4040
4041
4042
4043
4044
4045
4046
4047
4048
4049
4050
4051
4052
4053
4054
4055
4056
4057
4058
4059
4060
4061
4062
4063
4064
4065
4066
4067
4068
4069
4070
4071
4072
4073
4074
4075
4076
4077
4078
4079
4080
4081
4082
4083
4084
4085
4086
4087
4088
4089
4090
4091
4092
4093
4094
4095
4096
4097
4098
4099
4100
4101
4102
4103
4104
4105
4106
4107
4108
4109
4110
4111
4112
4113
4114
4115
4116
4117
4118
4119
4120
4121
4122
4123
4124
4125
4126
4127
4128
4129
4130
4131
4132
4133
4134
4135
4136
4137
4138
4139
4140
4141
4142
4143
4144
4145
4146
4147
4148
4149
4150
4151
4152
4153
4154
4155
4156
4157
4158
4159
4160
4161
4162
4163
4164
4165
4166
4167
4168
4169
4170
4171
4172
4173
4174
4175
4176
4177
4178
4179
4180
4181
4182
4183
4184
4185
4186
4187
4188
4189
4190
4191
4192
4193
4194
4195
4196
4197
4198
4199
4200
4201
4202
4203
4204
4205
4206
4207
4208
4209
4210
4211
4212
4213
4214
4215
4216
4217
4218
4219
4220
4221
4222
4223
4224
4225
4226
4227
4228
4229
4230
4231
4232
4233
4234
4235
4236
4237
4238
4239
4240
4241
4242
4243
4244
4245
4246
4247
4248
4249
4250
4251
4252
4253
4254
4255
4256
4257
4258
4259
4260
4261
4262
4263
4264
4265
4266
4267
4268
4269
4270
4271
4272
4273
4274
4275
4276
4277
4278
4279
4280
4281
4282
4283
4284
4285
4286
4287
4288
4289
4290
4291
4292
4293
4294
4295
4296
4297
4298
4299
4300
4301
4302
4303
4304
4305
4306
4307
4308
4309
4310
4311
4312
4313
4314
4315
4316
4317
4318
4319
4320
4321
4322
4323
4324
4325
4326
4327
4328
4329
4330
4331
4332
4333
4334
4335
4336
4337
4338
4339
4340
4341
4342
4343
4344
4345
4346
4347
4348
4349
4350
4351
4352
4353
4354
4355
4356
4357
4358
4359
4360
4361
4362
4363
4364
4365
4366
4367
4368
4369
4370
4371
4372
4373
4374
4375
4376
4377
4378
4379
4380
4381
4382
4383
4384
4385
4386
4387
4388
4389
4390
4391
4392
4393
4394
4395
4396
4397
4398
4399
4400
4401
4402
4403
4404
4405
4406
4407
4408
4409
4410
4411
4412
4413
4414
4415
4416
4417
4418
4419
4420
4421
4422
4423
4424
4425
4426
4427
4428
4429
4430
4431
4432
4433
4434
4435
4436
4437
4438
4439
4440
4441
4442
4443
4444
4445
4446
4447
4448
4449
4450
4451
4452
4453
4454
4455
4456
4457
4458
4459
4460
4461
4462
4463
4464
4465
4466
4467
4468
4469
4470
4471
4472
4473
4474
4475
4476
4477
4478
4479
4480
4481
4482
4483
4484
4485
4486
4487
4488
4489
4490
4491
4492
4493
4494
4495
4496
4497
4498
4499
4500
4501
4502
4503
4504
4505
4506
4507
4508
4509
4510
4511
4512
4513
4514
4515
4516
4517
4518
4519
4520
4521
4522
4523
4524
4525
4526
4527
4528
4529
4530
4531
4532
4533
4534
4535
4536
4537
4538
4539
4540
4541
4542
4543
4544
4545
4546
4547
4548
4549
4550
4551
4552
4553
4554
4555
4556
4557
4558
4559
4560
4561
4562
4563
4564
4565
4566
4567
4568
4569
4570
4571
4572
4573
4574
4575
4576
4577
4578
4579
4580
4581
4582
4583
4584
4585
4586
4587
4588
4589
4590
4591
4592
4593
4594
4595
4596
4597
4598
4599
4600
4601
4602
4603
4604
4605
4606
4607
4608
4609
4610
4611
4612
4613
4614
4615
4616
4617
4618
4619
4620
4621
4622
4623
4624
4625
4626
4627
4628
4629
4630
4631
4632
4633
46
```

2322 E.7 RESEARCH REPORT
2323

2324 After completing the entire data science pipeline, AutoKaggle reviews the interim reports from each
2325 phase and finally compiles a Research Report to document its key findings, major decisions, and their
2326 rationale at every phase. This helps users better understand the solutions provided by AutoKaggle.

2327 **Research Report**
2328

```
2329
2330 # COMPETITION RESEARCH REPORT
2331
2332 ## 1. PRELIMINARY EDA
2333
2334 The preliminary exploratory data analysis (EDA) involved examining the dataset's
2335 structure and identifying key features. The analysis revealed critical insights
2336 such as:
2337 - Missing Values: Notably, the 'Age' and 'Cabin' features had significant missing
2338 values.
2339 - Outliers: Outliers were identified in the 'Fare' and 'Age' features, with fares
2340 exceeding 200 and ages above 60.
2341 - Statistical Summary: The median age was around 30, while the median fare was
2342 approximately 14.45.
2343 ...
2344
2345 ## 2. DATA CLEANING
2346
2347 The data cleaning phase focused on addressing missing values and outliers. Key actions
2348 included:
2349 - Imputation:
2350 - 'Age': Missing values were filled with the mean.
2351 - 'Cabin': Filled with 'Missing' to indicate absence.
2352 - 'Embarked': Missing values filled with the mode.
2353 - Outlier Treatment:
2354 - 'Fare' and 'Age' were clipped using the IQR method to reduce the influence of
2355 outliers.
2356 ...
2357
2358 ## 3. DEEP EDA
2359
2360 In-depth EDA revealed valuable insights about the relationships between features and
2361 survival rates:
2362 - Pclass: Higher classes showed higher survival rates; average Pclass was 2.31.
2363 - Sex: Females had a significantly higher survival rate compared to males.
2364 - Fare: Higher fares correlated positively with survival, indicating a potential
2365 link to better accommodation and safety.
2366 ...
2367
2368 ## 4. FEATURE ENGINEERING
2369
2370 Feature engineering involved creating and transforming features to enhance model
2371 performance:
2372 - New Features Created:
2373 - Title: Extracted from 'Name'.
2374 - FamilySize: Summation of 'SibSp' and 'Parch'.
2375 - IsAlone: Binary feature indicating if a passenger traveled alone.
2376 - FarePerPerson: Calculated fare per individual in a family.
2377 ...
2378
2379 ## 5. MODEL BUILDING, VALIDATION, AND PREDICTION
2380
2381 Multiple models were trained during this phase, including:
2382 - Models: XGBoost, SVM, Random Forest, Decision Tree, and Logistic Regression.
2383 - Best Model: Random Forest achieved the highest validation score of 0.8379.
2384 ...
2385
2386 ## 6. CONCLUSION
2387
2388 The competition's approach involved a structured process of EDA, data cleaning,
2389 feature engineering, and model evaluation. Key insights included the strong
2390 influence of 'Sex', 'Pclass', and 'Fare' on survival rates. The most impactful
2391 decisions involved addressing missing values and outliers, which collectively
2392 improved data quality and model accuracy. Future recommendations include further
2393 feature engineering, hyperparameter tuning, and validation of feature importance
2394 to enhance model performance.
```

2374

2375