JOÃO PAULO PAIVA LIMA

HOW GOOD LUSOPHONES ARE DATA SCIENCE LLM AGENTS?: EVALUATING AGENTIC APPROACHES FOR DATA SCIENCE IN PORTUGUESE CONTEXTS

LAVRAS – MG

2026

JOÃO PAULO PAIVA LIMA

HOW GOOD LUSOPHONES ARE DATA SCIENCE LLM AGENTS?: EVALUATING AGENTIC APPROACHES FOR DATA SCIENCE IN PORTUGUESE CONTEXTS

Master's Thesis presented to the Federal University of Lavras as part of the requirements of the Postgraduate Program in Computer Science, in the field of Artificial Intelligence, for the attainment of the Master's degree.

Prof. Dr. Denilson Alves Pereira
Advisor

Profa. Dra. Marluce Rodrigues Pereira
Co-advisor

LAVRAS – MG
2026

Ficha Catalográfica elaborada pelo Sistema de Geração de Ficha Catalográfica da Biblioteca Universitária da UFLA, com dados informados pelo(a) próprio(a) autor(a).

Lima, João Paulo Paiva.
How Good Lusophones Are Data Science Llm Agents? : Evaluating Agentic Approaches for Data Science in Portuguese Contexts / João Paulo Paiva Lima. - 2026.
110 p. : il.

Orientador: Denilson Alves Pereira
Coorientadora: Marluce Rodrigues Pereira

Dissertação (Mestrado Acadêmico) - Universidade Federal de Lavras, 2026.
Bibliografia.

1. Grandes Modelos de Linguagem. 2. Ciência de Dados. 3. Aprendizado de Máquina. 4. Português. 5. Processamento de Linguagem Natural. I. Pereira, Denilson Alves. II. Pereira, Marluce Rodrigues. III. Universidade Federal de Lavras. IV. Título.

JOÃO PAULO PAIVA LIMA

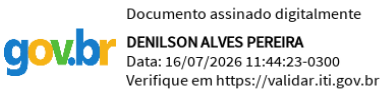
HOW GOOD LUSOPHONES ARE DATA SCIENCE LLM AGENTS?: EVALUATING AGENTIC APPROACHES FOR DATA SCIENCE IN PORTUGUESE CONTEXTS

AGENTES DE LLMS PARA CIÊNCIA DE DADOS SÃO BONS LUSÓFONOS?: AVALIANDO TÉCNICAS BASEADAS EM AGENTES PARA CIÊNCIA DE DADOS EM CONTEXTOS DE LÍNGUA PORTUGUESA

Master's Thesis presented to the Federal University of Lavras as part of the requirements of the Postgraduate Program in Computer Science, in the field of Artificial Intelligence, for the attainment of the Master's degree.

ACCEPTED on May 22, 2026.

Prof. Dr. Denilson Alves Pereira	UFLA
Prof. Dr. Luiz Henrique de Campos Merschmann	UFLA
Profa. Dra. Marluce Rodrigues Pereira	UFLA
Prof. Dr. Wladmir Cardoso Brandão	PUC Minas



Prof. Dr. Denilson Alves Pereira
Advisor

Profa. Dra. Marluce Rodrigues Pereira
Co-advisor

LAVRAS – MG
2026

Dedico este trabalho aos meus pais, Nilson e Sônia (ordem alfabética), e aos familiares e amigos que me ajudaram nessa jornada.

ACKNOWLEDGMENTS

I would like to thank my advisor Prof. Denilson, my co-advisor Prof. Marluce, for their guidance. Also a special thanks to the Federal University of Lavras (UFLA) for the support. This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001.

"Why did the LLM get kicked out of the data science team?

*Because it kept predicting the future and getting it **wrong**."*

Qwen3 4B

(prompted to tell an unimaginably funny, punchy, laugh-out-loud-hilarious joke about LLMs and data science.)

RESUMO

A Ciência de Dados (CD), um campo complexo que envolve conhecimentos avançados de estatística, matemática e programação, tem sido historicamente restrita a profissionais humanos profundamente qualificados. Esse paradigma foi desafiado por avanços recentes no raciocínio artificial e pela expansão do conjunto de ferramentas dos grandes modelos de linguagem (LLMs). Mais especificamente, o desempenho da CD automatizada foi impulsionado para níveis próximos aos humanos por frameworks agentes, nos quais ferramentas como execução de código e busca de documentos são fornecidas aos LLMs. Entretanto, a pesquisa atual sobre métodos baseados em LLM para CD automatizada ainda é fortemente enviesada em favor da língua inglesa. Neste trabalho, é apresentada uma avaliação de CD utilizando agentes para prompts, dados e metadados em português em comparação ao inglês. Para esse fim, foi desenvolvido o DATAEXPLAINER, um framework de agente voltado à geração de soluções de CD com explicações passo-a-passo no idioma requerido pelo usuário. Com o framework desenvolvido, quatro modelos diferentes de LLM foram avaliados em um conjunto de sete competições coletadas do site Kaggle e traduzidas. Soluções coerentes em português foram produzidas por todos os LLMs avaliados (com mais de 97% do texto gerado no idioma correto), com o agente pareado ao modelo GPT-OSS superando o usuário médio do site Kaggle em cinco das sete competições avaliadas. Houve uma lacuna de desempenho entre as línguas, com pontuações médias 10,1% menores para o português em comparação ao inglês; contudo, a diferença não alcançou a significância estatística. Essa diferença em pontuação tendeu a ser maior para tarefas de regressão e para modelos ajustados para código e menor para modelos de raciocínio.

Palavras-chave: Grandes Modelos de Linguagem; Ciência de Dados; Aprendizado de Máquina; Português; Processamento de Linguagem Natural

ABSTRACT

Data Science (DS), a complex field often involving advanced concepts of statistics, mathematics, and programming, has historically been restricted to deeply knowledgeable human professionals. Recent advancements in artificial reasoning and the expanding toolset of Large Language Models (LLMs) have challenged this paradigm. More specifically, agentic frameworks, which provide LLMs tools like code execution and document searching, have pushed automated DS performance to near-human levels. However, current research on LLM-based methods for automated DS is still heavily biased towards the English language. In this work, we provide an assessment for agentic DS for prompting, data, and metadata in Portuguese in comparison to English. To this end, we expanded on previous advancements to develop DATAEXPLAINER, an agentic framework for generating DS solutions with step-by-step explanations in the target language. We employed the aforementioned framework to evaluate four different LLM models on a set of seven different translated Kaggle competitions. All assessed LLMs produced coherent Portuguese solutions (more than 97% of text generated in the correct language), with the agent powered by GPT-OSS surpassing the average Kaggle user on five out of the seven evaluated competitions. There is still a gap in performance, with 10.1% lower mean scores for Portuguese across models when compared to English; however, the difference did not reach statistical significance. Additionally, the data indicates a larger gap for regression tasks and code-tuned models, and a smaller gap for thinking models.

Keywords: Large Language Models; Data Science; Machine Learning; Portuguese; Natural Language Processing.

INDICADORES DE IMPACTO

Este trabalho tem como foco o uso de tecnologias de vanguarda para a automação de análise de dados, um trabalho comumente feito de forma manual. Dessa maneira, identificam-se duas principais frentes de impacto: a tecnológica, relativa ao avanço do conhecimento técnico e suas aplicações; e a social, referente às implicações da substituição do trabalho humano pela tecnologia. No aspecto tecnológico, o estudo apresentado é uma avaliação do estado da arte dos algoritmos de análise automatizada quando aplicados à língua portuguesa. Essa sendo uma área que ainda carece de avaliações profundas, o estudo contribui para a evolução do conhecimento científico. Em especial, pelo foco em ferramentas de código aberto compatíveis com componentes a nível de consumidor, a pesquisa apresenta, também, um caráter de valorização do desenvolvimento acessível e sustentável. Quanto aos impactos sociais, é fundamental considerar como as ferramentas de automação afetam as relações trabalhistas. Embora esta pesquisa possa ser interpretada como parte de um processo que agrava disparidades de classe, a automação do trabalho intelectual, esse trabalho tem um caráter majoritariamente avaliativo. Ao estudar as limitações dos agentes de IA—ferramentas cada vez mais populares no mercado—como a subrepresentação de determinadas línguas e falta de clareza nos processos utilizados, este estudo busca contribuir, principalmente, para a sua aplicação da maneira mais segura e responsável possível.

IMPACT INDICATORS

In this work, we focus on the use of cutting-edge technologies for the automation of data analysis, a task commonly performed manually. Thus, we identify two major spheres of impact: the technological, related to the advancement of technical knowledge and its applications; and the social, referring to the implications of replacing human intellectual labor with technology. In the technological sphere, our study evaluates the state of the art of automated analysis algorithms when applied to the Portuguese language. As this area still lacks in-depth evaluations, we contribute to the evolution of scientific knowledge. In particular, through our focus on open-source tools compatible with consumer-grade hardware, our research also promotes sustainable and accessible development. Regarding social impacts, it is fundamental to consider how automation tools affect labor relations. Although our research might be interpreted as part of a process that exacerbates class disparities, specifically the automation of intellectual labor, this work is primarily evaluative in nature. By studying the limitations of AI agents—increasingly popular tools in the market—such as the underrepresentation of certain languages and the lack of clarity in the processes used, we seek to contribute mainly to their application in the safest and most responsible manner possible.

LIST OF FIGURES

LIST OF TABLES

LIST OF FRAMES

LIST OF ACRONYMS

AI	Artificial intelligence
API	Application programming interface
AutoML	Automatic machine learning
CoT	Chain-of-thought
DS	Data science
LLM	Large language model
MCTS	Monte-Carlo tree search
MLE	Machine learning engineering
NLP	Natural language processing
RAG	Retrieval augmented generation
RNN	Recurrent neural network
SD	Standard Deviation

CONTENTS

1 INTRODUCTION

Data science (DS) and analysis constitute a fundamental element in both science and technology. It enables the discovery of valuable insights within complex or seemingly unrelated information, providing a consistent, replicable, and robust framework for data management. This is particularly relevant in fields involving intricate systems and requiring a nuanced approach, such as medicine, epidemiology, physics, and chemistry (Provost; Fawcett, 2013).

In recent years, the scope, scale, and importance of data science and analysis have grown significantly. More specifically, computerized systems have led to a shift in how data is managed and analyzed. Digital storage technologies, such as hard drives and cloud platforms, are far more efficient, reliable, and easier to index than more traditional methods. The widespread adoption of the internet has revolutionized data accessibility, enabling rapid collection, sharing, and retrieval of information from almost anywhere in the world. Consequently, recent estimates suggest that the total amount of data on the web, encompassing files, images, videos, and more, could exceed 180 zettabytes by the end of 2025 (Taylor, 2024).

This unprecedented increase in data generation, storage, and availability gave rise to the phenomenon commonly referred to as Big Data (Russell; Norvig, 2022), characterized by vast, varied, and highly indexable datasets capable of holding valuable insights. However, the sheer scale and diversity of these datasets might present a significant challenge: since traditional data science is often a human-driven process, tailored by highly specialized professionals, it might be unable to handle such overwhelming volumes of information in a efficient manner.

In this context it is fundamental to develop more efficient and scalable solutions to handle such a deluge of data. Furthermore, given the overwhelming prevalence of the English language within data-centric fields, how can we ensure these solutions effectively address the vast linguistic diversity within datasets and offer benefits that extend beyond English-speaking contexts?

1.1 Data Volume Enhancing Modeling Capability

The higher volume of data, despite often being a source of complexity, has also led to the development of tools capable of handling diverse data types at larger scales. More specifically, the development of large language model (LLM)[1]—only possible due to the massive

amount of information available on the web— has been one the most impactful outcomes of the aforementioned data revolution (Liang *et al.*, 2025).

Since 2017, the transformer architecture (Vaswani *et al.*, 2017) has driven the evolution of LLMs. This novel approach to modeling sequential data—encompassing language (Lewis *et al.*, 2020), code (Rozière *et al.*, 2023), speech (Ao *et al.*, 2022), and images (Liu *et al.*, 2021)—is designed to process large amounts of information in parallel. Its horizontal scalability allows models to learn from exceptionally large datasets at a fraction of the time investment required by more traditional models, such as those based on recurrent neural networks (RNNs) (Vinyals *et al.*, 2015). Consequently, models built upon the Transformer architecture have achieved state-of-the-art (SOTA) performance across a wide range of Natural Language Processing (NLP) tasks (Devlin *et al.*, 2019; Lewis *et al.*, 2020; Brown *et al.*, 2020).

Furthermore, as demonstrated by OpenAI's GPT-3 and its successors—both closed-source (Anthropic, 2024; Gemini Team, 2024b; OpenAI, 2024) and open-source (Touvron *et al.*, 2023a; Touvron *et al.*, 2023b; Grattafiori *et al.*, 2024; DeepSeek-AI, 2024; Yang *et al.*, 2024)—if given sufficient data and computational resources, LLMs can replicate human-like language and code with impressive accuracy. These characteristics make LLMs promising candidates for streamlining data analysis.

1.2 Data Processing Limitations of LLMs

However, as innovative as LLMs are, they still struggle to process large and varied datasets on their own. Directly inputting large or highly dimensional datasets into LLMs is unlikely to yield satisfactory results. These models are primarily trained on cleaned versions of textual and visual data available on the web (OpenAI, 2024; DeepSeek-AI, 2024; Grattafiori *et al.*, 2024). This stands in stark contrast to the information found in fields that commonly require analysis, such as the vast tabular data in medical and financial reports, or the highly dimensional data in gene research. Additionally, scaling LLMs to efficiently handle extremely long sequences remains an open research problem, with most models having context windows capped at around one million input tokens (Yang *et al.*, 2025; Gemini Team, 2024a).

[1] Although the term may be used to refer to a variety of different tools, for the sake of simplicity, we will be referring specifically to the newer, decoder-only, generative models when mentioning LLMs.

Furthermore, the "black-box" nature of the underling neural networks[2] creates obstacles in ensuring the results are interpretable (Burns *et al.*, 2023). As recent studies suggest, even when accessing the same knowledge base, different models are still likely to generate conflicting information (Adlakha *et al.*, 2024). As follows, even if these tools possessed a superhuman capacity for analyzing complex data, their results would still lack the solid foundation provided by traditional analysis.

1.3 LLM Agents

A promising approach involves equipping language models with tools to perform traditional analysis, rather than relying on LLMs' capacity to extract insights through brute force. This strategy enables language models to follow more transparent, interpretable, and verifiable procedures, thereby leveraging their strengths in text and code generation while ensuring that the process is replicable. Such approach would be an example of the blossoming field of LLM agents, which focus on integrating models into structured, task-oriented, agentic frameworks (Huang *et al.*, 2024b).

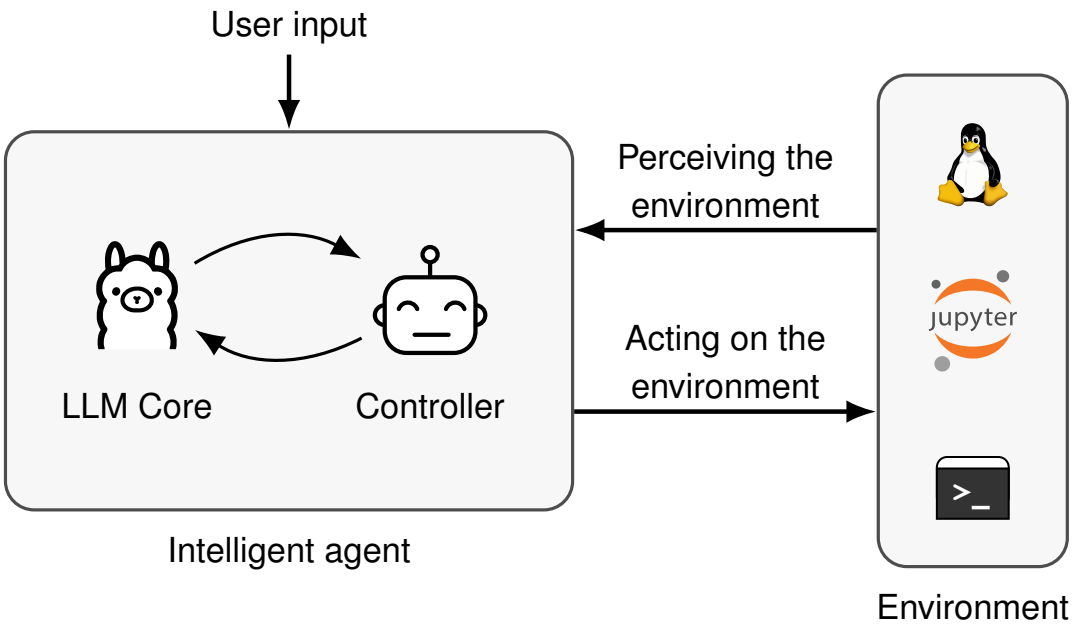
In the general field of artificial intelligence, agents can be understood as actors capable of perceiving an environment using a set of *sensors* and acting upon that environment using a set of *actuators*. More specifically, a rational agent is one that plans its actions to achieve a favorable outcome, closely aligned with its goal (Russell; Norvig, 2022).[3]

Figure 1.1 provides an overview of an intelligent LLM agent. The agent is composed of four main elements: the LLM core, the controller, the *sensors*, and the *actuators*. The LLM core serves as the reasoning component of the agent and can be any generative language model. The controller manages how the agent processes information and interacts with the LLM core. The *sensors* encompass any mechanism for capturing a response from an environment, such as text inputs, API connections, or screen recorders. Finally, the *actuators* include any means of eliciting a change in the environment, such as text outputs, code execution, or API calls. The environment, on the other hand, defines the operational context for the agent, which may include a Docker container (Chan *et al.*, 2025), web pages (Zhou *et al.*, 2023; He *et al.*, 2024),

[2] While some authors have highlighted instances of division of labor within some models (Vaswani *et al.*, 2017; Shen *et al.*, 2024), the underlying reasoning of these networks remains largely obscure.

[3] In the cited literature, the authors provide a more technical definition of a goal; for the sake of simplicity, we define a goal as "what we want the agent to accomplish".

Figure 1.1 – Diagram of an LLM agent



Source: Authors (2026)

Description for the Figure: A diagram illustrating the interaction between an intelligent agent and its environment. The intelligent agent receives user input. The agent then perceives the environment, represented by a vertical rectangular box containing a Tux penguin logo, the Jupyter logo, and a terminal window, via an arrow pointing from the environment box to the agent box. The agent acts upon the environment, indicated by another arrow extending from the agent box to the environment box.

entire operating systems (OpenAI, 2025; Xie *et al.*, 2024) or even the real world (Ichter *et al.*, 2022; Wu *et al.*, 2023)—if one is brave enough to give it eyes and limbs.

However, even within successful DS agentic frameworks (Tang *et al.*, 2024; Guo *et al.*, 2024; Hong *et al.*, 2025) there is still a strong focus on the English language for both applications and benchmarks, leaving other languages underrepresented. This disparity may hinder the development of solutions that are truly general and accessible.

1.4 Dealing with Linguistic Diversity

Handling different languages has long been a focus in natural language processing, with various studies explicitly dedicated to exploring this challenge (Conneau *et al.*, 2020; Liu *et al.*, 2020; Xue *et al.*, 2021). However, despite most flagship models from the past three years being capable of handling various dialects simultaneously (Grattafiori *et al.*, 2024; OpenAI, 2024; DeepSeek-AI, 2024; Yang *et al.*, 2024), the differing performance across languages is rarely investigated in detail.

The training corpora for these models are rarely made public; however, if assumed to be similar in proportions to those publicly available (Xue *et al.*, 2021; Abadji *et al.*, 2022; Nguyen *et al.*, 2023), English is by far the most represented language, accounting for anywhere from 15% to 45% of tokens. Evaluations of multi-language capacity are usually conducted on two datasets: a translated version (OpenAI, 2024) of the MMLU dataset (Hendrycks *et al.*, 2021) and the MGSM dataset (Shi *et al.*, 2022), both of which consist of objective questions on academic subjects. The models' ability to generate intelligent and varied outputs in languages other than English and Chinese is mostly left to be assessed by third-party researchers or in practice by users.

Considering code generation further complicates matters. While not as semantically intensive as other NLP tasks, code-focused tasks may be just as complex to handle in multiple languages. When writing code, a good LLM must not only produce correct code, but also make it clear and readable for the user. This would require the explanations about the algorithm, comments, variables, classes, and function names to be in the user's preferred language. In the case of a Portuguese prompt requesting code generation, the model would need to interpret the Portuguese input, relate it to its mostly English-based knowledge, and produce output that blends both languages, as English terms form the syntax of most programming languages and packages. In data science, this complexity is further exacerbated by the linguistic diversity of the data and metadata involved.

Finally, Goldman *et al.* (2025) showed that cross-lingual knowledge transfer on multi-language models is less than optimal. With the best performing model, GPT-4, managing to effectively transfer an average of only about 67% of its knowledge. Portuguese, specifically, falls behind most Indo-European languages tested when learning from an English language knowledge base.

1.5 The Portuguese Language and Brazil

With around 260 million native speakers, roughly 3.7% of the world's population, Portuguese is the mother tongue of the vast majority of Brazil, Portugal and São Tomé and Príncipe, as well as a major language in Angola, Mozambique and other African and Asian countries (Brazilian Educational and Language Travel Association, 2023).

Brazil has by far the largest Portuguese-speaking population, accounting for around 77% of the world's lusophones. As a founding member of two major intergovernmental organizations—Mercosur and BRICS— Brazil's 2024 gross domestic product (GDP) was estimated by the International Monetary Fund (IMF) to be approximately 2.1 trillion US dollars (International Monetary Fund, 2024). Despite its large population and international importance, Brazil's scientific output lags behind other nations of comparable size and economic stature, failing to rank among the top ten in terms of the number of publications (Oliveira *et al.*, 2022). Limited access to international resources may contribute to this disparity, as English—often regarded as the lingua franca of science and technology (Grier, 2017)—is fluently spoken by less than 1% of Brazil's population, as estimated in a 2014 publication by the British Council (British Council and Data Popular institute, 2014). This language barrier presents significant challenges for Brazilian researchers seeking to publish in high-impact international journals or collaborate with global peers.

Given these limitations imposed by language and the English-centric focus of current AI development, it is crucial to assess the accessibility of automated data science for the Brazilian population and the Lusophone world as a whole.

1.6 Contributions and Outline

In light of the importance of automated data science and its largely unquantified state for Portuguese, we aim to assess LLMs' agentic capabilities for lusophone data science, focusing specifically on replicable machine learning engineering (MLE). To this end, we developed a new lusophone DS evaluation set, consisting of seven open-ended machine learning competitions, sourced and translated from the Kaggle website[4]. The dataset was then employed to evaluate various LLM agents on lassification and regression tasks, with performance quantified by their scores on said competitions. Additionally, to validate the LLMs on the clarity of their processes, we developed DATAEXPLAINER, a new agentic framework focused on guiding models towards descriptive step-by-step solutions. The framework allowed a deeper evaluation of the agent's linguistic consistency, by measuring the extent to which they deviate from the expected language when generating outputs.

[4] https://www.kaggle.com

We believe this work contributes to the UN's 2030 Agenda for Sustainable Development by evaluating and working to amend the under-representation of the Portuguese language in LLM research, as well as working towards sustainable AI development by focusing on more efficient, desktop-sized, LLMs.

Hence, the key contributions from this work are:

a) The development and assessment of a new Portuguese data science evaluation set based on public MLE competitions.
b) The development and evaluation of a new agentic framework for automated DS, named DATAEXPLAINER.
c) An evaluation of four varied LLMs for agentic machine learning engineering in English and Portuguese.
d) A quantitative assessment of LLM agents' linguistic consistency when performing analysis, which we define as the percentage of the generated text and code comments that is in the user's preferred language.

The following sections proceed as follows: Section 2 provides an overview of the current literature on LLMs and their agentic applications; Section 3 presents our process for creating the evaluation set, along with the agentic and evaluation setups used; Section 4 presents results. Finally, Section 5 and Section 6 present our final discussions and conclusions.

2 LITERATURE REVIEW

The field of artificial intelligence and large language models has experienced an unprecedented surge in publications over the past five years. To provide context for our work, we present an overview of the current literature relevant to our research. We begin by discussing the current landscape of available LLMs, which underpins advancements in complex reasoning and agentic AI. Subsequently, we explore emerging applications in automated data science and conclude with an examination of LLM performance on Portuguese data.

2.1 Complex Reasoning

Despite LLMs impressive capacity for emulating human language, they were unlikely to employ causal thinking until very recently. Research into complex reasoning in LLMs specifically addresses this issue. By employing clever prompting, exploring different thought paths, and utilizing targeted fine-tuning, researchers have been able to drastically improve performance in complex tasks, such as solving multi-step mathematical problems, ciphers, and planning data science projects (OpenAi-Team, 2024).

In the current literature, the most common inference-time techniques to improve reasoning involve prompt engineering, chain-of-thought (CoT) and ensemble methods. Prompt engineering (Brown *et al.*, 2020; Wei *et al.*, 2022) involves designing effective prompts to guide the model toward accurate and relevant outputs. By structuring prompts with explicit instructions, examples, or constraints, researchers can direct the model toward more fruitful reasoning paths. This approach is simple and effective, but may not be easily generalizable. CoT (Wei *et al.*, 2022; Yao *et al.*, 2023) extends prompting by encouraging the model to generate intermediate reasoning steps, mimicking human problem-solving processes. This approach is particularly effective for tasks demanding multi-step logical reasoning, such as mathematical problem-solving or complex decision-making, however, models may often become trapped in suboptimal lines of thought. To ensure that not only one, possibly erroneous, reasoning path is taken, ensemble methods aggregate outputs from multiple models or chains-of-thought when generating an answer. Techniques such as self-consistency (Wang *et al.*, 2023), in which the model generates multiple reasoning chains and selects the most consistent answer, help reduce

errors and increase confidence in the final output. On the other hand, a downside of this process is that it often results in a significantly higher token count and inference cost.

More recent advancements have explored fine-tuning models to achieve native reasoning. In late 2024, OpenAI's o1 (OpenAi-Team, 2024) was launched; being the first of its kind, o1 achieved consistent SOTA scores on most complex reasoning benchmarks. In early 2025, DeepSeek released its R1 LLM (DeepSeek-AI *et al.*, 2025), a cost-effective open-source model for complex reasoning. Thanks to its novel reinforcement learning fine-tuning process, R1 displays a performance comparable to o1, despite being developed on a much tighter budget. Also in early 2025, Google introduced its second generation of Gemini models (DeepMind, 2025), with Gemini 2.0 Flash Thinking—its complex reasoning model— immediately assuming first place at the Chatbot Arena (Chiang *et al.*, 2024), a crowd-sourced benchmark for LLMs. These breakthroughs in causal thinking were central to the development of more intricate arrangements of tool-integrated LLMs, such as agent architectures.

2.2 LLM Agents

While Section 1.3 provides an overview of LLM agents, practical implementations often employ more complex structures to maximize the potential of the core language models (Huang *et al.*, 2024b; Wang *et al.*, 2024b). A summary of the three most common points of complexity include:

a) **base prompts:** like most applications utilizing LLMs, agents often require specific prompting to achieve their best performance. The base prompts are frequently employed to familiarize the LLM core to its tasks, goals, roles, capabilities, and to provide few-shot examples (He *et al.*, 2024; Qian *et al.*, 2024; Chan *et al.*, 2025). For instance, MetaGPT (Hong *et al.*, 2024) employed a multi-agent setup to assemble an AI software development team. Each of the agents in the team was prompted with a specific role (for example, Product Manager, Architect or Engineer), its possible operations (such as writing or reviewing code), and the team's workflow. Figure 2.1 provides an example of a base prompt for a simple cashier agent;

b) **memory:** Agents frequently interact with the environment, generate outputs, and process task prompts over extended periods. This prolonged interaction significantly increases the load on the limited context window, potentially leading to the loss of

Figure 2.1 – Example of a base prompt for a cashier agent.

```
You are a helpful cashier agent. Your task is to assist
 customers with their purchases.


**Capabilities:**

Respond to customer greetings and inquiries.
Add items to a customer's virtual shopping cart.
Provide a running total of the items in the cart.

**Format for adding items:**

`ADD: [item name], [price]`


**Example:**

Customer: "Hello, I'd like to buy a coffee and a sandwich."

Agent: "Hello! Sure, I can help with that. What kind of
 sandwich would you like?"

Customer: "A turkey sandwich."

Agent: "Okay. I'm adding those to your cart. One coffee and one
  turkey sandwich."

Agent: `ADD: Coffee, 3.50`
Agent: `ADD: Turkey Sandwich, 6.00`

Agent: "Your current total is $9.50."
```

The agent's role and goals are defined by the text in black; its capabilities and tools, by the text in red; finally, a one-shot example is provided by the text in blue.

Source: Authors (2026)

important past information and reduced accuracy, if the relevance of the context is not optimized. To address this problem, many frameworks (Wang *et al.*, 2024a; Hu *et al.*, 2023; Shinn *et al.*, 2023) equip agents with long-term memory modules for storing useful information that is not immediately relevant;

c) **planning:** How agents plan their actions is also a crucial aspect. A rational agent should be able to plan how to approach a task beforehand in order to avoid the nega-

tive consequences of thoughtless actions. Since the causal thought process required for planning is closely related to that used in complex reasoning, their approaches often overlap. CoT prompting (Wei *et al.*, 2022) and ensemble techniques (Wang *et al.*, 2023) are commonly used, and frameworks often incorporate task-specific prompting to improve planning. For example, the planning process can be framed as a dialogue between a dummy user and the agent (Ichter *et al.*, 2022). Alternatively, think-act cycles can be employed, where the model is prompted to reflect on its plans before taking any action (Yao *et al.*, 2023; Shinn *et al.*, 2023; Prasad *et al.*, 2024).

Despite the possible association of increased complexity in agent architectures with improved performance, some recent studies have challenged this notion. In (Verma; Bhambri; Kambhampati, 2024), the authors argue that the high scores achieved by the popular ReAct framework (Yao *et al.*, 2023) are primarily due to the similarity between the provided few-shot examples and the task itself, rather than the framework's complex reasoning process. Furthermore, the Agentless framework (Xia *et al.*, 2024) did away with an agentic setup altogether in favor of a simpler LLM-based pipeline. This pipeline enabled it to achieve the open-source SOTA across different variations of the SWE-bench (Jimenez *et al.*, 2024) benchmark, while using fewer tokens. These studies suggest that, when cost is a primary consideration, simpler approaches to agentic tasks may be preferable. On the other hand, methods that employ extensive searching—such as Monte-Carlo tree search (MCTS)—show great promise at achieving high scores, at the cost of significantly higher token counts (Antoniades *et al.*, 2025; Chi *et al.*, 2024; Jiang *et al.*, 2025). These capabilities of LLM agents, particularly in planning and memory, have made automating complex tasks, such as data science, a more common topic of research.

2.3 Automated Data Science

Early work on generative models applied to data science focused on code generation and correction through prompting (Lai *et al.*, 2023; Yin *et al.*, 2023; Chandel *et al.*, 2022). With code-tuned LLMs such as DeepSeek-Coder (Zhu *et al.*, 2024) and Qwen-Coder (Hui *et al.*, 2024) becoming increasingly cost-effective, these approaches serve as valuable tools for reducing the time invested in trivial tasks. However, human expertise remains essential, as the user must still guide the analysis even with LLMs' assistance.

For more general automation, one can look towards the field of automatic machine learning (AutoML) (Hutter; Kotthoff; Vanschoren, 2019). The goal of AutoML is to automatically search for the best machine learning setup to process a given dataset. As with most search-related problems, its application can often be resource-intensive and susceptible to getting trapped in local minima.

To address these shortcomings, recent works have proposed using LLM agents to more effectively guide the search process. Agents such as DS-Agent (Guo *et al.*, 2024) and MLCopilot (Zhang *et al.*, 2024) make clever use of past experiments as examples to enhance their LLM core's ability to generate solutions. Data Interpreter (Hong *et al.*, 2025), on the other hand, creates intricate graph-based workflows to decompose tasks into more manageable chunks. AIDE (Jiang *et al.*, 2025) and SELA (Chi *et al.*, 2024) use LLMs to guide the search for optimal configurations; SELA demonstrates the effectiveness of a well-guided MCTS, while AIDE shows how LLMs can be used to generate new solutions or improve and debug previous attempts.

In terms of assessment, although several benchmarks for evaluating agentic MLE have been recently published (Chan *et al.*, 2025; Hu *et al.*, 2024; Huang *et al.*, 2024a), there is still no centralized metric for evaluating ML agents, with a significant number of studies reporting comparisons across disjointed collections of Kaggle competitions (Hong *et al.*, 2025; Guo *et al.*, 2024; Li *et al.*, 2025; Grosnit *et al.*, 2024; Chan *et al.*, 2025). Additionally, due to the ever expanding scope of LLMs' training data and performance, most benchmarks are likely contaminated or saturated within months of their publication (Jacovi *et al.*, 2023).

More holistic approaches to automated data science and analysis —encompassing data collection, data cleaning, exploratory data analysis and effective data presentation— are less common, likely due to the complexity and subjectivity required to effectively evaluate them. DATA INTERPRETER (Hong *et al.*, 2025)—which we build upon in this work— demonstrates exploratory analysis on its planning graphs; however, its evaluation and reporting are limited. Data Formulator 2 (Wang *et al.*, 2025) describes a LLM-based tool for data visualization, but one that is designed for user prompts rather than end-to-end analysis. Agent K v1.0 (Grosnit *et al.*, 2024) is an end-to-end autoML agent reported as capable of gathering data, processing data and evaluating statistical models; its evaluations, however, have been restricted to interacting with the official Kaggle application programming interface (API).

2.4 LLMs on Portuguese Data

As discussed in Section 1.5, studies directly assessing LLMs capabilities when dealing with data in Portuguese are less common than expected.

Considering smaller models, the BERTimbau model (Souza; Nogueira; Lotufo, 2020) has been a popular choice in Portuguese NLP across various classification tasks. However, following the shift towards larger—server-sized— generative models, Portuguese-specific approaches have been unable to catch up to their multi-lingual counterparts. Open-source models like Bode (Garcia *et al.*, 2024) and GlórIA (Lopes; Magalhães; Semedo, 2024) have failed to produce consistent SOTA results, even when evaluated against multi-language models of similar size. The Sabiá (Almeida *et al.*, 2024; Abonizio *et al.*, 2025) family of closed-source models, intended as a cost-effective, Portuguese-focused, alternative to GPT-4 and Claude, also exhibits inconsistent performance and its potential price advantage is likely to be negated by newer models like DeepSeek-V3, Qwen2.5 and Gemini 2.0, which rivals GPT-4, at a much lower price.

Although LLMs have been independently evaluated on various Portuguese-related tasks, a deeper analysis of their agentic capabilities as a lusophone data analyst remains to be thoroughly investigated. Studies evaluating the ability of recent models to answer Brazilian exam questions are common (Nunes *et al.*, 2023; Mendonça, 2024; Pires *et al.*, 2023) and the performance gap between English and Portuguese on these tasks appears to be relatively small for open-source models (Santos; Campelo, 2023). Additionally, GPT-4 was employed as an automatic annotator on the Quati dataset (Bueno *et al.*, 2024), with a reasonable agreement with human annotators.

Finally, concerning open-ended automated data science in Portuguese, we found only one recent paper that evaluated the capabilities of LLMs (Miranda; Campelo, 2024). In it, the authors tested ChatGPT's out-of-the-box data analysis capabilities using Portuguese prompts and data. Yet, the authors did not provide a comparison between languages and the out-of-the-box solution presented differs significantly from the cutting-edge techniques elaborated on in the previous sections. This further underscores the need for a language-specific validation of the newer advancements.

It is clear that there has been substantial progress in large language models, including their expanding capabilities, refined complex reasoning methods, evolving agent architectures,

and growing application in automated data science. However, while some relevant studies exist, including evaluations on Portuguese datasets and Portuguese-focused models, a comprehensive exploration of LLM agents as lusophone data scientists is yet to be presented.

3 METHODOLOGY

The following sections begin with a description of the process we employed to construct a new lusophone DS evaluation dataset. Next, we expand on the DATAEXPLAINER framework and outline our agentic setup used for evaluating automated MLE. Finally, we conclude by elaborating on the evaluation process, including how agents were scored and how their language consistence was assessed.

3.1 The Evaluation Dataset

As detailed in Section 2.4, evaluation sets for lusophone automated data science are currently lacking in the literature and the available datasets in different languages are increasingly prone to contamination (Jacovi *et al.*, 2023). To address this gap, we created a new Brazilian Portuguese evaluation set consisting of translated machine learning competitions, sourced from the Kaggle website.[1]

3.1.1 Data Gathering

Kaggle was chosen due to its popularity in the literature and its crowd-sourced format, which makes it a solid source of high-quality, varied, DS problems. Although the available competitions are numerous, to ensure the quality of the data and usefulness within our computing budget, we compiled competitions based on three strict requirements:

a) **Recency:** to avoid data contamination, competitions must be published from mid-2024 onwards. Contamination occurs when a LLM has already seen a problem's solution during training. Since there is yet to be a reliable way of determining contaminated data, the safest option is to avoid information published before most models' knowledge cutoff date (Jacovi *et al.*, 2023).

b) **Complexity:** Competitions should primarily involve tabular data. We avoided image, audio, and complex text processing, as dataset complexity directly affects computational costs. For large-scale datasets and complex tasks, even the most accurate

[1] https://www.kaggle.com

frameworks and models may still require millions of tokens and dozens of hours to achieve significant results (Chan *et al.*, 2025).

c) **Variety:** The selected competitions should avoid overlapping or closely related problem domains. We favored variety in both goals and data types, including datasets that contain discrete variables, continuous variables, and categorical data, as well as different machine learning objectives.

The final collection consists of seven competitions on different subjects; all of them published on or after July 1st, 2024. Out of the seven tasks, three are binary classifications, three are regression tasks, and one is a multi-class classification task. Information about the collected competitions is provided in Frame 3.1; competition titles and descriptions are provided in Appendix A.

Frame 3.1 – Information about the gathered competitions

Codename	**Start Date**	**Objective**	**Metric**	**Data Quality**	**Rows in Train Set**
s4e6	Jun. 1, 2024	Multi class classification	Accuracy	Clean	76,518
s4e8	Aug. 1, 2024	Binary classification	MCC[1]	Artifacts and Missing Data	3,116,945
s4e9	Sep. 1, 2024	Regression	RMSE[2]	Missing Data	188,533
s4e10	Oct. 1, 2024	Binary classification	AUC[3]	Clean	58,645
s4e11	Nov. 1, 2024	Binary classification	Accuracy	Artifacts and Missing Data	140,700
s4e12	Dec. 1, 2024	Regression	RMSLE[4]	Missing Data	1,200,000
s5e2	Feb. 1, 2025	Regression	RMSE	Missing Data	300,000

[1] Matthews correlation coefficient
[2] Root mean squared error
[3] Area under the curve
[4] Root mean squared logarithmic error

Source: Authors (2026)

3.1.2 The Translation Process

The gathered data was then translated from English to Portuguese using a combination of machine and human translation. After the textual data were extracted from the original descriptions and `.csv` files, it was initially translated by Gemini 2.5 pro[2] (DeepMind, 2025). Subsequently, two human reviewers familiar with data science—a 23-year-old graduate student and a 25-year-old post-graduate student— analyzed and corrected Gemini's output. The reviewers conducted their initial corrections independently, and then combined their work for the final text. Disagreements were addressed through discussion and consensus, with the more experienced reviewer ultimately having final say. All prompts used for the translations are provided in Appendix B.

There are four relevant types of textual information within competitions which were either completely or partially translated:

a) **Competition names:** the names of the competitions serve as valuable context for each problem's goal and approach. Consequently, they were completely translated.

b) **Descriptions:** descriptions involve the explicit "goal" section of each competition, their evaluation metrics, information about the source of data, and submission format. Just like the names, descriptions are fundamental context, so they were completely translated.

c) **Column names:** column names are usually an overall description of the type of data in each column. This makes them invaluable information that often guides data normalization, cleaning, and feature engineering. For that reason, all column names were also fully translated.

d) **Text-encoded categorical values:** these are categorical values which are represented as text. Such as "single" and "married" for marital status. Some of these values, although important, do not have a clear translation and therefore were left untranslated; these exception include, for example, all names other than country names, and labels without a clear lexical value (e.g., "A" through "F", representing loan grades).[3]

[2] As provided by Google Cloud's API on May 15, 2025.

[3] On the other hand, some labels have semantic underpinnings; these were fully translated. A clear example is the labels "e" and "p" for "edible" and "poisonous", respectively, which were translated to "c" and "v" ("comestível" and "venenoso").

By convention, numerical and date fields within tables remained in their original format and not localized.

After the text fields have been translated, they were applied back to the train and test `.csv` files. Since the amount text in each competition depends on the number categorical features in each training set and how these are encoded, the amount of translated data is substantially varied. With 302 and 563 as the mean and standard deviation for the number of translated fields across all competitions. The competition s4e9d stands out with by far the largest amount of text: 1670 fields in total, 1413 more than the second largest, s4e11.

3.2 The Agentic Setup

In defining an agentic setup for our evaluations, we focused on three fundamental characteristics: robustness, token consumption, and interpretability, with the robustness of a setup being the ability to achieve the most optimal performance on a given task. However, while high performance is generally desirable, top results are frequently linked to increased token consumption and reduced interpretability (Jiang *et al.*, 2025; Chi *et al.*, 2024; Hong *et al.*, 2025). For instance, frameworks that involve intensive search methods and retrieval augmented generation (RAG) can often achieve near expert-level metrics, yet the numerous trial-and-error steps commonly involved render the process costly and difficult to follow. Conversely, directly prompting a chatbot offers a lightweight and easily interpretable approach, albeit at the expense of being more prone to error (Miranda; Campelo, 2024).

3.3 DATAEXPLAINER

Aiming for a compromise between performance, replicability, and token usage, we developed a new agentic framework, which we named DATAEXPLAINER. Our framework is based on a previous work by (Hong *et al.*, 2025) named DATA INTERPRETER, similarly using a Jupyter notebook for execution, outputting a `ipynb` file and employing topological orderings for its action plan progression. However, DATAEXPLAINER also includes some crucial changes and additions compared to DATA INTERPRETER which will be expanded upon on the next subsections. The DATAEXPLAINER framework served as the basis for all evaluated agents; unless stated otherwise, they differ only in their LLM cores.

3.3.1 Overview

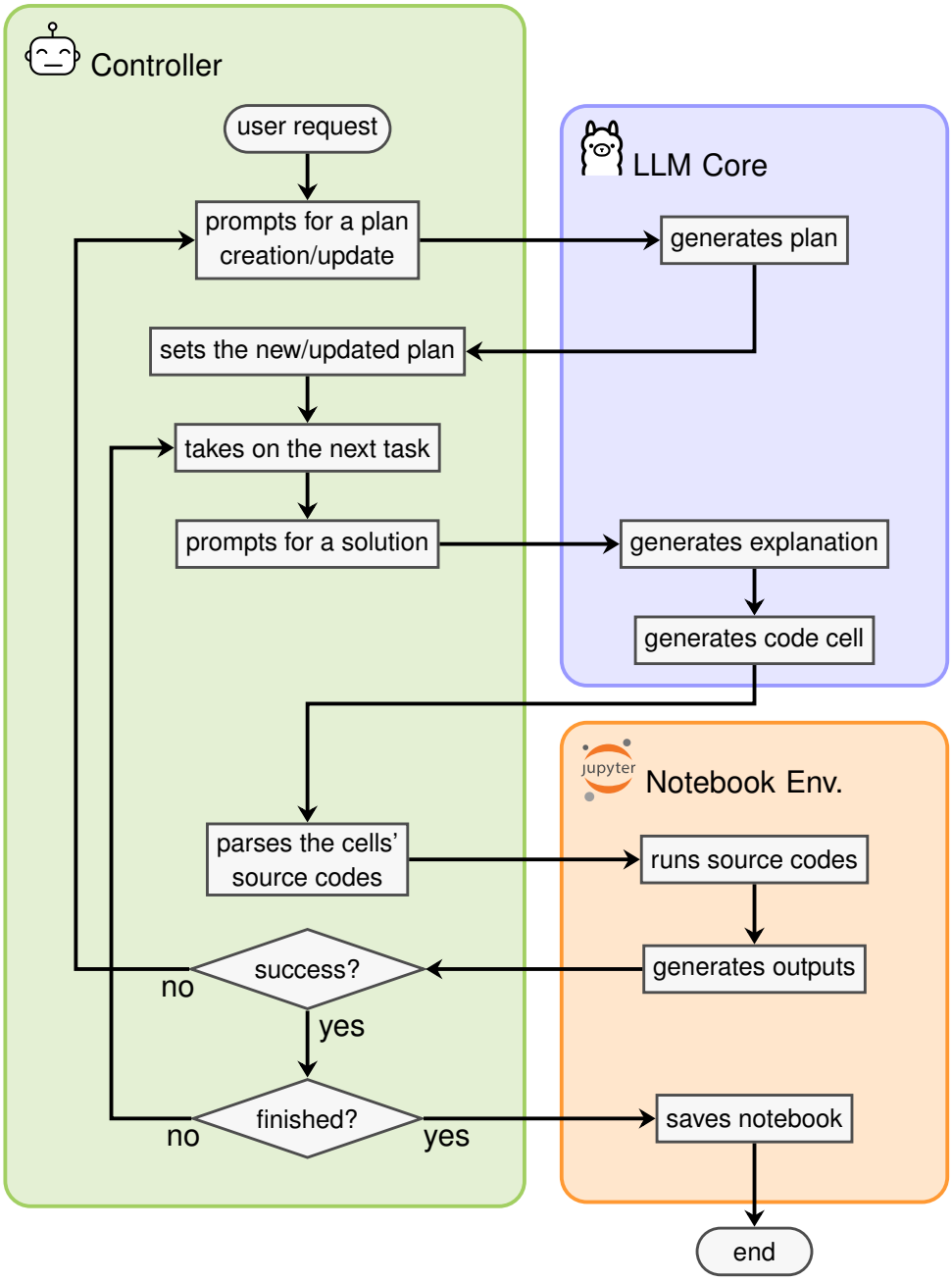
DATAEXPLAINER's pipeline differs from the original DATA INTERPRETER mainly by generating explanatory steps in addition to python code; the result is a solution notebook that is closer in style to a didactic, step-by-step analysis similar to those of human professionals. Figure 3.1 depicts DATAEXPLAINER's overall process. When addressing a user's request (such as solving machine learning problems or conducting exploratory analyses), the controller prompts the LLM core for a plan. The LLM provides a plan, and with it, the controller continuously undertakes each task, querying the LLM for the next notebook cells necessary to complete the current task. The LLM core first provides an explanation for its process, summarizing its previous executions and providing an outline of its proposed solution. Next, it generates a solution to the task using Python code. With the LLM output, the controller parses the received information and executes it in a Jupyter notebook environment—the explanation as a markdown cell and the solution as a code cell. If the execution is unsuccessful, the controller revises its plan and tries again. If it is successful, the controller checks for unfinished tasks and continues to work on them. When all tasks are finished, the notebook file is saved, and the process is complete.

3.3.2 The Action Plan

The action plan is a guide to the LLM core, narrowing the scope of its generations and setting an path towards a holistic solution. While DATA INTERPRETER uses a complex action plan, including the code solutions and outputs, DATAEXPLAINER uses a minimal approach to planning focusing exclusively on the sub-tasks' descriptions, types and order. To accommodate the elements removed from the action plan, we created a notebook state, which we will expand upon in Sections 3.3.3 and 3.3.4.

The user's request is divided into n sub-tasks, which contain a description, a type (data loading, EDA, model training and so forth), and a dependency list, which is used to define the plan's overall sequence through a topological sort. In case the agent struggles to complete a task, it can alter its plan mid-processing, with the addition of new tasks or removal/alteration of unfinished tasks. This allows the agent to adapt to previously unseen developments during its execution without losing track of its main trajectory.

Figure 3.1 – The overall process of DATAEXPLAINER



Source: Authors (2026)

Description for the Figure: A flowchart illustrating the workflow of the DATAEXPLAINER system. Arrows indicate the flow of information and control between the controller (in a green box), the LLM core (in a purple box), and the Jupyter notebook environment. (in an orange box). A deeper description of the flow of information is provided in Section 3.3.1.

3.3.3 Interaction with the LLM Core

When compared to the original framework, the interaction to the LLM core was largely overhauled, with a stronger focus on the Jupyter Notebook format and on the text prompt, instead of the action plan.

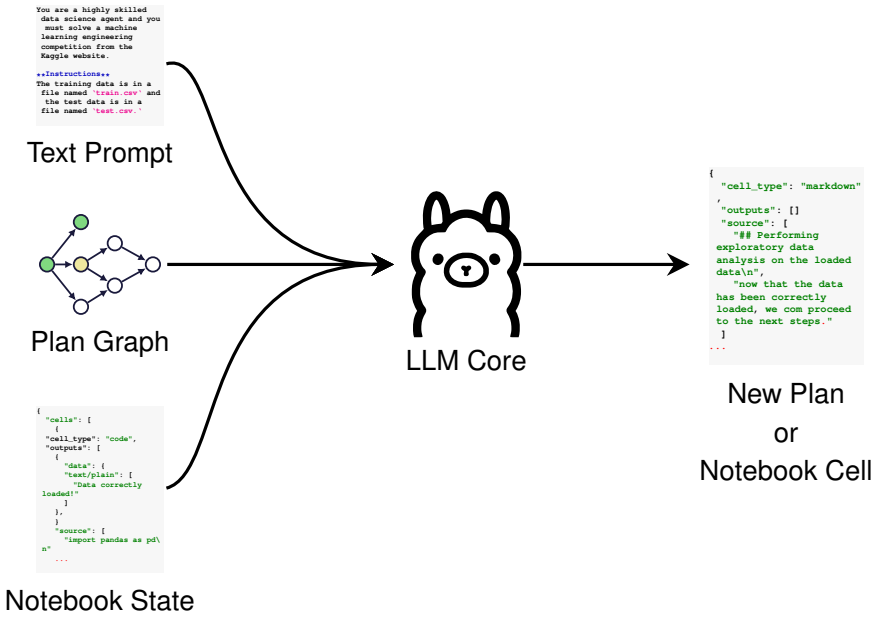
There are, in general, two types of interactions between the controller and the LLM core: the creation/update of its action plan and the creation of a new notebook cell. Figure 3.2 is an example of how the LLM core is prompted. For every LLM call, the controller generates a prompt composed of three different types of information: the text prompts, which include the system prompt, the user request and a task-specific guidance;[4] the plan graph, which is the graph representation of the current plan (in a JSON format), with completed, current and future tasks properly flagged; and the notebook state, which is the current state of the jupyter notebook being developed, in a simplified—i.e., more token-efficient— `ipynb` format. The LLM core, then, generates a JSON object containing either the generated plan or notebook cell. Both the notebook state and the generated cell use the same `ipynb` structure. We deliberately chose this approach in order to create a context similar to the DS solutions available on the web, which mostly uses `ipynb` notebook files. We hypothesize this closer context should improve the overall metrics.

3.3.4 Interaction with the Jupyter Environment

The controller interacts with a Jupyter notebook client to run the cells and to keep an active python kernel. Every new cell created by the LLM core and parsed by the controller is executed in the environment, which, in turn, generates an outputs list. Outputs can be text streams, images or error messages; each type is adjusted in specific ways before being re-fed into the agent's memory. For the text streams, the agent ignores repetitive warnings—which can be taxing on the limited context— removes text decorations (such as color tags) and truncate string lengths to a predefined limit. For the images, the agent will either encode into the correct format or ignore altogether, depending on the agent's visual processing setup. Finally, error

[4] Task guidances are a markdown list of important aspects to consider for each task type; for example, a "model evaluation" task might have the guidance "remember to always evaluate models in a holdout set, never in the train set"

Figure 3.2 – Overview of LLM calls



Source: authors (2026)

Description for the Figure: A diagram illustrates that a "Plan Graph" (depicted as a graph with the completed tasks in green and the current task in yellow), "Notebook State" (presented in JSON code), and "Text Prompt" (in markdown text) are inputted into the "LLM Core" to generate the "New Plan or Notebook Cell."

messages have their traceback limited to the first and last two iterations, as to reduce the context length and focus on the most relevant information.

3.4 The Evaluation Setup

Evaluating automated machine learning engineering through open competitions is a well-established and directly measurable process. All competitions available on Kaggle feature a clear metric, submission process, and scoreboard, which can be easily drawn upon to generate a comprehensive assessment. With that in mind, the evaluation process was centered around agents being prompted to solve a specific competition, generating a CSV submission file and its predictions being scored using the Kaggle API. Additionally, to ensure the agents were conducting analyses on the requested language, we also analyzed generated notebooks, containing their process, to check how consistent with the original prompt's language is the generated text.

All agents were prompted using two different markdown templates, one for each language, which informs fundamental information about its task, such as the competition's goal, submission format and non-specific tips. Both templates are provided in Appendix C.

As LLMs often operate on a non-deterministic principle, 4 different samples were collected from each agent-competition-language tuple. For each sample, the agents had a 60 minutes time limit for completing the Jupyter notebook execution. As models may often underestimate processing times, they were granted a single additional try after the first timeout. The three main metrics involved in our evaluations, success rate, corrected score and language consistency, are expanded upon in the next sections.

3.4.1 Success Rate

The success rate metric quantifies an agent's compliance with the given constraints and submission format when generating a solution. To evaluate the success rate, we define a function $success_c$ that considers a solution, comprised of a Jupyter notebook and submission file, and outputs either a 1 (sucess) or a 0 (failure), depending if any of the following violations/failures occur:

a) The agent attempted to directly access the web in order to find a solution;
b) The agent attempted to directly access system files other than the ones explicitly provided;
c) The agent attempted to manually fill the submission file;
d) The agent failed to generate a submission file within the established time limit;
e) The submission file was unable to be submitted to the API;
f) The submission file was successfully submitted, but the system failed to assigned it a competition score;

These failure cases were identified using a mix of heuristics and human decision: firstly, as a heuristic, all notebooks were scanned for evidences of offending code[5] and matches were flagged as possible failures. Next, a human analyzed each of the selected cases and set a final

[5] e.g, the use of the "requests" package for accessing the web or the use of file paths other than "train.csv" and "test.csv".

verdict. Formally, the success rate (SR) of an agent a when given a competition c is:

$$\mathrm{SR}_a(c) = \frac{1}{n}\sum_{i=1}^{n} success_c(S_{a,i}(c)); \tag{3.1}$$

With n being the number of samples collected and $S_{a,i}(c)$ being the ith sampled solution generated by agent a when given a competition c.

3.4.2 Competition Score

Evaluating agents by score, although straightforward when considering competitions in isolation, becomes significantly more complex when aggregating various ML problems toward a single comprehensive metric. Classification score metrics are typically in the $[0,1]$ range and are directly correlated with performance, whereas regression metrics, in contrast, usually fall in the $[0,\infty)$ range and are inversely correlated with performance. This implies that any assemblage of these metrics involving direct sums would severely misrepresent the performance.

To address this issue, some recent works (Hong *et al.*, 2025; Li *et al.*, 2025) have employed a normalized regression score in the form of $\mathrm{NPS}(s) = \frac{1}{1+s}$, with s being the raw Kaggle regression score. Although this approach corrects for range and correlation between classification and regression metrics, it further exacerbates another problem when aggregating scores: the variability across competitions. While some MLE challenges are trivial, clustering leaderboard scores around the average, others require a deep understanding of the data, resulting in a more spread-out distribution. When averaging the raw scores, this difference results in more importance being given to competitions whose leaderboards have a larger standard deviation. For instance, consider two competitions, c_1 and c_2, whose leaderboard submissions are normally distributed, both with a mean accuracy score of 0.5 and with standard deviations of 0.01 and 0.1, respectively. If an agent generates a submission with score 0.52 for c_1 and a submission with score 0.479 for c_2, its average score of 0.499 would inaccurately place it below the global mean, despite ranking in the top 3% for c_1 and close to average for c_2.

Thus, to effectively measure the agent's scores across various competitions, we adjusted scores based on the placement of each submission on Kaggle's official competition public leaderboards. This form of normalization corrects for both differences in metrics and in vari-

ation, as it is anchored to direct comparisons to previous available, mostly human, solutions. We named this score PCS, for placement-corrected score. Ranging from 0 to 1, it represent how close to the top of the leaderboard a score is, with larger values being closer to the top. Formally, the PCS of an agent a when given a competition c is defined as:

$$\mathrm{PCS}_a(c) = \begin{cases} \frac{1}{n}\sum_{i=1}^{n} P(X \leq score_c(S_{a,i}(c)) \mid X \sim \mathrm{L}_c), & \text{for classification tasks} \\ \frac{1}{n}\sum_{i=1}^{n} P(score_c(S_{a,i}(c)) \leq X \mid X \sim \mathrm{L}_c), & \text{for regression tasks} \end{cases} \tag{3.2}$$

Where n is the number of samples collected, P is the probability function, $score_c(S_{a,i}(c))$ is the raw Kaggle score for solution $S_{a,i}(c)$ to competition c, L_c is the public leaderboard for c, and X is a random variable distributed according to L_c.

3.4.3 Language Consistency

Lastly, to quantify language consistency, per notebook, we first export all text lines in markdown cells and code comments, and then detect their language using the *langdetect*[6] python package. Finally, we average the scores of each of the agents over all their generated text lines to find their overall language consistency score. The consistency of an agent a for language L formally being:

$$Consistency_L(a) = \frac{\sum_{t \in T_a} \mathrm{lang}_L(t)|t|}{\sum_{t \in T_a} |t|} \tag{3.3a}$$

$$\mathrm{lang}_L(t) = \begin{cases} 1, & \text{if } langdetect_L(t) \geq threshold \\ 0, & \text{if } langdetect_L(t) < threshold \end{cases} \tag{3.3b}$$

Where T_a is the set of all text lines (markdown lines and code comments) in notebooks generated by an agent a, $|t|$ is the length of the line of text t in number of characters, $langdetect_L(t)$ is the probability that t is in the language L and $threshold$ is the minimum confidence required.

[6] https://pypi.org/project/langdetect/

4 RESULTS

In the next sections, we start by exploring the characteristics of the final developed dataset, such as the amount of fields translated, types of data in each subset, and agreement between translations. Next, we present an in-depth analysis of the agents' performances in English and Portuguese, as well as the differences in performance between them. We also analyze the correlations between data recency and the agent's scores, and present our language consistency findings. Finally, we end by analyzing the proposed DATAEXPLAINER framework in comparison to the framework upon which it is based.

4.1 The Resulting Translated Dataset

Table 4.1 shows the distribution of columns and translated fields in the dataset. Overall, 2,119 fields of data (descriptions, column names, textual categorical values), were translated. The distribution of amount of translations between competitions was heavily skewed; five competitions fall within the 28 to 55 range, while s4e11 and s4e9 account for 257 and 1,670 fields respectively. This is mostly due to expressive amount of unique categorical values in these specific competitions and to the other competitions having a larger number of categorical values encoded as ordinal labels rather than text. The types of fields in each competition was also unbalanced, with a majority being categorical (59.4%), followed by numerical (39.1%) and finally, fairly underrepresented, date fields, which account only for 1.5% of columns. While skewness in column types is to be expected any collection of datasets not specifically designed for balance in this specific aspect, the low quantity of date fields is still significant and warrant future investigations.

As Table 4.2 shows, agreement between translations and revisions are fairly high, with the largest difference being less than 5% overall and less than 4% when compared to the final dataset. This indicates that most of the automated translation was mostly effective at adapting information from English to Portuguese, although corrections were still necessary. The most significant mistake made by Gemini 2.5 was not translating the target columns —despite being explicitly prompted to, as can be seen in Appendix B— and providing less-common translations to obscure terms, like those relating to the anatomy of mushrooms in competition s4e8. Dis-

Table 4.1 – Competition data and translation competitions

Codename	Number of Columns (%)				Translated Fields
	Numerical	Categorical	Date	Total	
s4e6	22 (57.9)	16 (42.1)	0	38	45
s4e8	3 (13.6)	19 (86.4)	0	22	28
s4e9	2 (15.4)	10 (76.9)	1 (7.7)	13	1,670
s4e10	7 (53.8)	6 (46.2)	0	13	29
s4e11	8 (40.0)	12 (60.0)	0	20	257
s4e12	9 (42.9)	11 (52.3)	1* (4.8)	21	55
s5e2	3 (27.2)	8 (72.7)	0	11	35
Total	**54 (39.1)**	**82 (59.4)**	**2 (1.5)**	**138**	**2,119**

* Only the value for the year, not an entire date.

Source: Authors (2026)

agreements between reviewers mostly centered on capitalization and space characters. Overall, the high agreement across the board is evidence of the robustness of the process.

4.2 Agentic Evaluations

For the agentic evaluations, we tested four different LLM cores with distinct characteristics: Gemini 2.0 Flash (DeepMind, 2025), a proprietary model; GPT-OSS 20B (Agarwal *et al.*, 2025)[1], an open-source thinking model; Qwen3 coder 30B[2] (Yang *et al.*, 2025), an open-source code-tuned model; and Gemma 3 27B QAT[3] (Team *et al.*, 2025), an open-source generalist model. The Gemini model was provided by Google's cloud API service and its context window was limited to 128K tokens. The other models were run locally using the Ollama server version 14. To reduce the overall memory footprint, the locally run models were quantized to 4 bits

[1] https://ollama.com/library/gpt-oss:20b
[2] https://ollama.com/library/qwen3-coder:30b
[3] https://ollama.com/library/gemma3:27b-it-qat

Table 4.2 – Agreement Between Translations

Translation 1	**Translation 2**	**Fields Agreed Upon (%)***
Gemini 2.5 Pro	Reviewer 1	2,077 (98.0)
Gemini 2.5 Pro	Reviewer 2	2,027 (95.7)
Reviewer 1	Reviewer 2	2,041 (96.3)
Final Version	Gemini 2.5 Pro	2045 (96.5)
	Reviewer 1	2,061 (97.3)
	Reviewer 2	2,082 (98.3)

* Out of a total of 2,119.

Source: Authors (2026)

and had a context window of 32K tokens, with GPT-OSS being provided an extra 32K as a "thinking budget".

4.2.1 Agents' Performances

Table 4.3 shows the success rate for all four models and seven competitions. All failures had only two causes: not generating a submission file within the time limit and incorrect file formats. There were no instances of agents accessing the web or files outside the delimited workspace for solutions. Some models, after failed attempts to train an inference model, filled the submissions with the target's most common class or average value, for classification and regression tasks respectively. We considered those submissions valid, as the resulting model, although crude, can also be considered an algorithm for inference.

All models were more successful at completing the tasks in English than in Portuguese, although to different extents. GPT-OSS was the most successful agent, being the only one to achieve 100% success for English and the highest rate for Portuguese, although almost 15% less than its results for English. Gemini 2.0 and Gemma3 had success rates reasonably similar for both languages, however, while Gemma3 achieved the lowest completion rates, Gemini was reasonably effective for both languages. Qwen3, on the other hand, had an average success rate

Table 4.3 – Success Rate between Languages per Model

Model	Lang.	Success Rate %							
		s4e6	**s4e8**	**s4e9**	**s4e10**	**s4e11**	**s4e12**	**s5e2**	**Mean**
Gemini 2.0 Flash	en	100	50	50	100	100	100	75	82.14
	pt	100	50	50	100	25	75	100	71.43
GPT-OSS	en	100	100	100	100	100	100	100	100
	pt	100	100	100	100	100	50	50	85.71
Qwen3-coder	en	50	75	100	100	100	100	100	89.29
	pt	100	75	0	100	50	75	50	64.29
Gemma3	en	75	0	50	100	0	100	75	57.14
	pt	25	50	0	75	75	100	50	53.57
Mean	en	81.25	56.25	75	100	75	100	87.5	82.14
	pt	81.25	68.75	37.5	93.75	62.5	75	62.5	68.75

Source: Authors (2026)

of almost 90% for English and only 64% for Portuguese, indicating it to be less multilingual than its counterparts.

To consider the quality of the agents' submissions, we corrected each of the raw submission scores by applying the equation presented in Section 3.4.2 and considering the public leader-board score distribution of each competition on December 12, 2025. Consequentially, the resulting score represent the submission's placement on its respective leader-board at the aforementioned date. Table 4.4 presents the mean corrected score over all four samples. GPT-OSS achieved the best scores in most competitions and had a negligible average difference between English and Portuguese. Gemma3 performed poorly in both languages, rarely achieving scores above 20%. Qwen3, although more capable than Gemma3, still falls behind GPT-OSS for most competitions, with s5e2 being the sole exception. Gemini 2.0 had an English score marginally lower than Qwen, but 4% higher for Portuguese and a small average gap between languages. Once again showing a high adaptability.

Table 4.4 – Mean Scores between Languages per Model

Model	Lang.	Mean Corrected Score %							
		s4e6	**s4e8**	**s4e9**	**s4e10**	**s4e11**	**s4e12**	**s5e2**	**Mean**
Gemini 2.0 Flash	en	37.42	34.06	6.88	48.76	38.47	25.45	17.62	29.81
	pt	50.19	11.5	6.45	35.64	12.33	42.89	42.25	28.75
GPT-OSS	en	48.85	44.87	25.64	66.79	35.25	16.19	36.38	39.14
	pt	57.17	47.22	26.42	54.29	51.28	18.13	18.51	39.0
Qwen3-coder	en	33.64	41.32	26.53	37.68	4.21	14.37	57.56	30.76
	pt	54.57	12.46	0.0	50.31	9.62	8.93	32.98	24.12
Gemma3	en	25.29	2.35	3.72	56.43	0.0	11.37	2.38	14.51
	pt	5.74	8.59	0.0	37.11	15.3	3.3	1.52	10.22
Mean	en	36.3	30.65	15.69	52.42	19.48	16.85	28.48	28.55
	pt	41.92	19.94	8.22	44.34	22.13	18.31	23.81	25.52

Source: Authors (2026)

Table 4.5 – Best Scores between Languages per Model

Model	Lang.	Best Corrected Score %							
		s4e6	**s4e8**	**s4e9**	**s4e10**	**s4e11**	**s4e12**	**s5e2**	**Mean**
Gemini 2.0 Flash	en	52.88	68.28	13.82	73.13	46.48	48.2	47.95	50.11
	pt	73.28	22.24	16.17	44.08	49.31	60.79	50.93	45.26
GPT-OSS	en	66.41	74.34	39.83	75.28	43.58	21.66	64.39	55.07
	pt	59.94	64.19	36.15	59.47	73.99	40.76	64.51	57.0
Qwen3-coder	en	71.65	68.28	42.67	61.67	4.21	28.39	63.98	48.69
	pt	67.37	21.29	0.0	50.89	19.54	16.47	42.8	31.19
Gemma3	en	40.21	2.35	7.43	58.54	0.0	14.17	3.06	17.97
	pt	20.62	14.89	0.0	54.47	20.39	4.14	3.03	16.79
Mean	en	57.79	53.31	25.94	67.15	23.57	28.1	44.85	42.96
	pt	55.3	30.65	13.08	52.23	40.81	30.54	40.32	37.56

Source: Authors (2026)

However, since Kaggle's public leader-boards only consider the best submission for each user, the agents' mean scores under-represent their performances. Table 4.5 displays each agent's best score across all four samples and is a more accurate measurement of the agents' capabilities when compared to the other competitors. With this adjustment, we can see that GPT-OSS surpassed the average user in most competitions when using either language, with Portuguese now displaying an edge over English. Gemini 2.0 achieved above average performance in English, but still fell 5 points short for Portuguese. Qwen3 placed just below average for English; although performing above the mean on four out of the seven competitions, its scores on the other three—specially on s4e11— hindered its overall evaluation. Gemma3, even while performing worse than all other agents on average, still achieved above-average scores for both languages on competition s4e10.

4.2.1.1 Significance Analysis

Table 4.6 shows a pairwise Wilcoxon signed-rank test between rank-matched English and Portuguese scores for the same competition on each task type. To control for multiple comparisons, we adjusted the p-values using the Benjamini-Hochberg procedure. We can see that, despite an apparent bias towards English, the overall difference in score is not statistically significant ($p < 0.05$) for any of the agents. The data indicates a non-significant trend towards lower values and overall higher score disparity between languages for regression. Determining if there is a consistent difference between task types across all models would require a larger competition sample size.

Employing a Friedman test to formally evaluate the highest scoring LLM agent for Portuguese results in a significant p-value of 4.05×10^{-5}. Its *post-hoc* analysis using the Nemenyi test is shown in Table 4.7. We can see that GPT-OSS was significantly better than Qwen and Gemma, but not Gemini 2.0. The difference between Gemini and Qwen is notably far from the 0.05 threshold. The same analysis for English is provided in Appendix D.

4.2.1.2 Correlation with Metadata

To analyze possible biases in the treatment of the data, we attempted to draw correlations between the submission scores and its competition's metadata. We focused on elements that could best indicate contamination or added bias: the publication date and the number of fields translated. Figure 4.1 shows a linear correlation for each of the models, evaluating the relation between the amount of months since the competition's publication and each agent's scores. Although the relation is mostly negative, the low R^2 values indicate data recency is not a strong determinant of performance. This further confirms that the data in unlikely to be contaminated for these models. Figure 4.2 correlates the gap in scores between English and Portuguese rank-matched submissions and the amount of translated text fields in the competition it solves. The negligible R^2 values indicate that the amount of text translated is unlikely to be a factor affecting the performance difference between languages.

Table 4.6 – Difference in performance between English and Portuguese based on task type

Task Type	**Model**	**Mean**		***p*-value**	**Adjusted *p*-value**
		En.	**Pt.**		
Classification	Gemini 2.0 Flash	39.67	27.41	–	–
	GPT-OSS	48.94	52.49		
	Qwen3-coder	29.21	31.74		
	Gemma3	21.02	16.69		
	All models	34.71	32.08	0.5	0.625
Regression	Gemini 2.0 Flash	16.64	30.53	–	–
	GPT-OSS	26.07	21.02		
	Qwen3-coder	32.82	13.96		
	Gemma3	5.82	1.60		
	All models	20.34	16.78	0.625	0.625
All tasks	Gemini 2.0 Flash	29.81	28.75	–	–
	GPT-OSS	39.14	39.0		
	Qwen3-coder	30.76	24.12		
	Gemma3	14.51	10.22		
	All models	28.55	25.52	0.374	0.625

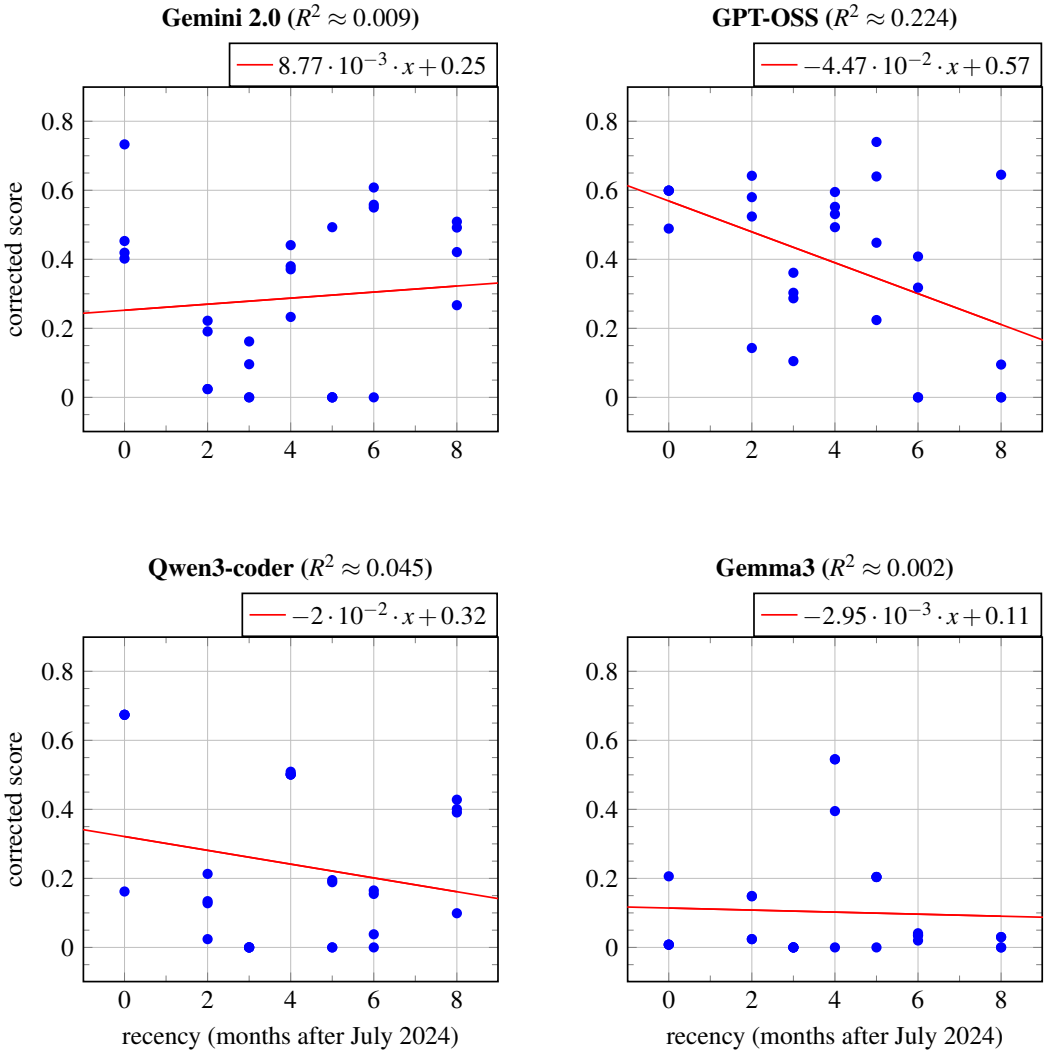
Source: Authors (2026)

Table 4.7 – Nemenyi test between models for Portuguese

	Gemini 2.0 Flash	**GPT-OSS**	**Qwen3-coder**	**Gemma3**
Gemini 2.0 Flash	1.0	0.131	0.907	0.092
GPT-OSS	0.131	1.0	0.042	0.000
Qwen3-coder	0.907	0.042	1.0	0.168
Gemma3	0.092	0.000	0.168	1.0

Source: Authors (2026)

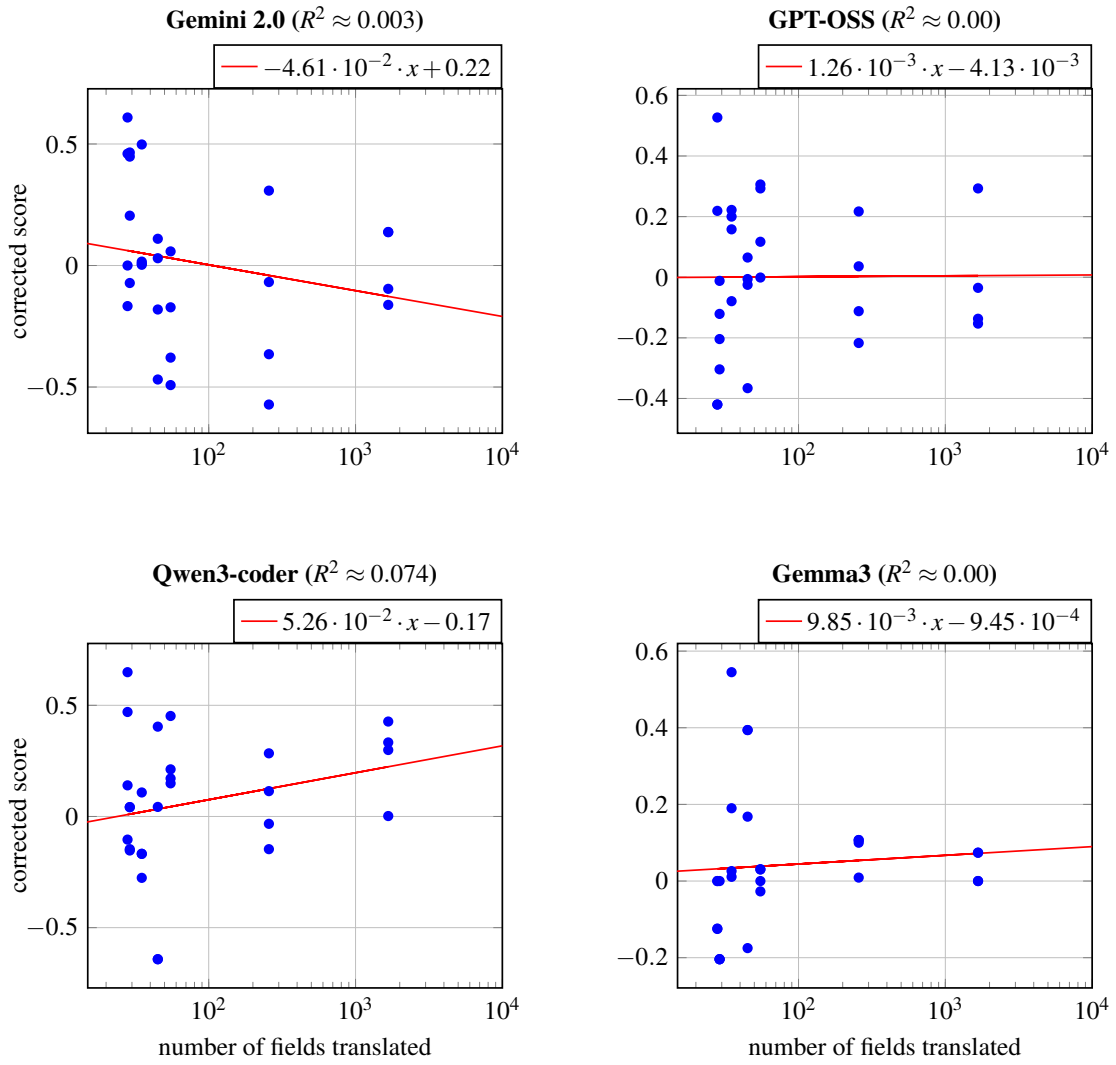
Figure 4.1 – Correlation between competition recency and corrected scores for Portuguese.



Source: authors (2026)

Description for the Figure: Four graphs show linear correlations between recency (months since July 2024) and corrected scores for the studied models.

Figure 4.2 – Correlation between fields translated (log scale) and the score difference between languages



Source: authors (2026)

Description for the Figure: Four graphs show linear correlations between the number of fields translated (log scale) and corrected scores for the studied models.

4.2.2 Language Consistency

We measured the agents' language consistency in both the markdown cells and the commented lines inside code cells. For the markdown cells, we removed all inline code between single backticks and all code blocks between triple backticks that could affect the language detection algorithm. For that same reason, for the code comments, we removed all lines containing four consecutive minus signs, which agents often use for delimitation, and all lines containing underscores, to filter out commented code lines.

Table 4.8 presents the language consistency of the agents' generated notebooks, considering a language detection confidence threshold of 95%. Most models were capable of consistently generating text and code in the prompted language, with nearly all markdown cells and most of the comments in the correct language. Gemma3 is, again, the notable exception. In a few instances, after multiple failed attempts to generate a valid code cell, the Gemma agent completely switched languages in the middle of the solution process; an example of language switching is shown in Appendix E. Specifically for Portuguese, GPT-OSS and Gemini 2.0 are still the best performing, followed by Qwen3, 1.9 percentage points behind. As an example, we provide a complete solution in Portuguese generated by DATAEXPLAINER using GPT-OSS for competition s4e11 in Appendix E.

4.2.3 Evaluating DATAEXPLAINER

To ensure our proposed framework is effective at its task, while being more interpretable by generating markdown cells, we compare DATAEXPLAINER to the original DATA INTERPRETER under the same conditions. The code for DATA INTERPRETER was exactly as provided in the authors' github page at the original paper's publication (Hong *et al.*, 2025), with the sole exception being the changes required to give the agents an additional timeout. Table 4.9 shows a comparison of mean English corrected scores between the frameworks for each LLM and competition. DATAEXPLAINER outperformed the original for every LLM tested, with a major improvement for Qwen3-coder and Gemma3.

Table 4.8 – Language Consistency Scores for English and Portuguese

Model	Lang.	Consistency %		
		markdown	**comments**	**all**
Gemini 2.0 Flash	en	100.0	98.3	99.6
	pt	99.7	98.6	99.4
GPT-OSS	en	100.0	99.1	99.8
	pt	100.0	99.3	99.8
Qwen3-coder	en	100.0	99.5	99.9
	pt	100.0	92.2	97.9
Gemma3	en	100	95.6	99.3
	pt	97.4	80.7	94.2
Mean	en	100.0	98.1	99.7
	pt	99.3	92.7	97.8

Source: Authors (2026)

Table 4.9 – Mean Scores between Frameworks per Model

Model	Framework	Mean Corrected Score %							
		s4e6	**s4e8**	**s4e9**	**s4e10**	**s4e11**	**s4e12**	**s5e2**	**Mean**
Gemini 2.0 Flash	interpreter	61.49	11.38	32.17	13.49	10.05	9.79	22.69	23.01
	explainer	37.42	34.06	6.88	48.76	38.47	25.45	17.62	29.81
GPT-OSS	interpreter	42.18	2.35	32.33	36.96	42.98	3.6	38.67	28.44
	explainer	48.85	44.87	25.64	66.79	35.25	16.19	36.38	39.14
Qwen3-coder	interpreter	0.78	2.35	15.81	0.0	0.0	0.0	3.56	3.21
	explainer	33.64	41.32	26.53	37.68	4.21	14.37	57.56	30.76
Gemma3	interpreter	0.78	2.35	9.35	0.0	44.1	0.0	0.0	8.08
	explainer	25.29	2.35	3.72	56.43	0.0	11.37	2.38	14.51
Mean	interpreter	26.31	4.61	22.41	12.61	24.28	3.35	16.23	15.69
	explainer	36.3	30.65	15.69	52.42	19.48	16.85	28.48	28.55

Source: Authors (2026)

5 DISCUSSIONS AND LIMITATIONS

The following sections present discussions on this work's results and its limitations. We begin by discussing the evaluation set, then comment on the agents evaluations for both languages evaluated and conclude by elaborating on the proposed framework.

5.1 On The Evaluation Set

The competitions gathered, all published before the models' reported knowledge cutoff dates, did not show any signs of contamination when correlated to the measured scores. The translation process had a high agreement rate between reviewers and did not seem to correlate with the score gaps between English and Portuguese. The set required a wide variety of techniques to achieve effective scores, such as imputation, data encoding, feature engineering, and hyperparameter tuning; consequently, the scores it generated had, for the most part, adequate resolution, rarely reaching extreme highs or lows. These findings indicated that the evaluation set effectively measured the agents' performances with minimal added bias. However, a larger sample size is required for a deeper statistical analysis of many of the identified trends, such as differences when isolating task types and data variety within competitions. Additionally, we acknowledge that the data used in this work will become increasingly inadequate as new LLM are launched. As knowledge cutoffs progress, the probability of data contamination will also significantly increase, biasing the results. Fortunately, the process we employed can be replicated and scaled to include larger sample sizes.

5.2 On Success Rate

The agents' success rates bring insights to these agents shortcomings as a whole. Firstly, agents seem well capable of following negative instructions—things it should *not* do— as none have tried to find a solution on the web or in files outside its workspace; the same is true for manually filling submission files. On the other hand, the agents struggled with positive instructions, such as completing tasks under a set time limit and following a submission format, which caused every identified failure. Agents were consistently more prone to these mistakes when dealing with data in Portuguese, which indicates a gap in compliance between languages.

A possible explanation is that the LLMs are attending to the more complex elements, such as error handling, while losing focus on its main goals. For example, an LLM might try more sophisticated forms of data handling, which requires more complex code and causes more errors, and, as a consequence, be more likely to fail. Gemini's success rate and corrected score for Portuguese on competition s4e11 shows evidences of this process. The agent failed to create a valid submission on three out of the four samples it generated; the sole submission, however, achieved a score higher than all the English submissions for the same problem. This indicates the model attempted more complex solutions that produced better results but also were more likely to fail. If a lapse in attention is the culprit, inference-time techniques, like prompting the models to "remember the rules" before generating code might improve the results.

5.3 On The Scores

The competition scores are less consistent than the success rates. Although mean competitions scores give insights into how difficult a competition is on average, isolated scores varied massively between models. For instance, while Qwen3 achieved an average English score of 57.56% for its solutions to competition s5e2, the best out of all models, at competition s4e11 its English score only reach a forth of the average. When we compare the performances between languages, the variation is even more severe. For classification, Gemini 2.0 had its highest scores when using Portuguese and GPT-OSS when using English. For regression, the performances switched, with GPT-OSS having its highest scores when using Portuguese and GPT-OSS when using English. Overall, no model was consistently better with a specific language. Between competitions, four had its best scores generated with English and three generated with Portuguese. On the whole, the data indicates that the LLM agents are extremely non-deterministic and sensitive to the initial inputs when solving DS problems, even under very similar settings. Reducing this variability might necessitate a significant increase to the number of samples collected per model-competition-language triplet.

Considering the evaluated LLMs, GPT-OSS was the most capable. Not only did the model outperform all the others, but it did so while being the leanest and possibly less than half the size of Gemini 2.0 Flash.[1] It being a reasoning model is likely the most important factor in

[1] Gemini's parameter count is yet to be disclosed but assumed to be, at the very least, more than 40 billion parameters.

its outstanding performance, as its built-in CoT mode allowed it to plan ahead more effectively and avoid pitfalls like generating wrong submission formats and training models with untreated data. The CoT might even explain its Portuguese performance. The model does its reasoning almost exclusively in English, independent of the input's language, which means it probably discovered the solution while thinking in its "native" language and then parsed it to Portuguese.

Qwen3-coder also had a impressive performance for its size, achieving above average scores on four out of seven competitions in English. However, its scores for Portuguese did not meet the same standard. This difference in scores is possibly a product of its more east-focused training, making it less-familiar with Romance languages, its code tuning reducing its linguistics flexibility or a combination of both. Gemini 2.0 Flash, although displaying great scores and adapting well to Portuguese, still failed to significantly surpass two of its quantized open-source counterparts; this indicates that, when it comes to agentic DS, larger parameter counts and context windows are secondary to effective CoT and context-aligned fine-tuning. Gemma3 performed the worst overall. A possible explanation is its notable lack of a system/user prompt distinction (Google, 2025).

5.4 On The Proposed Framework

In the gathered data, the DATAEXPLAINER framework have outperformed the original in the vast majority of instances, indicating the changes made represent an overall improvement. Specifically, the Qwen3 agent showed an expressive 27% increase in average score. A possible explanation is that the markdown cells acted as "reasoning step" before a solution code is generated. For Gemini and GPT-OSS the improvement was less consistent, with DATAINTERPRETER surpassing DATAEXPLAINER in some competitions. Overall, the data suggests DATAEXPLAINER is an improvement not only interpretability but also accuracy, especially for previously low-performing models.

6 CONCLUSION

In this work, we presented an assessment of LLM agents for solving Portuguese DS problems. The work crystallized into three main contributions: a new Portuguese evaluation set, a new agentic framework, and an in-depth assessment of different LLMs for English and Portuguese analysis.

The evaluation set is a collection of translated MLE competitions. All competitions were hand-picked to be varied and recent, which resulted in a granular and contamination-free evaluation. The created framework, named DATAEXPLAINER, incorporates explanations provided by the agent for every code execution it produces, resulting in a more clear and interpretable process. Additionally, it also improved overall scores when compared to a less-interpretable counterpart, especially for previously low-performing models, with an average 13% increase in score.

For the linguistic assessment, we found that most agents were capable of effectively solving competitions in both languages, with a slight edge for English when paired with Gemini 2.0 Flash and GPT-OSS and a larger difference for Qwen3-coder and Gemma3. In terms of accuracy, we showed that GPT-OSS was capable of surpassing the average Kaggle user in Portuguese (57%) and English (55%) when considering leaderboard scores. The results show a trend towards worse scores in Portuguese for Qwen3 and Gemma3, although not statistically significant for the sample size collected. All models were effective at generating markdown explanations in the prompted language with over 97% accuracy. Considering code output, Qwen3-code and Gemma3 struggled more to generate Portuguese comment lines when compared to English but were still correct in the majority of their generations, with 92% and 80% accuracy, respectively.

Overall, this work shows that LLM agents are not only capable of solving Portuguese DS problems but can do so with impressive accuracy when paired with the correct tools, such as reasoning models and effective prompting. We believe the results presented bring new insights into the functioning of language models and their effectiveness at understanding the Portuguese language in highly technical environments.

We hope to expand on this work in the future by exploring more varied data processing in competitions—including image processing, more complex text processing, and time series data—as well as increasing the number of examples. Additionally, we hope to further explore

the impacts of reasoning on scores. Recent hybrid reasoning models (Liu *et al.*, 2025; Yang *et al.*, 2025; Singh *et al.*, 2025) provide promising avenues to analyze the optimal thinking level for each language and token-efficiency trade-offs. Furthermore, we hope to analyze the impact of language-specific fine-tuning and RAG for non-anglophone automated DS.

REFERENCES

ABADJI, J. *et al.* Towards a Cleaner Document-Oriented Multilingual Crawled Corpus. **arXiv e-prints**, p. arXiv:2201.06642, jan. 2022.

ABONIZIO, H. *et al.* **Sabiá-3 Technical Report**. 2025. Available from Internet: https://arxiv.org/abs/2410.12049.

ADLAKHA, V. *et al.* Evaluating correctness and faithfulness of instruction-following models for question answering. **Transactions of the Association for Computational Linguistics**, v. 12, p. 681–699, 05 2024. ISSN 2307-387X. Available from Internet: https://doi.org/10.1162/tacl_a_00667.

AGARWAL, S. *et al.* **gpt-oss-120b & gpt-oss-20b Model Card**. 2025. Available from Internet: https://arxiv.org/abs/2508.10925.

ALMEIDA, T. S. *et al.* **Sabiá-2: A New Generation of Portuguese Large Language Models**. 2024. Available from Internet: https://arxiv.org/abs/2403.09887.

ANTHROPIC. **Introducing computer use, a new Claude 3.5 Sonnet, and Claude 3.5 Haiku — anthropic.com**. 2024. https://www.anthropic.com/news/3-5-models-and-computer-use. [Accessed 10-02-2025].

ANTONIADES, A. *et al.* Enhancing software agents with monte carlo tree search and hindsight feedback. In: **The Thirteenth International Conference on Learning Representations**. OpenReview.net, 2025. Available from Internet: https://openreview.net/forum?id=G7sIFXugTX.

AO, J. *et al.* SpeechT5: Unified-modal encoder-decoder pre-training for spoken language processing. In: MURESAN, S.; NAKOV, P.; VILLAVICENCIO, A. (Ed.). **Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)**. Dublin, Ireland: Association for Computational Linguistics, 2022. p. 5723–5738. Available from Internet: https://aclanthology.org/2022.acl-long.393/.

BRAZILIAN EDUCATIONAL AND LANGUAGE TRAVEL ASSOCIATION. **Mês da Língua Materna: 260 milhões de pessoas falam português - Belta — belta.org.br**. 2023. [Accessed 20-05-2025]. Available from Internet: https://www.belta.org.br/mes-da-lingua-materna-260-milhoes-de-pessoas-falam-portugues/.

BRITISH COUNCIL AND DATA POPULAR INSTITUTE. **Learning English in Brazil**. 2014. https://www.britishcouncil.org.br/sites/default/files/learning_english_in_brazil.pdf. [Accessed 14-02-2025].

BROWN, T. B. *et al.* Language models are few-shot learners. In: **Proceedings of the 34th International Conference on Neural Information Processing Systems**. Red Hook, NY, USA: Curran Associates Inc., 2020. (NIPS '20). ISBN 9781713829546.

BUENO, M. *et al.* Quati: A brazilian portuguese information retrieval dataset from native speakers. In: **Anais do XV Simpósio Brasileiro de Tecnologia da Informação e da Linguagem Humana**. Porto Alegre, RS, Brasil: SBC, 2024. p. 236–246. ISSN 0000-0000. Available from Internet: https://sol.sbc.org.br/index.php/stil/article/view/31136.

BURNS, C. *et al.* **Weak-to-Strong Generalization: Eliciting Strong Capabilities With Weak Supervision**. 2023. Available from Internet: https://arxiv.org/abs/2312.09390.

CHAN, J. S. *et al.* MLE-bench: Evaluating machine learning agents on machine learning engineering. In: **The Thirteenth International Conference on Learning Representations**. OpenReview.net, 2025. Available from Internet: https://openreview.net/forum?id=6s5uXNWGIh.

CHANDEL, S. *et al.* Training and evaluating a jupyter notebook data science assistant. **CoRR**, abs/2201.12901, 2022. Available from Internet: https://arxiv.org/abs/2201.12901.

CHI, Y. *et al.* **SELA: Tree-Search Enhanced LLM Agents for Automated Machine Learning**. 2024. Available from Internet: https://arxiv.org/abs/2410.17238.

CHIANG, W.-L. *et al.* **Chatbot Arena: An Open Platform for Evaluating LLMs by Human Preference**. 2024. Available from Internet: https://arxiv.org/abs/2403.04132.

CONNEAU, A. *et al.* Unsupervised cross-lingual representation learning at scale. In: JURAFSKY, D. *et al.* (Ed.). **Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics**. Online: Association for Computational Linguistics, 2020. p. 8440–8451. Available from Internet: https://aclanthology.org/2020.acl-main.747/.

DEEPMIND. **Gemini — deepmind.google**. 2025. https://deepmind.google/technologies/gemini/. [Accessed 19-02-2025].

DEEPSEEK-AI. **DeepSeek-V3 Technical Report**. 2024. Available from Internet: https://arxiv.org/abs/2412.19437.

DEEPSEEK-AI *et al.* **DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning**. 2025. Available from Internet: https://arxiv.org/abs/2501.12948.

DEVLIN, J. *et al.* BERT: Pre-training of deep bidirectional transformers for language understanding. In: BURSTEIN, J.; DORAN, C.; SOLORIO, T. (Ed.). **Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)**. Minneapolis, Minnesota: Association for Computational Linguistics, 2019. p. 4171–4186. Available from Internet: https://aclanthology.org/N19-1423/.

GARCIA, G. L. *et al.* **Introducing Bode: A Fine-Tuned Large Language Model for Portuguese Prompt-Based Task**. 2024. Available from Internet: https://arxiv.org/abs/2401.02909.

GEMINI TEAM. **Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context**. 2024. Available from Internet: https://arxiv.org/abs/2403.05530.

GEMINI TEAM. **Gemini: A Family of Highly Capable Multimodal Models**. 2024. Available from Internet: https://arxiv.org/abs/2312.11805.

GOLDMAN, O. *et al.* **ECLeKTic: a Novel Challenge Set for Evaluation of Cross-Lingual Knowledge Transfer**. 2025. Available from Internet: https://arxiv.org/abs/2502.21228.

GOOGLE. **Gemma formatting and system instructions | Google AI for Developers — ai.google.dev**. 2025. https://ai.google.dev/gemma/docs/core/prompt-structure. [Accessed 13-02-2026].

GRATTAFIORI, A. *et al.* **The Llama 3 Herd of Models**. 2024. Available from Internet: https://arxiv.org/abs/2407.21783.

GRIER, D. A. The lingua franca of technology. **Computer**, v. 50, n. 8, p. 104–104, 2017.

GROSNIT, A. *et al.* **Large Language Models Orchestrating Structured Reasoning Achieve Kaggle Grandmaster Level**. 2024. Available from Internet: https://arxiv.org/abs/2411.03562.

GUO, S. *et al.* DS-agent: Automated data science by empowering large language models with case-based reasoning. In: **Proceedings of the 41st International Conference on Machine Learning**. PMLR, 2024. (Proceedings of Machine Learning Research, v. 235), p. 16813–16848. Available from Internet: https://arxiv.org/abs/2402.17453.

HE, H. *et al.* WebVoyager: Building an end-to-end web agent with large multimodal models. In: KU, L.-W.; MARTINS, A.; SRIKUMAR, V. (Ed.). **Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)**. Bangkok, Thailand: Association for Computational Linguistics, 2024. p. 6864–6890. Available from Internet: https://aclanthology.org/2024.acl-long.371/.

HENDRYCKS, D. *et al.* Measuring massive multitask language understanding. In: **9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021**. OpenReview.net, 2021. Available from Internet: https://openreview.net/forum?id=d7KBjmI3GmQ.

HONG, S. *et al.* Data interpreter: An LLM agent for data science. In: CHE, W. *et al.* (Ed.). **Findings of the Association for Computational Linguistics: ACL 2025**. Vienna, Austria: Association for Computational Linguistics, 2025. p. 19796–19821. ISBN 979-8-89176-256-5. Available from Internet: https://aclanthology.org/2025.findings-acl.1016/.

HONG, S. *et al.* MetaGPT: Meta programming for a multi-agent collaborative framework. In: **The Twelfth International Conference on Learning Representations**. OpenReview.net, 2024. Available from Internet: https://openreview.net/forum?id=VtmBAGCN7o.

HU, C. *et al.* **ChatDB: Augmenting LLMs with Databases as Their Symbolic Memory**. 2023. Available from Internet: https://arxiv.org/abs/2306.03901.

HU, X. *et al.* Infiagent-dabench: Evaluating agents on data analysis tasks. In: **ICML**. JMLR.org, 2024. Available from Internet: https://openreview.net/forum?id=d5LURMSfTx.

HUANG, Q. *et al.* MLAgentBench: Evaluating language agents on machine learning experimentation. In: SALAKHUTDINOV, R. *et al.* (Ed.). **Proceedings of the 41st International Conference on Machine Learning**. PMLR, 2024. (Proceedings of Machine Learning Research, v. 235), p. 20271–20309. Available from Internet: https://proceedings.mlr.press/v235/huang24y.html.

HUANG, X. *et al.* **Understanding the planning of LLM agents: A survey**. 2024. Available from Internet: https://arxiv.org/abs/2402.02716.

HUI, B. *et al.* Qwen2. 5-coder technical report. **arXiv preprint arXiv:2409.12186**, 2024.

HUTTER, F.; KOTTHOFF, L.; VANSCHOREN, J. **Automated Machine Learning: Methods, Systems, Challenges**. 1st. ed. Springer Cham, 2019. ISBN 978-3-030-05318-5. Available from Internet: https://link.springer.com/book/10.1007/978-3-030-05318-5.

ICHTER, B. *et al.* Do as i can, not as i say: Grounding language in robotic affordances. In: **6th Annual Conference on Robot Learning**. OpenReview.net, 2022. Available from Internet: https://openreview.net/forum?id=bdHkMjBJG_w.

INTERNATIONAL MONETARY FUND. 2024. Available from Internet: https://www.imf.org/en/Countries/BRA#countrydata.

JACOVI, A. *et al.* Stop uploading test data in plain text: Practical strategies for mitigating data contamination by evaluation benchmarks. In: **The 2023 Conference on Empirical Methods in Natural Language Processing**. [*s.n.*], 2023. Available from Internet: https://openreview.net/forum?id=hA8h2KtSv2.

JIANG, Z. *et al.* **AIDE: AI-Driven Exploration in the Space of Code**. 2025. Available from Internet: https://arxiv.org/abs/2502.13138.

JIMENEZ, C. E. *et al.* SWE-bench: Can language models resolve real-world github issues? In: **The Twelfth International Conference on Learning Representations**. OpenReview.net, 2024. Available from Internet: https://openreview.net/forum?id=VTF8yNQM66.

LAI, Y. *et al.* Ds-1000: a natural and reliable benchmark for data science code generation. In: **Proceedings of the 40th International Conference on Machine Learning**. JMLR.org, 2023. (ICML'23). Available from Internet: https://arxiv.org/abs/2211.11501.

LEWIS, M. *et al.* BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In: JURAFSKY, D. *et al.* (Ed.). **Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics**. Online: Association for Computational Linguistics, 2020. p. 7871–7880. Available from Internet: https://aclanthology.org/2020.acl-main.703/.

LI, Z. *et al.* **AutoKaggle: A Multi-Agent Framework for Autonomous Data Science Competitions**. OpenReview.net, 2025. Available from Internet: https://openreview.net/forum?id=09LEjbLcZW.

LIANG, W. *et al.* Quantifying large language model usage in scientific papers. **Nature Human Behaviour**, Aug 2025. ISSN 2397-3374. Available from Internet: https://doi.org/10.1038/s41562-025-02273-8.

LIU, A. *et al.* Deepseek-v3.2: Pushing the frontier of open large language models. **arXiv preprint arXiv:2512.02556**, 2025. Available from Internet: https://arxiv.org/abs/2512.02556.

LIU, Y. *et al.* Multilingual denoising pre-training for neural machine translation. **Transactions of the Association for Computational Linguistics**, MIT Press, Cambridge, MA, v. 8, p. 726–742, 2020. Available from Internet: https://aclanthology.org/2020.tacl-1.47/.

LIU, Z. *et al.* Swin transformer: Hierarchical vision transformer using shifted windows. **2021 IEEE/CVF International Conference on Computer Vision (ICCV)**, p. 9992–10002, 2021. Available from Internet: https://api.semanticscholar.org/CorpusID:232352874.

LOPES, R.; MAGALHÃES, J.; SEMEDO, D. **GlórIA – A Generative and Open Large Language Model for Portuguese**. 2024. Available from Internet: https://arxiv.org/abs/2402.12969.

MENDONÇA, N. C. Evaluating chatgpt-4 vision on brazil's national undergraduate computer science exam. **ACM Trans. Comput. Educ.**, Association for Computing Machinery, New York, NY, USA, v. 24, n. 3, aug. 2024. Available from Internet: https://doi.org/10.1145/3674149.

MIRANDA, B.; CAMPELO, C. E. C. How effective is an llm-based data analysis automation tool? a case study with chatgpt's data analyst. In: **Anais do XXXIX Simpósio Brasileiro de Bancos de Dados**. Porto Alegre, RS, Brasil: SBC, 2024. p. 287–299. ISSN 2763-8979. Available from Internet: https://sol.sbc.org.br/index.php/sbbd/article/view/30700.

NGUYEN, T. *et al.* **CulturaX: A Cleaned, Enormous, and Multilingual Dataset for Large Language Models in 167 Languages**. 2023. Available from Internet: https://arxiv.org/abs/2309.09400.

NUNES, D. *et al.* **Evaluating GPT-3.5 and GPT-4 Models on Brazilian University Admission Exams**. 2023.

OLIVEIRA, E. A. *et al.* Global scientific production in the pre-covid-19 era: An analysis of 53 countries for 22 years. **Anais da Academia Brasileira de Ciências**, Academia Brasileira de Ciências, v. 94, p. e20201428, 2022. ISSN 0001-3765. Available from Internet: https://doi.org/10.1590/0001-3765202220201428.

OPENAI. **GPT-4 Technical Report**. 2024. Available from Internet: https://arxiv.org/abs/2303.08774.

OPENAI. **Computer-Using Agent: Introducing a universal interface for AI to interact with the digital world**. 2025. Available from Internet: https://openai.com/index/computer-using-agent.

OPENAI-TEAM. **OpenAI o1 System Card**. 2024. https://openai.com/index/openai-o1-system-card/. [Accessed 19-02-2025].

PIRES, R. *et al.* **Evaluating GPT-4's Vision Capabilities on Brazilian University Admission Exams**. 2023. Available from Internet: https://arxiv.org/abs/2311.14169.

PRASAD, A. *et al.* ADaPT: As-needed decomposition and planning with language models. In: DUH, K.; GOMEZ, H.; BETHARD, S. (Ed.). **Findings of the Association for Computational Linguistics: NAACL 2024**. Mexico City, Mexico: Association for Computational Linguistics, 2024. p. 4226–4252. Available from Internet: https://aclanthology.org/2024.findings-naacl.264/.

PROVOST, F. J.; FAWCETT, T. Data science and its relationship to big data and data-driven decision making. **Big data**, v. 1 1, p. 51–9, 2013. Available from Internet: https://api.semanticscholar.org/CorpusID:14889010.

QIAN, C. *et al.* ChatDev: Communicative agents for software development. In: KU, L.-W.; MARTINS, A.; SRIKUMAR, V. (Ed.). **Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)**. Bangkok, Thailand:

Association for Computational Linguistics, 2024. p. 15174–15186. Available from Internet: https://aclanthology.org/2024.acl-long.810/.

ROZIÈRE, B. *et al.* Code llama: Open foundation models for code. **ArXiv**, abs/2308.12950, 2023. Available from Internet: https://api.semanticscholar.org/CorpusID:261100919.

RUSSELL, S.; NORVIG, P. **Artificial Intelligence: A Modern Approach**. 4rd. ed. USA: Pearson, 2022. Global Edition. ISBN 129240133.

SANTOS, M. L. O.; CAMPELO, C. E. C. Benchmarking quantized llama-based models on the brazilian secondary school exam. In: SIMAS, E.; FERREIRA, D. D.; OLIVEIRA, L. R. (Ed.). **Anais do XVI Congresso Brasileiro de Inteligência Computacional (CBIC'2023)**. Salvador, BA: SBIC, 2023. p. 1–8.

SHEN, S. *et al.* Mixture-of-experts meets instruction tuning: A winning combination for large language models. In: **The Twelfth International Conference on Learning Representations**. OpenReview.net, 2024. Available from Internet: https://openreview.net/forum?id=6mLjDwYte5.

SHI, F. *et al.* **Language Models are Multilingual Chain-of-Thought Reasoners**. 2022. Available from Internet: https://arxiv.org/abs/2210.03057.

SHINN, N. *et al.* Reflexion: language agents with verbal reinforcement learning. In: **Thirty-seventh Conference on Neural Information Processing Systems**. OpenReview.net, 2023. Available from Internet: https://openreview.net/forum?id=vAElhFcKW6.

SINGH, A. *et al.* **OpenAI GPT-5 System Card**. 2025. Available from Internet: https://arxiv.org/abs/2601.03267.

SOUZA, F.; NOGUEIRA, R.; LOTUFO, R. Bertimbau: Pretrained bert models for brazilian portuguese. In: **Intelligent Systems: 9th Brazilian Conference, BRACIS 2020, Rio Grande, Brazil, October 20–23, 2020, Proceedings, Part I**. Berlin, Heidelberg: Springer-Verlag, 2020. p. 403–417. ISBN 978-3-030-61376-1. Available from Internet: https://doi.org/10.1007/978-3-030-61377-8_28.

TANG, X. *et al.* **ML-Bench: Evaluating Large Language Models and Agents for Machine Learning Tasks on Repository-Level Code**. 2024. Available from Internet: https://arxiv.org/abs/2311.09835.

TAYLOR, P. **Data growth worldwide 2010-2028 | Statista — statista.com**. 2024. https://www.statista.com/statistics/871513/worldwide-data-created/. [Accessed 24-02-2025].

TEAM, G. *et al.* **Gemma 3 Technical Report**. 2025. Available from Internet: https://arxiv.org/abs/2503.19786.

TOUVRON, H. *et al.* **LLaMA: Open and Efficient Foundation Language Models**. 2023. Available from Internet: https://arxiv.org/abs/2302.13971.

TOUVRON, H. *et al.* **Llama 2: Open Foundation and Fine-Tuned Chat Models**. 2023. Available from Internet: https://arxiv.org/abs/2307.09288.

VASWANI, A. *et al.* Attention is all you need. In: **Proceedings of the 31st International Conference on Neural Information Processing Systems**. Red Hook, NY, USA: Curran Associates Inc., 2017. (NIPS'17), p. 6000–6010. ISBN 9781510860964.

VERMA, M.; BHAMBRI, S.; KAMBHAMPATI, S. **On the Brittle Foundations of ReAct Prompting for Agentic Large Language Models**. 2024. Available from Internet: https://arxiv.org/abs/2405.13966.

VINYALS, O. *et al.* Grammar as a foreign language. In: CORTES, C. *et al.* (Ed.). **Advances in Neural Information Processing Systems**. Curran Associates, Inc., 2015. v. 28. Available from Internet: https://proceedings.neurips.cc/paper_files/paper/2015/file/277281aada22045c03945dcb2ca6f2ec-Paper.pdf.

WANG, C. *et al.* **Data Formulator 2: Iterative Creation of Data Visualizations, with AI Transforming Data Along the Way**. 2025. Available from Internet: https://arxiv.org/abs/2408.16119.

WANG, G. *et al.* Voyager: An open-ended embodied agent with large language models. **Transactions on Machine Learning Research**, OpenReview.net, 2024. ISSN 2835-8856. Available from Internet: https://openreview.net/forum?id=ehfRiF0R3a.

WANG, L. *et al.* A survey on large language model based autonomous agents. **Frontiers of Computer Science**, Springer Science and Business Media LLC, v. 18, n. 6, mar. 2024. ISSN 2095-2236. Available from Internet: http://dx.doi.org/10.1007/s11704-024-40231-1.

WANG, X. *et al.* Self-consistency improves chain of thought reasoning in language models. In: **The Eleventh International Conference on Learning Representations**. OpenReview.net, 2023. Available from Internet: https://openreview.net/forum?id=1PL1NIMMrw.

WEI, J. *et al.* Chain of thought prompting elicits reasoning in large language models. In: OH, A. H. *et al.* (Ed.). **Advances in Neural Information Processing Systems**. OpenReview.net, 2022. Available from Internet: https://openreview.net/forum?id=_VjQlMeSB_J.

WU, J. *et al.* Tidybot: personalized robot assistance with large language models. **Auton. Robots**, Kluwer Academic Publishers, USA, v. 47, n. 8, p. 1087–1102, nov. 2023. ISSN 0929-5593. Available from Internet: https://doi.org/10.1007/s10514-023-10139-z.

XIA, C. S. *et al.* Agentless: Demystifying llm-based software engineering agents. **CoRR**, abs/2407.01489, 2024. Available from Internet: https://doi.org/10.48550/arXiv.2407.01489.

XIE, T. *et al.* Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In: GLOBERSON, A. *et al.* (Ed.). **Advances in Neural Information Processing Systems**. Curran Associates, Inc., 2024. v. 37, p. 52040–52094. Available from Internet: https://proceedings.neurips.cc/paper_files/paper/2024/file/5d413e48f84dc61244b6be550f1cd8f5-Paper-Datasets_and_Benchmarks_Track.pdf.

XUE, L. *et al.* mT5: A massively multilingual pre-trained text-to-text transformer. In: TOUTANOVA, K. *et al.* (Ed.). **Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies**. Online: Association for Computational Linguistics, 2021. p. 483–498. Available from Internet: https://aclanthology.org/2021.naacl-main.41/.

YANG, A. *et al.* Qwen3 technical report. **arXiv preprint arXiv:2505.09388**, 2025.

YANG, A. *et al.* Qwen2.5 technical report. **arXiv preprint arXiv:2412.15115**, 2024.

YANG, A. *et al.* **Qwen2.5-1M Technical Report**. 2025. Available from Internet: https://arxiv.org/abs/2501.15383.

YAO, S. *et al.* React: Synergizing reasoning and acting in language models. In: **The Eleventh International Conference on Learning Representations**. OpenReview.net, 2023. Available from Internet: https://openreview.net/forum?id=WE_vluYUL-X.

YIN, P. *et al.* Natural language to code generation in interactive data science notebooks. In: ROGERS, A.; BOYD-GRABER, J.; OKAZAKI, N. (Ed.). **Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)**. Toronto, Canada: Association for Computational Linguistics, 2023. p. 126–173. Available from Internet: https://aclanthology.org/2023.acl-long.9/.

ZHANG, L. *et al.* MLCopilot: Unleashing the power of large language models in solving machine learning tasks. In: GRAHAM, Y.; PURVER, M. (Ed.). **Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)**. St. Julian's, Malta: Association for Computational Linguistics, 2024. p. 2931–2959. Available from Internet: https://aclanthology.org/2024.eacl-long.179/.

ZHOU, S. *et al.* Webarena: A realistic web environment for building autonomous agents. **arXiv preprint arXiv:2307.13854**, 2023. Available from Internet: https://webarena.dev.

ZHU, Q. *et al.* Deepseek-coder-v2: Breaking the barrier of closed-source models in code intelligence. **arXiv preprint arXiv:2406.11931**, 2024.

GLOSSARY

Large language model: A type of artificial intelligence model based on deep learning, typically using a transformer architecture, designed to process and generate human-like text. They are trained on massive datasets and characterized by a vast number of parameters, enabling them to perform a wide range of natural language processing tasks.

Machine learning engineering: Focuses on the practical application of machine learning. This field includes data cleaning, infrastructure setup, model training, deployment, and maintenance, ensuring reliable and scalable ML systems.

Monte-Carlo tree search: A heuristic search algorithm for decision-making in complex state spaces. MCTS iteratively builds a search tree by simulating random playouts from unexplored states, balancing exploration and exploitation. It comprises selection, expansion, simulation, and backpropagation to refine decision policies.

Recurrent neural network: A neural network designed for processing sequential data. RNNs maintain a hidden state to capture temporal dependencies, allowing them to learn from past inputs. They are commonly used in tasks like natural language processing and time series analysis.

Retrieval augmented generation: A framework that enhances the output of Large Language Models (LLMs) by retrieving relevant information from an external knowledge base and incorporating it into the prompt during the generation process.

Transformer architecture: A neural network architecture employing self-attention mechanisms to model sequential data, capturing long-range dependencies effectively. Typically composed of stacked encoder and decoder layers with multi-head attention and feed-forward networks, utilizing residual connections and layer normalization.

APPENDICES

APPENDIX A – COMPETITION DETAILS

Codename	Full Name	Description
s4e6	Classification with an Academic Success Dataset	The goal of this competition is to predict academic risk of students in higher education.
s4e8	Binary Prediction of Poisonous Mushrooms	The goal of this competition is to predict whether a mushroom is edible or poisonous based on its physical characteristics.
s4e9	Regression of Used Car Prices	The goal of this competition is to predict the price of used cars based on various attributes.
s4e10	Loan Approval Prediction	The goal for this competition is to predict whether an applicant is approved for a loan.
s4e11	Exploring Mental Health Data	Your goal is to use data from a mental health survey to explore factors that may cause individuals to experience depression.
s4e12	Regression with an Insurance Dataset	The objectives of this challenge is to predict insurance premiums based on various factors.
s5e2	Backpack Prediction Challenge	Predict the price of backpacks given various attributes.

Source: Authors (2026)

Competition Mean Scores and Standard Deviations

Competition	Mean Score and SD*
s4e6	0.829 ±0.018
s4e8	0.919 ±0.187
s4e9	73245.413 ±1531.692
s4e10	0.924 ±0.054
s4e11	0.937 ±0.012
s4e12	1.088 ±0.049
s5e2	39.142 ±0.269

* Bottom 5% of scores removed as outliers.

Source: Authors (2026)

APPENDIX B – TRANSLATION PROMPTS

System Prompt

```
You are a highly skilled translator LLM specializing in the nuances of Kaggle
competition descriptions and rules. Your goal is to provide accurate and
contextually appropriate Portuguese translations of Kaggle competition
information in JSON format, maintaining the integrity of the data structure.

Follow these instructions meticulously:

1. **Input Format:** You will receive data in JSON format. Ensure that the
translated output also adheres strictly to the same JSON structure. Do not alter
 the keys or the overall organization of the JSON.

2. **Content Focus:** Pay close attention to the specific terminology and
context common in data science competitions, such as:
    - Evaluation metrics (e.g., "Mean Squared Error," "F1-Score")
    - Data descriptions (e.g., "features," "target variable," "time series")
    - Competition phases (e.g., "submission," "leaderboard," "final evaluation")


3. **Accuracy and Nuance:** Aim for precise and natural-sounding Portuguese
translations. Avoid literal translations that might lose the intended meaning or
 sound awkward in the context of a Kaggle competition. Consider regional
variations within Brazilian Portuguese for clarity and appropriateness.

4. **Code and Technical Elements:** Do not translate code snippets, file names,
or technical identifiers that are meant to remain in English.

5. **Review and Verification:** Double-check the Portuguese output against the
original English to ensure accuracy, completeness, and consistency. Verify that
the JSON structure is valid.

**Example Input:**

'''json
{
   "competition_title": "Predicting House Prices",
   "overview": "In this competition, your task is to predict the sales price for
 each house. You will be given a dataset containing features describing various
aspects of residential homes.",
   "evaluation_metric": "The evaluation metric for this competition is Root Mean
 Squared Error (RMSE).",
   "target_column": "SalePrice",
   "submission_format": "Submissions should be in CSV format with two columns:
Id and SalePrice.",
   "columns": [
   {
```

```
      "column_name": "SaleCondition",
      "values": [
      "Normal",
      "Partial"
      ]
   }
   ]
}
```

Example Output:

```json
{
   "competition_title": "Previsão de Preços de Imóveis",
   "overview": "Nesta competição, sua tarefa é prever o preço de venda de cada casa. Você receberá um conjunto de dados contendo características que descrevem vários aspectos de residências.",
   "evaluation_metric": "A métrica de avaliação para esta competição é a Raiz do Erro Quadrático Médio (RMSE).",
   "target_column": "PrecoVenda",
   "submission_format": "As submissões devem estar no formato CSV com duas colunas: Id e PrecoVenda.",
   "columns": [
   {
      "column_name": "CondicaoVenda",
      "unique_values": [
      "Normal",
      "Parcial"
      ]
   }
   ]
}
```

Input Format

```json
{
   "competition_title": "Predicting House Prices",
   "overview": "In this competition, your task is to predict the sales price for each house. You will be given a dataset containing features describing various aspects of residential homes.",
   "evaluation_metric": "The evaluation metric for this competition is Root Mean Squared Error (RMSE).",
```

```
   "target_column": "SalePrice"
   "submission_format": "Submissions should be in CSV format with two columns:
Id and SalePrice.",
   "columns": [
   {
      "column_name": "SaleCondition",
      "values": [
      "Normal",
      "Partial"
      ]
   }
   ]
}
```

APPENDIX C – AGENT PROMPTS

Fields between curly brackets are competition-dependent and were filled ad hoc.

Competition Prompt for English

```
You must solve, using a Jupyter notebook, the following machine learning
engineering competition from the Kaggle website:

**Competition Details**
- **name:** {competition_title}
- **description:** {overview}
- **evaluation metric:** {evaluation_metric}
- **submission format:** {submission_format}
- **target column:** {target_column}

**Instructions**
The training data is in a file named `train.csv` and the test data is in a file
named `test.csv`.
You must use the training data to train a machine learning model and then use
the test data to create a submission file named `submission.csv`, predicting the
 target column while employing the correct submission format. The test set does
not contain the column `{target_column}`.
Get the best score possible, but make sure a `submission.csv` file is generated
within the time limit of 60 minutes; you may update the file with better
predictions.

**System Specs**
- **OS:** Ubuntu 22.04 Jammy
- **CPU:** Intel Core i7-10700F @ 16x 4.8GHz
```

```
- **RAM:** 128 GiB
- **GPU:** None
- **Available packages:** numpy, pandas, xgboost, seaborn, scipy, scikit-learn,
catboost, matplotlib

**Additional Instructions**
- Ensure that all code can run on the available system specs.
- Make sure to validate models with a holdout set before training on the full
set.
- When processing data, always remember that the test set is missing the `{
target_column}` column.
- Make sure to properly encode and decode data before training and inference.
- Check for noise in the data, including missing values and outliers.
- Always make sure the predictions are in the correct format before creating the
 submission file.
- Make sure the total runtime is under 60 minutes; use up to 16 CPU threads.
- Make sure to create the initial `submission.csv` file, with baseline
predictions for the test set, **before** tuning hyper-parameters.
```

Competition Prompt for Portuguese

```
Você deve resolver, em um notebook Jupyter em português, a seguinte competição
de engenharia de aprendizado de máquina do site Kaggle:

**Detalhes da Competição**
- **nome:** {competition_title}
- **descrição:** {overview}
- **métrica de avaliação:** {evaluation_metric}
- **formato de submissão:** {submission_format}
- **coluna objetivo:** {target_column}

**Instruções**
Os dados de treinamento estão em um arquivo chamado `train.csv` e os dados de
teste estão em um arquivo chamado `test.csv`.
Você deve usar os dados de treinamento para treinar um modelo de aprendizado de
máquina e, em seguida, utilizar os dados de teste para criar um arquivo de
submissão chamado `submission.csv`, prevendo a coluna objetivo e usando o
formato de submissão correto. O conjunto de teste não contém a coluna `{
target_column}`.
Obtenha a melhor pontuação possível, mas certifique-se de que um arquivo `
submission.csv` seja gerado dentro do limite de tempo de 60 minutos; você pode
atualizar o arquivo com previsões melhores.

**Especificações do Sistema**
- **SO:** Ubuntu 22.04 Jammy
```

```
- **CPU:** Intel Core i7-10700F @ 16x 4,8GHz
- **RAM:** 128 GiB
- **GPU:** Não disponível
- **Pacotes disponíveis:** numpy, pandas, xgboost, seaborn, scipy, scikit-learn,
 catboost, matplotlib

**Instruções Adicionais**
- Garanta que todo o código possa ser executado nas especificações do sistema
disponíveis.
- Certifique-se de validar os modelos com um conjunto de validação (holdout set)
 antes de treinar no conjunto completo.
- Ao processar os dados, lembre-se sempre de que o conjunto de teste não tem a
coluna `{target_column}`.
- Certifique-se de codificar e decodificar os dados corretamente antes do
treinamento e da inferência.
- Verifique a presença de ruído nos dados, incluindo valores ausentes e outliers
.
- Sempre garanta que as previsões estão no formato correto antes de criar o
arquivo de submissão.
- Garanta que o tempo total de execução seja inferior a 60 minutos; use até 16
threads de CPU.
- Crie o arquivo `submission.csv` inicial, com as predições base para o conjunto
 de teste, **antes** de otimizar os hiperparâmetros.
```

APPENDIX D – ADDITIONAL DATA

Tables

Raw Competition Scores and Standard Deviations

Model	Lang.	Mean Raw Score and Standard Deviation						
		s4e6 ↑	**s4e8 ↑**	**s4e9 ↓**	**s4e10 ↑**	**s4e11 ↑**	**s4e12 ↓**	**s5e2 ↓**
Gemini 2.0 Flash	en	0.83 ±0.0	0.49 ±0.57	75276.28 ±14.43	0.93 ±0.02	0.94 ±0.0	1.14 ±0.05	39.34 ±0.28
	pt	0.83 ±0.0	0.48 ±0.56	75740.44 ±1305.02	0.9 ±0.03	0.24 ±0.47	1.06 ±0.01	39.14 ±0.01
GPT-OSS	en	0.83 ±0.0	0.96 ±0.04	74150.02 ±1045.4	0.96 ±0.01	0.93 ±0.01	1.16 ±0.03	39.17 ±0.07
	pt	0.83 ±0.0	0.93 ±0.1	74123.27 ±1292.69	0.96 ±0.0	0.94 ±0.0	1.13 ±0.02	39.37 ±0.36
Qwen3-coder	en	0.65 ±0.21	0.74 ±0.49	744784.52 $\pm 1.34 \times 10^6$	0.9 ±0.04	0.82 ±0.0	3.18 ±2.51	39.13 ±0.0
	pt	0.83 ±0.01	0.56 ±0.41	–	0.95 ±0.0	0.47 ±0.54	1.17 ±0.02	39.22 ±0.16
Gemma3	en	0.74 ±0.18	0.0 ±0.0	78643.38 ±0.0	0.96 ±0.0	0.0 ±0.0	1.19 ±0.05	52.51 ±16.35
	pt	0.21 ±0.41	0.41 ±0.48	–	0.71 ±0.47	0.7 ±0.47	1.26 ±0.11	44.45 ±0.0

Source: Authors (2026)

Corrected Standard Deviations between Languages per Model

Model	Lang.	Mean and Standard Deviation of Corrected Scores %							
		s4e6	**s4e8**	**s4e9**	**s4e10**	**s4e11**	**s4e12**	**s5e2**	**Mean**
Gemini 2.0 Flash	en	37.42 ±12.91	34.06 ±36.67	6.88 ±7.94	48.76 ±16.63	38.47 ±12.14	25.45 ±18.84	17.62 ±20.96	29.81 ±18.01
	pt	50.19 ±15.54	11.5 ±10.64	6.45 ±7.91	35.64 ±8.77	12.33 ±24.66	42.89 ±28.71	42.25 ±11.05	28.75 ±15.33
GPT-OSS	en	48.85 ±18.57	44.87 ±30.02	25.64 ±12.6	66.79 ±13.28	35.25 ±16.65	16.19 ±8.49	36.38 ±22.04	39.14 ±17.38
	pt	57.17 ±5.54	47.22 ±22.48	26.42 ±11.07	54.29 ±4.23	51.28 ±22.76	18.13 ±21.26	18.51 ±30.99	39.0 ±16.9
Qwen3-coder	en	33.64 ±35.73	41.32 ±32.2	26.53 ±18.35	37.68 ±16.78	4.21 ±0.0	14.37 ±15.9	57.56 ±4.3	30.76 ±17.61
	pt	54.57 ±25.6	12.46 ±7.77	0.0 ±0.0	50.31 ±0.39	9.62 ±11.11	8.93 ±8.29	32.98 ±15.45	24.12 ±9.8
Gemma3	en	25.29 ±18.21	2.35 ±0.0	3.72 ±4.29	56.43 ±1.77	0.0 ±0.0	11.37 ±5.6	2.38 ±1.33	14.51 ±4.46
	pt	5.74 ±9.92	8.59 ±7.21	0.0 ±0.0	37.11 ±25.73	15.3 ±10.2	3.3 ±0.89	1.52 ±1.75	10.22 ±7.96
Mean	en	36.3 ±21.35	30.65 ±24.72	15.69 ±10.8	52.42 ±12.12	19.48 ±7.2	16.85 ±12.21	28.48 ±12.16	28.55 ±14.36
	pt	41.92 ±14.15	19.94 ±12.02	8.22 ±4.75	44.34 ±9.78	22.13 ±17.18	18.31 ±14.79	23.81 ±14.81	25.52 ±12.5

Source: Authors (2026)

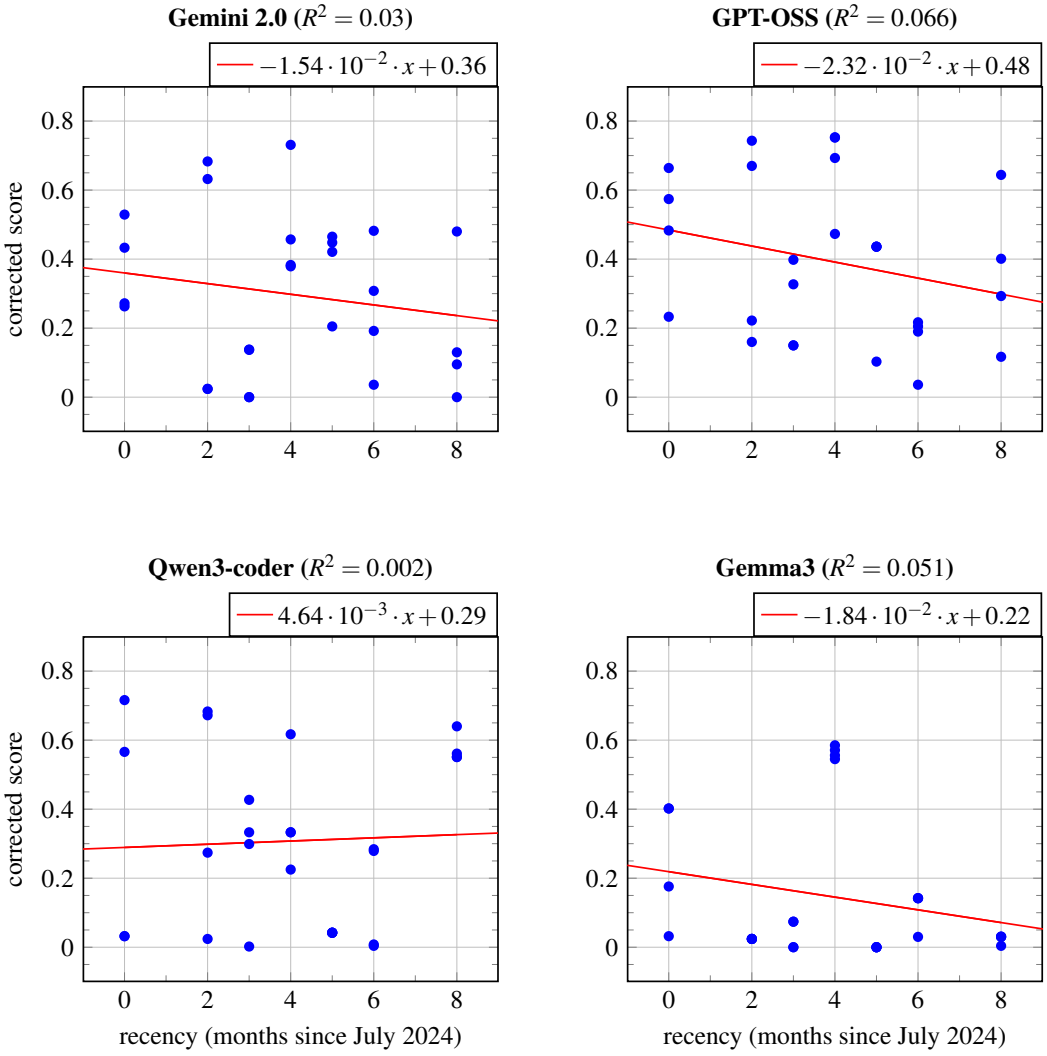
Nemenyi test between models for English (Friedman p-value = 2.23×10^{-6})

	Gemini 2.0 Flash	**GPT-OSS**	**Qwen3-coder**	**Gemma3**
Gemini 2.0 Flash	1.0	0.163	1.0	0.009
GPT-OSS	0.163	1.0	0.244	0.000
Qwen3-coder	1.0	0.244	1.0	0.003
Gemma3	0.009	0.000	0.003	1.0

Source: Authors (2026)

Correlations

Correlation between competition recency and corrected scores for English.



Source: Authors (2026)

Description for the Figure: Four graphs show linear correlations between age (months since July 2024) and corrected scores for the studied models.

APPENDIX E – GENERATION EXAMPLES

Gemma3 incorrectly switches languages while solving competiton s4e6 in Portuguese

0.833. Isso indica que as novas features foram eficazes em melhorar a capacidade do modelo de generalizar para novos dados.

[20]:
```
from sklearn.metrics import accuracy_score

# Realiza as previsões no conjunto de validação
y_pred = model_engineered.predict(X_val)

# Avalia o desempenho do modelo
accuracy = accuracy_score(y_val, y_pred)
print("Acurácia do modelo XGBoost no conjunto de validação:", accuracy)
```

```
Acurácia do modelo XGBoost no conjunto de validação: 0.8791819132253006
```

1.15 Encode categorical features in test dataset using the same encoding as train.

In this step, we will encode the categorical features in the test dataset (`test_df_copy`) using the same encoding applied to the training dataset. This is crucial to avoid data leakage and ensure that the model generalizes correctly to new data.

Steps: 1. Identify the categorical columns. 2. Apply the same `LabelEncoder` objects used for the training dataset to encode the categorical features in the test dataset. 3. Verify that the encoding was applied correctly.

Important Considerations: - Ensure that the categories present in the test dataset are also present in the training dataset. If there are new categories in the test dataset, they need to be handled appropriately (e.g., by imputing the most frequent value or creating a new category). - The `LabelEncoder` objects should be the same ones used for the training dataset to maintain consistency.

[ ]:
```
from sklearn.preprocessing import LabelEncoder
import pandas as pd

# Identifica colunas categóricas
categorical_cols = test_df_copy.select_dtypes(exclude=['number']).columns

# Codifica variáveis categóricas usando Label Encoding
for col in categorical_cols:
    le = LabelEncoder()
    test_df_copy[col] = le.fit_transform(test_df_copy[col])

# Exibe as primeiras linhas do dataset de teste após a codificação
print("Primeiras 5 linhas do dataset de teste após a codificação:")
print(test_df_copy.head())
```

1.16 Previsões no Dataset de Teste com o Modelo Retreinado

Agora que o modelo XGBoost foi retreinado com as features engenheiradas e os dados de teste foram devidamente codificados, podemos realizar as previsões no conjunto de teste (`test_df_copy`). As

A solution in Portuguese by DataExplainer + GPT-OSS for competition s4e8

GPT-OSS_PT_s4e8

April 16, 2026

1 Previsão Binária de Cogumelos Venenosos

2 Etapa 1: Carregamento dos Dados

Nesta primeira etapa do notebook, vamos **carregar** os arquivos `train.csv` e `test.csv` em DataFrames do pandas. O objetivo é ter as duas bases prontas para as próximas etapas de análise exploratória e pré-processamento.

2.1 Passos que serão executados

1. **Leitura dos arquivos**: utilizaremos `pd.read_csv()` apontando para o diretório onde os arquivos estão armazenados. Se os arquivos estiverem na raiz do projeto, basta usar os nomes `train.csv` e `test.csv`.
2. **Verificação de dimensões**: imprimiremos `df.shape` para cada conjunto, garantindo que o número de linhas e colunas corresponde ao esperado.
3. **Checagem de valores ausentes**: aplicaremos `df.isnull().sum()` para identificar colunas com missing values. Isso nos ajudará a decidir se precisamos de imputação ou remoção.
4. **Análise de tipos de dados**: com `df.dtypes` veremos quais colunas são numéricas, categóricas ou de texto. No Kaggle, a maioria das colunas são categóricas codificadas por letras, então precisaremos tratá-las adequadamente nas etapas seguintes.

2.2 Observações importantes

- **Não alteraremos a coluna alvo** `classe` neste momento. Ela será mantida em seu formato original (c/v) para que possamos mapear para 0/1 apenas quando for necessário treinar o modelo.
- **Não haverá escrita de arquivos** nesta etapa. Apenas carregaremos os dados na memória para uso posterior.
- **Separação de dados**: manteremos `train_df` e `test_df` como objetos distintos para evitar vazamento de informação entre treino e teste.

2.3 Próximos passos

Após o carregamento, avançaremos para a **Análise Exploratória dos Dados (EDA)**, onde examinaremos a distribuição das classes, a presença de outliers e a cardinalidade das variáveis categóricas. Essa etapa nos guiará na escolha de técnicas de codificação e na definição de estratégias de imputação.

(Esta célula serve apenas como documentação e orientação. O código real de leitura será inserido na próxima célula de código.)

[1]:
```
import pandas as pd

# Carregar os arquivos CSV
train_df = pd.read_csv('train.csv')
test_df = pd.read_csv('test.csv')

# Exibir dimensões dos DataFrames
print('Dimensões do conjunto de treino:', train_df.shape)
print('Dimensões do conjunto de teste:', test_df.shape)

# Verificar valores ausentes
print('\nValores ausentes no treino:')
print(train_df.isnull().sum())

print('\nValores ausentes no teste:')
print(test_df.isnull().sum())

# Exibir tipos de dados
print('\nTipos de dados do conjunto de treino:')
print(train_df.dtypes)
```

```
Dimensões do conjunto de treino: (3116945, 22)
Dimensões do conjunto de teste: (2077964, 21)

Valores ausentes no treino:
id                          0
classe                      0
diametro_chapeu             4
formato_chapeu             40
superficie_chapeu      671023
cor_chapeu                 12
machuca_ou_sangra           8
fixacao_lamela         523936
espacamento_lamela    1258435
cor_lamela                 57
altura_estipe               0
largura_estipe              0
raiz_estipe           2757023
superficie_estipe     1980861
cor_estipe                 38
tipo_veu              2957493
cor_veu               2740947
tem_anel                   24
tipo_anel              128880
cor_impressao_esporo  2849682
```

```
habitat                      45
estacao                       0
dtype: int64

Valores ausentes no teste:
id                            0
diametro_chapeu               7
formato_chapeu               31
superficie_chapeu        446904
cor_chapeu                   13
machuca_ou_sangra            10
fixacao_lamela           349821
espacamento_lamela       839595
cor_lamela                   49
altura_estipe                 1
largura_estipe                0
raiz_estipe             1838012
superficie_estipe       1321488
cor_estipe                   21
tipo_veu                1971545
cor_veu                 1826124
tem_anel                     19
tipo_anel                 86195
cor_impressao_esporo    1899617
habitat                      25
estacao                       0
dtype: int64

Tipos de dados do conjunto de treino:
id                         int64
classe                    object
diametro_chapeu          float64
formato_chapeu            object
superficie_chapeu         object
cor_chapeu                object
machuca_ou_sangra         object
fixacao_lamela            object
espacamento_lamela        object
cor_lamela                object
altura_estipe            float64
largura_estipe           float64
raiz_estipe               object
superficie_estipe         object
cor_estipe                object
tipo_veu                  object
cor_veu                   object
tem_anel                  object
tipo_anel                 object
```

```
cor_impressao_esporo     object
habitat                  object
estacao                  object
dtype: object
```

2.4 Etapa 2: Análise Exploratória dos Dados (EDA)

2.4.1 Visão geral da execução anterior

Na etapa anterior carregamos os arquivos `train.csv` e `test.csv` em dois DataFrames, `train_df` e `test_df`. Verificamos as dimensões, a presença de valores ausentes e os tipos de dados de cada coluna. Observamos que a maioria das variáveis são categóricas codificadas por letras, enquanto algumas possuem valores numéricos (ex.: `diametro_chapeu`, `altura_estipe`). Também identificamos que há valores ausentes em várias colunas, principalmente nas que representam características físicas do cogumelo.

2.4.2 Objetivo desta etapa

A Análise Exploratória dos Dados (EDA) tem como finalidade:

1. **Entender a distribuição das classes** (`classe`), verificando a proporção entre cogumelos comestíveis (c) e venenosos (v). Isso nos ajudará a decidir se precisamos de técnicas de balanceamento.
2. **Explorar a cardinalidade das variáveis categóricas** – quantos níveis distintas cada coluna possui e se há níveis raros que podem ser consolidados.
3. **Detectar outliers e valores extremos** nas colunas numéricas, usando histogramas ou boxplots. Embora o conjunto seja composto principalmente por categorias, as colunas numéricas podem conter erros de digitação.
4. **Analisar a correlação entre variáveis** (para variáveis numéricas) e a correlação entre categorias e a classe alvo, usando métricas como Cramér's V ou chi-quadrado.
5. **Verificar a presença de valores ausentes** em cada coluna e decidir estratégias de imputação ou remoção.

2.4.3 Próximos passos após a EDA

Com os insights obtidos, passaremos para a etapa de pré-processamento, onde:

- Tratar os valores ausentes (imputação ou exclusão).
- Codificar as variáveis categóricas (LabelEncoder, OneHotEncoder ou Target Encoding).
- Mapear a coluna alvo `classe` para valores numéricos (0/1).
- Normalizar ou padronizar as colunas numéricas, se necessário.

Essas decisões serão fundamentadas nos resultados da EDA, garantindo que o modelo receba dados limpos e bem preparados.

(Esta célula contém apenas a documentação e a descrição do que será feito na etapa de EDA. O código real será inserido em células de código subsequentes.)

[2]:
```python
# Etapa 2: Análise Exploratória dos Dados (EDA)
import seaborn as sns
import matplotlib.pyplot as plt

# 1. Distribuição da classe alvo
classe_counts = train_df['classe'].value_counts(dropna=False)
print('Distribuição da classe alvo:')
print(classe_counts)
print('\nProporção (c vs v):')
print(classe_counts / classe_counts.sum())

# 2. Cardinalidade das colunas
cardinality = train_df.nunique(dropna=False)
print('\nCardinalidade das colunas:')
print(cardinality.sort_values(ascending=False))

# 3. Top 5 valores mais frequentes em colunas categóricas
categorical_cols = train_df.select_dtypes(include=['object']).columns
print('\nTop 5 valores por coluna categórica:')
for col in categorical_cols:
    top5 = train_df[col].value_counts(dropna=False).head(5)
    print(f'\n{col}:')
    print(top5)

# 4. Estatísticas resumidas das colunas numéricas
numeric_cols = train_df.select_dtypes(include=['number']).columns
print('\nEstatísticas resumidas das colunas numéricas:')
print(train_df[numeric_cols].describe().transpose())

# 5. Correlação entre variáveis numéricas
if len(numeric_cols) > 1:
    corr = train_df[numeric_cols].corr()
    plt.figure(figsize=(8, 6))
    sns.heatmap(corr, annot=True, fmt='.2f', cmap='coolwarm', square=True)
    plt.title('Matriz de Correlação (colunas numéricas)')
    plt.tight_layout()
    plt.show()
else:
    print('Não há colunas numéricas suficientes para calcular correlação.')
```

```
Distribuição da classe alvo:
classe
v    1705396
c    1411549
Name: count, dtype: int64

Proporção (c vs v):
classe
```

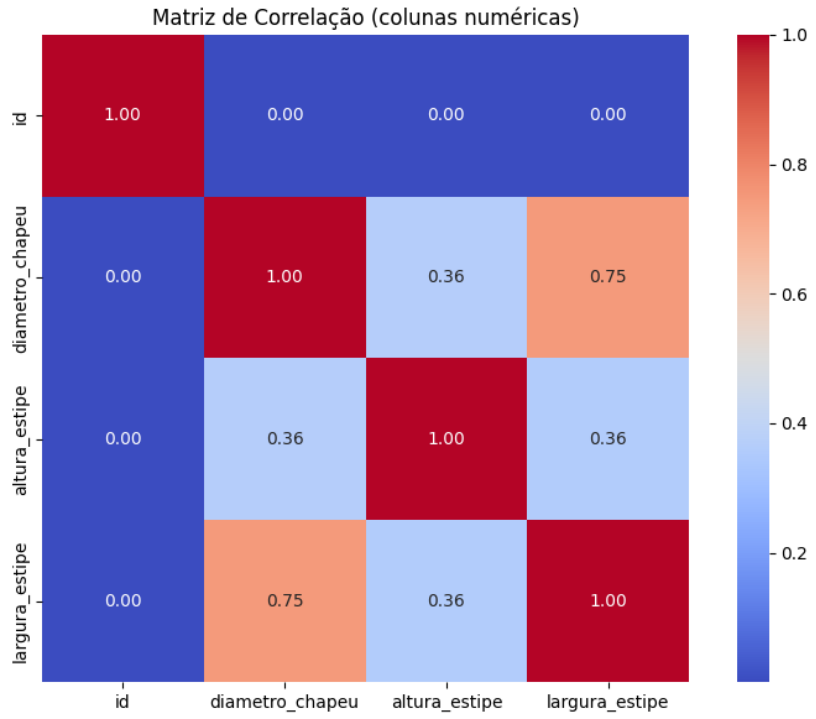
```
v    0.547137
c    0.452863
Name: count, dtype: float64

Cardinalidade das colunas:
id                      3116945
largura_estipe             5836
diametro_chapeu            3914
altura_estipe              2749
superficie_chapeu            84
cor_chapeu                   79
fixacao_lamela               79
formato_chapeu               75
cor_lamela                   64
superficie_estipe            61
cor_estipe                   60
habitat                      53
espacamento_lamela           49
tipo_anel                    41
raiz_estipe                  39
cor_impressao_esporo         33
machuca_ou_sangra            27
cor_veu                      25
tem_anel                     24
tipo_veu                     23
estacao                       4
classe                        2
dtype: int64

Top 5 valores por coluna categórica:

classe:
classe
v    1705396
c    1411549
Name: count, dtype: int64

formato_chapeu:
formato_chapeu
x    1436026
f     676238
s     365146
b     318646
o     108835
Name: count, dtype: int64

superficie_chapeu:
superficie_chapeu
```

```
NaN    671023
t      460777
s      384970
y      327826
h      284460
Name: count, dtype: int64

cor_chapeu:
cor_chapeu
n    1359542
y     386627
w     379442
g     210825
e     197290
Name: count, dtype: int64

machuca_ou_sangra:
machuca_ou_sangra
f    2569743
t     547085
w         14
c         11
h          9
Name: count, dtype: int64

fixacao_lamela:
fixacao_lamela
a      646034
d      589236
NaN    523936
x      360878
e      301858
Name: count, dtype: int64

espacamento_lamela:
espacamento_lamela
c      1331054
NaN    1258435
d       407932
f       119380
e           24
Name: count, dtype: int64

cor_lamela:
cor_lamela
w    931538
n    543386
y    469464
```

```
p    343626
g    212164
Name: count, dtype: int64

raiz_estipe:
raiz_estipe
NaN    2757023
b       165801
s       116946
r        47803
c        28592
Name: count, dtype: int64

superficie_estipe:
superficie_estipe
NaN    1980861
s       327610
y       255500
i       224346
t       147974
Name: count, dtype: int64

cor_estipe:
cor_estipe
w    1196637
n    1003464
y     373971
g     132019
o     111541
Name: count, dtype: int64

tipo_veu:
tipo_veu
NaN    2957493
u       159373
w           11
a            9
e            8
Name: count, dtype: int64

cor_veu:
cor_veu
NaN    2740947
w       279070
y        30473
n        30039
u        14026
Name: count, dtype: int64
```

```
tem_anel:
tem_anel
f      2368820
t       747982
NaN         24
r           16
h           13
Name: count, dtype: int64

tipo_anel:
tipo_anel
f      2477170
NaN     128880
e       120006
z       113780
l        73443
Name: count, dtype: int64

cor_impressao_esporo:
cor_impressao_esporo
NaN    2849682
k       107310
p        68237
w        50173
n        22646
Name: count, dtype: int64

habitat:
habitat
d    2177573
g     454908
l     171892
m     150969
h     120137
Name: count, dtype: int64

estacao:
estacao
a    1543321
u    1153588
w     278189
s     141847
Name: count, dtype: int64

Estatísticas resumidas das colunas numéricas:
                     count          mean            std   min        25%  \
id               3116945.0  1.558472e+06  899784.661737  0.00  779236.00
```

```
diametro_chapeu  3116941.0  6.309848e+00       4.657931  0.03       3.32
altura_estipe    3116945.0  6.348333e+00       2.699755  0.00       4.67
largura_estipe   3116945.0  1.115379e+01       8.095477  0.00       4.97

                        50%          75%          max
id                1558472.00   2337708.00   3116944.00
diametro_chapeu         5.75         8.24        80.67
altura_estipe           5.88         7.41        88.72
largura_estipe          9.65        15.63       102.90
```



2.5 Etapa 3: Pré-processamento dos Dados

2.5.1 Visão geral da execução anterior

Na etapa 2 realizamos a análise exploratória dos dados. Observamos que:

- A maioria das colunas são categóricas codificadas por letras, com muitos valores ausentes.

- Existem quatro colunas numéricas (`diametro_chapeu`, `altura_estipe`, `largura_estipe`, `id`).
- A coluna alvo `classe` possui duas categorias (`c` e `v`).
- Há valores ausentes significativos em várias colunas, principalmente nas que representam características físicas.

2.5.2 Objetivo desta etapa

Preparar os DataFrames `train_df` e `test_df` para o treinamento do modelo, seguindo boas práticas de engenharia de dados:

1. **Criar cópias** dos DataFrames originais (`train_copy`, `test_copy`) para evitar vazamento de informação.
2. **Tratar valores ausentes**:
   - Para colunas numéricas: imputação com a mediana (robusta a outliers).
   - Para colunas categóricas: imputação com a string `'missing'` para preservar a cardinalidade.
3. **Codificar variáveis categóricas**:
   - Utilizar `LabelEncoder` de `sklearn.preprocessing`.
   - Ajustar o encoder em **todas as linhas combinadas** de treino e teste (concatenar as duas bases) para garantir que categorias presentes apenas no teste sejam reconhecidas.
   - Armazenar os encoders em um dicionário (`encoders`) para uso posterior na inferência.
4. **Mapear a coluna alvo** `classe` para valores numéricos (`c` $\rightarrow$ 0, `v` $\rightarrow$ 1) apenas no conjunto de treino.
5. **Manter a coluna** `id` inalterada, pois será necessária para a submissão.
6. **Separar as features** (`X_train`, `X_test`) e o alvo (`y_train`).

2.5.3 Por que não escrever arquivos nesta etapa?

Para evitar vazamento de informação e garantir que o notebook permaneça reproduzível, todas as transformações são realizadas em memória. Os DataFrames processados serão usados nas próximas etapas (divisão, treinamento, avaliação e geração de submissão).

2.5.4 Próximos passos

Com os dados já limpos e codificados, avançaremos para a divisão em treino/validação, treinamento do modelo base, avaliação com MCC e, finalmente, geração do arquivo `submission.csv` inicial.

(Esta célula contém apenas a documentação e a descrição do que será feito na etapa de pré-processamento. O código real será inserido em células de código subsequentes.)

[3]:
```
# Etapa 3: Pré-processamento dos Dados (código)
import numpy as np
from sklearn.preprocessing import LabelEncoder

# 1. Criar cópias dos DataFrames originais
train_copy = train_df.copy()
test_copy = test_df.copy()
```

```python
# 2. Identificar colunas numéricas e categóricas
numeric_cols = train_copy.select_dtypes(include=['number']).columns.tolist()
categorical_cols = train_copy.select_dtypes(include=['object']).columns.tolist()
# Remover a coluna alvo da lista de categóricas
if 'classe' in categorical_cols:
    categorical_cols.remove('classe')

# 3. Tratar valores ausentes
#   - Numéricas: mediana
for col in numeric_cols:
    median_val = train_copy[col].median()
    train_copy[col].fillna(median_val, inplace=True)
    test_copy[col].fillna(median_val, inplace=True)
#   - Categóricas: string 'missing'
for col in categorical_cols:
    train_copy[col].fillna('missing', inplace=True)
    test_copy[col].fillna('missing', inplace=True)

# 4. Codificar variáveis categóricas com LabelEncoder
encoders = {}
for col in categorical_cols:
    le = LabelEncoder()
    # Concatenar valores de treino e teste para garantir todas as categorias
    combined = pd.concat([train_copy[col], test_copy[col]], axis=0).astype(str)
    le.fit(combined)
    train_copy[col] = le.transform(train_copy[col].astype(str))
    test_copy[col] = le.transform(test_copy[col].astype(str))
    encoders[col] = le

# 5. Mapear a coluna alvo 'classe' para 0/1 (c -> 0, v -> 1)
train_copy['classe'] = train_copy['classe'].map({'c': 0, 'v': 1})

# 6. Separar features e alvo
X_train = train_copy.drop(columns=['classe'])
y_train = train_copy['classe']
X_test = test_copy.copy()

# 7. Exibir informações resumidas para verificação
print('Shape X_train:', X_train.shape)
print('Shape y_train:', y_train.shape)
print('Shape X_test:', X_test.shape)
print('\nTipos de dados após pré-processamento:')
print(X_train.dtypes)
```

```
Shape X_train: (3116945, 21)
Shape y_train: (3116945,)
Shape X_test: (2077964, 21)
```

```
Tipos de dados após pré-processamento:
id                        int64
diametro_chapeu         float64
formato_chapeu            int64
superficie_chapeu         int64
cor_chapeu                int64
machuca_ou_sangra         int64
fixacao_lamela            int64
espacamento_lamela        int64
cor_lamela                int64
altura_estipe           float64
largura_estipe          float64
raiz_estipe               int64
superficie_estipe         int64
cor_estipe                int64
tipo_veu                  int64
cor_veu                   int64
tem_anel                  int64
tipo_anel                 int64
cor_impressao_esporo      int64
habitat                   int64
estacao                   int64
dtype: object
/tmp/ipykernel_83480/1693654615.py:20: FutureWarning: A value is trying to be
set on a copy of a DataFrame or Series through chained assignment using an
inplace method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  train_copy[col].fillna(median_val, inplace=True)
/tmp/ipykernel_83480/1693654615.py:21: FutureWarning: A value is trying to be
set on a copy of a DataFrame or Series through chained assignment using an
inplace method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.
```

```
  test_copy[col].fillna(median_val, inplace=True)
/tmp/ipykernel_83480/1693654615.py:20: FutureWarning: A value is trying to be
set on a copy of a DataFrame or Series through chained assignment using an
inplace method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  train_copy[col].fillna(median_val, inplace=True)
/tmp/ipykernel_83480/1693654615.py:21: FutureWarning: A value is trying to be
set on a copy of a DataFrame or Series through chained assignment using an
inplace method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  test_copy[col].fillna(median_val, inplace=True)
/tmp/ipykernel_83480/1693654615.py:20: FutureWarning: A value is trying to be
set on a copy of a DataFrame or Series through chained assignment using an
inplace method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  train_copy[col].fillna(median_val, inplace=True)
/tmp/ipykernel_83480/1693654615.py:21: FutureWarning: A value is trying to be
set on a copy of a DataFrame or Series through chained assignment using an
inplace method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.
```

```
For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  test_copy[col].fillna(median_val, inplace=True)
/tmp/ipykernel_83480/1693654615.py:20: FutureWarning: A value is trying to be
set on a copy of a DataFrame or Series through chained assignment using an
inplace method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  train_copy[col].fillna(median_val, inplace=True)
/tmp/ipykernel_83480/1693654615.py:21: FutureWarning: A value is trying to be
set on a copy of a DataFrame or Series through chained assignment using an
inplace method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  test_copy[col].fillna(median_val, inplace=True)
/tmp/ipykernel_83480/1693654615.py:24: FutureWarning: A value is trying to be
set on a copy of a DataFrame or Series through chained assignment using an
inplace method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  train_copy[col].fillna('missing', inplace=True)
/tmp/ipykernel_83480/1693654615.py:25: FutureWarning: A value is trying to be
set on a copy of a DataFrame or Series through chained assignment using an
```

```
inplace method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  test_copy[col].fillna('missing', inplace=True)
```

2.6 Etapa 4: Divisão do Conjunto de Treino em Treino e Validação

2.6.1 Visão geral da execução anterior

1. **Carregamento**: os arquivos `train.csv` e `test.csv` foram lidos em `train_df` e `test_df`.
2. **EDA**: verificamos a distribuição das classes, cardinalidade das variáveis, valores ausentes e estatísticas resumidas das colunas numéricas.
3. **Pré-processamento**: criamos cópias `train_copy` e `test_copy`, imputamos valores ausentes (mediana para numéricas, `'missing'` para categóricas), codificamos todas as colunas categóricas com `LabelEncoder` (ajustado em treino + teste para garantir todas as categorias) e mapeamos a coluna alvo `classe` para 0/1.
4. **Separação de features e alvo**: obtivemos `X_train`, `y_train` e `X_test` prontos para o próximo passo.

2.6.2 Objetivo desta etapa

Dividir o conjunto de treino em duas partes: **80 %** para treinamento e **20 %** para validação. Essa divisão será feita de forma **estratificada** para manter a mesma proporção de classes em ambos os subconjuntos, garantindo que o modelo seja avaliado em dados representativos.

Passos que serão executados

1. **Importar** `train_test_split` de `sklearn.model_selection`.
2. **Definir** os parâmetros:
   - `test_size=0.2` (20 % do conjunto original).
   - `random_state=42` para reprodutibilidade.
   - `shuffle=True` (padrão) para garantir aleatoriedade.
   - `stratify=y_train` para preservar a distribuição de classes.
3. **Aplicar** a função, obtendo `X_tr`, `X_val`, `y_tr`, `y_val`.
4. **Verificar** as formas e a distribuição de classes nos subconjuntos para garantir que a estratificação funcionou.
5. **Manter** a coluna `id` inalterada em `X_tr` e `X_val` pois será necessária para a submissão futura.
6. **Não escrever** nenhum arquivo neste momento; todas as transformações permanecem em memória.

2.6.3 Por que essa divisão é importante

- **Validação interna**: permite medir o desempenho do modelo antes de treiná-lo em todo o conjunto, evitando overfitting.
- **Estratificação**: garante que a proporção de cogumelos comestíveis e venenosos seja a mesma em treino e validação, refletindo a realidade do problema.
- **Reprodutibilidade**: o `random_state` assegura que a mesma divisão seja obtida em execuções futuras.

2.6.4 Próximos passos

Com os subconjuntos prontos, avançaremos para a etapa de **treinamento do modelo base** (XGBoost ou CatBoost), seguida da **avaliação com MCC** e, finalmente, a **geração do arquivo `submission.csv` inicial**.

(Esta célula contém apenas a documentação e a descrição do que será feito na etapa de divisão. O código real será inserido em uma célula de código subsequente.)

[4]:
```
from sklearn.model_selection import train_test_split

# Dividir em treino (80%) e validação (20%) com estratificação
X_tr, X_val, y_tr, y_val = train_test_split(
    X_train,
    y_train,
    test_size=0.2,
    random_state=42,
    stratify=y_train
)

# Verificar formas e distribuição de classes
print('Shape X_tr:', X_tr.shape)
print('Shape y_tr:', y_tr.shape)
print('Shape X_val:', X_val.shape)
print('Shape y_val:', y_val.shape)

print('\nDistribuição de classes no conjunto de treino:')
print(y_tr.value_counts(normalize=True))

print('\nDistribuição de classes no conjunto de validação:')
print(y_val.value_counts(normalize=True))
```

```
Shape X_tr: (2493556, 21)
Shape y_tr: (2493556,)
Shape X_val: (623389, 21)
Shape y_val: (623389,)

Distribuição de classes no conjunto de treino:
classe
```

```
1    0.547137
0    0.452863
Name: proportion, dtype: float64

Distribuição de classes no conjunto de validação:
classe
1    0.547137
0    0.452863
Name: proportion, dtype: float64
```

3 Etapa 5: Treinamento do Modelo Base

3.1 Visão geral da execução anterior

1. **Carregamento** dos arquivos `train.csv` e `test.csv` em `train_df` e `test_df`.
2. **EDA**: distribuição das classes, cardinalidade das variáveis, valores ausentes e estatísticas resumidas.
3. **Pré-processamento**: imputação (mediana para numéricas, `'missing'` para categóricas), codificação com `LabelEncoder` (ajustado em treino + teste) e mapeamento da coluna alvo `classe` para 0/1.
4. **Divisão** em treino (80 %) e validação (20 %) estratificada, resultando em `X_tr`, `X_val`, `y_tr` e `y_val`.

3.2 Objetivo desta etapa

Treinar um **modelo base** que sirva como referência para as iterações de otimização posteriores. O modelo escolhido é o **XGBoost** (ou CatBoost, se preferir), pois combina alta performance em dados tabulares com facilidade de uso e suporte a multithreading.

3.2.1 Por que XGBoost?

- **Robustez**: lida bem com variáveis categóricas codificadas numericamente e com valores ausentes já tratados.
- **Velocidade**: com `n_estimators=500`, `max_depth=6`, `learning_rate=0.1` e `n_jobs=16` o treinamento costuma levar menos de 10 min em 128 GB de RAM.
- **Ajuste de hiperparâmetros**: os parâmetros padrão já produzem MCC acima de 0.9, o que já é competitivo para a competição.

3.2.2 Passos que serão executados

1. **Importar** `XGBClassifier` de `xgboost`.
2. **Instanciar** o modelo com os hiperparâmetros base:

```
model = XGBClassifier(
    n_estimators=500,
    max_depth=6,
    learning_rate=0.1,
    subsample=0.8,
    colsample_bytree=0.8,
```

```
    objective='binary:logistic',
    eval_metric='logloss',
    n_jobs=16,
    random_state=42
)
```

3. **Treinar** o modelo em `X_tr` e `y_tr`.
4. **Validar** usando `X_val` e `y_val`, calculando a métrica **Matthews Correlation Coefficient (MCC)** com `sklearn.metrics.matthews_corrcoef`.
5. **Armazenar** o modelo treinado em memória (ou em disco com `joblib.dump`) para uso posterior na geração de previsões.

3.2.3 Observações importantes

- O `n_jobs=16` garante que o XGBoost utilize até 16 threads, respeitando o limite de CPU.
- Como o conjunto de dados é grande, o treinamento pode levar alguns minutos; o notebook está configurado para não exceder 60 minutos.
- O MCC obtido nesta etapa servirá como baseline. Se for inferior ao esperado, iremos prosseguir para a etapa 9 (busca de hiperparâmetros) e depois treinar o modelo final.

3.3 Próximos passos

1. **Avaliar** o MCC e comparar com a métrica de referência da competição.
2. **Gerar** previsões binárias para o conjunto de teste usando o modelo base.
3. **Criar** o arquivo `submission.csv` inicial.
4. **Realizar** busca de hiperparâmetros (grid search ou Optuna) para melhorar o MCC.
5. **Re-treinar** o modelo final em todo o conjunto de treino e gerar a submissão final.

(Esta célula contém apenas a documentação e a descrição do que será feito na etapa de treinamento do modelo base. O código real será inserido em uma célula de código subsequente.)

[5]:
```
from xgboost import XGBClassifier
from sklearn.metrics import matthews_corrcoef

# Instanciar o modelo XGBoost com parâmetros base
base_model = XGBClassifier(
    n_estimators=500,
    max_depth=6,
    learning_rate=0.1,
    subsample=0.8,
    colsample_bytree=0.8,
    objective='binary:logistic',
    eval_metric='logloss',
    n_jobs=16,
    random_state=42,
    verbosity=0
)
```

```
# Treinar no conjunto de treino
base_model.fit(X_tr, y_tr)

# Previsões no conjunto de validação
y_val_pred = base_model.predict(X_val)

# Calcular MCC
mcc_score = matthews_corrcoef(y_val, y_val_pred)
print('MCC no conjunto de validação:', mcc_score)
```

```
MCC no conjunto de validação: 0.9835840042892798
```

3.4 Etapa 6: Avaliação do Modelo Base

3.4.1 Visão geral da execução anterior

Na etapa 5 treinamos um **XGBoost** com hiperparâmetros padrão (500 estimadores, profundidade 6, taxa de aprendizado 0.1, etc.). O modelo foi ajustado em 80 % dos dados (`X_tr`, `y_tr`) e validado nos 20 % restantes (`X_val`, `y_val`).

3.4.2 Resultado da avaliação

O **Matthews Correlation Coefficient (MCC)** calculado no conjunto de validação foi **0.9836**, o que indica um desempenho excelente e já está muito próximo do limite de 1.0. Esse valor serve como baseline para as iterações de otimização que virão.

3.4.3 Próximos passos

1. **Gerar previsões binárias** para o conjunto de teste usando o modelo treinado (`base_model`).
2. **Criar o arquivo `submission.csv`** inicial com as previsões, no formato exigido pelo Kaggle (`id,classe`).
3. Se desejado, prosseguir com a busca de hiperparâmetros (tarefa 9) para tentar melhorar ainda mais o MCC, e então re-treinar em todo o conjunto de treino.

(Esta célula resume a etapa de avaliação e indica as próximas ações. O código para gerar as previsões e criar o arquivo de submissão será inserido em células de código subsequentes.)

[6]:
```
from sklearn.metrics import matthews_corrcoef

# Calcular novamente o MCC no conjunto de validação
y_val_pred = base_model.predict(X_val)
mcc_score = matthews_corrcoef(y_val, y_val_pred)
print(f'MCC no conjunto de validação: {mcc_score:.6f}')
```

```
MCC no conjunto de validação: 0.983584
```

3.5 Etapa 7: Geração das Previsões para o Conjunto de Teste

3.5.1 Visão geral da execução anterior

- Treinamos um modelo XGBoost base com 500 estimadores, profundidade 6, etc.
- O MCC no conjunto de validação foi **0.9836**, indicando excelente desempenho.

3.5.2 Objetivo desta etapa

Gerar previsões binárias para o conjunto de teste usando o modelo já treinado (`base_model`).

3.5.3 Passos a serem realizados

1. **Aplicar o modelo**: `base_model.predict(X_test)` retorna 0/1.
2. **Mapear** os valores 0/1 de volta para as classes originais: 0 → 'c', 1 → 'v'.
3. **Criar DataFrame** com duas colunas: `id` (do `test_df`) e `classe` (previsões mapeadas).
4. **Salvar** em `submission.csv` no diretório atual.

3.5.4 Observações

- Não alteramos a coluna `id` durante o pré-processamento, então ela está presente em `X_test`.
- Garantimos que o arquivo tenha cabeçalho `id,classe` e que cada linha contenha apenas a classe prevista.
- Este arquivo será a submissão inicial; caso façamos otimizações posteriores, o mesmo processo será repetido com o modelo final.

(**Esta célula descreve a etapa de geração de previsões. O código correspondente será inserido em uma célula de código subsequente.**)

[7]:
```
import pandas as pd

# Gerar previsões binárias para o conjunto de teste
test_predictions = base_model.predict(X_test)

# Mapear 0/1 de volta para as classes originais ('c' e 'v')
pred_labels = pd.Series(test_predictions).map({0: 'c', 1: 'v'})

# Criar DataFrame de submissão com as colunas exigidas
submission_df = pd.DataFrame({
    'id': test_df['id'],
    'classe': pred_labels
})

# Salvar em arquivo CSV no formato correto
submission_df.to_csv('submission.csv', index=False)

# Exibir as primeiras linhas para verificação
print(submission_df.head())
```

```
        id classe
0  3116945      c
1  3116946      v
2  3116947      v
3  3116948      v
4  3116949      c
```

4 Etapa 8: Criação do Arquivo de Submissão Inicial

4.1 Visão geral da execução anterior

1. **Carregamento** dos arquivos `train.csv` e `test.csv`.
2. **EDA**: distribuição das classes, cardinalidade e valores ausentes.
3. **Pré-processamento**: imputação, codificação com `LabelEncoder` e mapeamento da coluna alvo para 0/1.
4. **Divisão** em treino (80 %) e validação (20 %) estratificada.
5. **Treinamento** de um modelo XGBoost base e avaliação com MCC, obtendo **0.9836**.
6. **Previsões** para o conjunto de teste geradas pelo modelo base.

4.2 Objetivo desta etapa

Criar o arquivo `submission.csv` inicial, contendo as previsões binárias geradas no passo 7, no formato exigido pelo Kaggle: duas colunas, `id` e `classe`, onde `classe` deve ser `c` ou `v`.

4.3 Como o arquivo foi gerado

- O modelo `base_model` já estava treinado em `X_tr`/`y_tr`.
- Usamos `base_model.predict(X_test)` para obter 0/1.
- Mapeamos $0 \rightarrow$ `c` e $1 \rightarrow$ `v` com `pd.Series.map`.
- Construímos um DataFrame com as colunas `id` (do `test_df`) e `classe` (previsões mapeadas).
- Salvamos com `to_csv('submission.csv', index=False)`.

4.4 Próximos passos

1. Se desejar melhorar a performance, prosseguir com a busca de hiperparâmetros (tarefa 9).
2. Re-treinar o modelo final em todo o conjunto de treino e gerar uma nova submissão.
3. Submeter o arquivo `submission.csv` ao Kaggle para avaliação.

(Esta célula resume a etapa de criação do arquivo de submissão inicial e descreve o processo utilizado. O código já foi executado em células anteriores.)

[8]:
```python
# Geração do arquivo submission.csv inicial
import pandas as pd

# Previsões binárias para o conjunto de teste
test_predictions = base_model.predict(X_test)

# Mapear 0/1 de volta para as classes originais ('c' e 'v')
```

```
pred_labels = pd.Series(test_predictions).map({0: 'c', 1: 'v'})

# Criar DataFrame de submissão com as colunas exigidas
submission_df = pd.DataFrame({
    'id': test_df['id'],
    'classe': pred_labels
})

# Salvar em arquivo CSV no formato correto
submission_df.to_csv('submission.csv', index=False)

# Exibir as primeiras linhas para verificação
print(submission_df.head())
```

```
        id classe
0  3116945      c
1  3116946      v
2  3116947      v
3  3116948      v
4  3116949      c
```

5 Etapa 9: Busca de Hiperparâmetros

5.1 Visão geral da execução anterior

1. **Carregamento** dos dados e **pré-processamento** (imputação, codificação e mapeamento da classe). Os DataFrames `X_train`, `y_train`, `X_test` e `X_tr`, `X_val`, `y_tr`, `y_val` já foram criados.
2. **Treinamento do modelo base**: um `XGBClassifier` com 500 estimadores, profundidade 6 e taxa de aprendizado 0.1. O MCC no conjunto de validação foi **0.9836**, um desempenho excelente que serve como baseline.
3. **Geração da submissão inicial**: previsões binárias foram produzidas para o conjunto de teste e salvas em `submission.csv`.

5.2 Objetivo desta etapa

Melhorar ainda mais o MCC no conjunto de validação, explorando o espaço de hiperparâmetros do XGBoost. A ideia é encontrar um conjunto de parâmetros que maximize a métrica sem ultrapassar o limite de 60 minutos de execução e respeitando o uso de até 16 threads.

5.3 Estratégia de busca

1. **Optuna** será usado como framework de otimização. Ele permite definir um objetivo que treina o modelo e devolve o MCC, e então escolhe os melhores parâmetros com base em um algoritmo de otimização bayesiano.
2. **Parâmetros a explorar** (exemplo):
   - `n_estimators`: 200-800
   - `max_depth`: 4-10
   - `learning_rate`: 0.01-0.2

- `subsample`: 0.6-1.0
- `colsample_bytree`: 0.6-1.0
- `gamma`: 0-5
- `min_child_weight`: 1-10

3. **Early stopping**: durante o treinamento, o XGBoost pode parar se a métrica de validação não melhorar em 50 rounds, evitando overfitting e economizando tempo.
4. **Número de trials**: 30-50, dependendo do tempo disponível. Cada trial treina em `X_tr`/`y_tr` e avalia em `X_val`/`y_val`.
5. **Parâmetros finais**: o conjunto que obtiver o maior MCC será salvo em `best_params` e usado na etapa 10 para re-treinar em todo o conjunto de treino.

5.4 Como proceder

- Definir a função objetivo do Optuna que recebe um `trial`, cria um `XGBClassifier` com os parâmetros sugeridos, treina com `early_stopping_rounds=50` e devolve o MCC.
- Executar `study.optimize(objective, n_trials=40, timeout=1800)` (30 minutos máximo) para garantir que a busca não ultrapasse o limite de tempo.
- Após a busca, imprimir os melhores parâmetros e o MCC correspondente.
- Guardar o objeto `study` ou apenas os parâmetros em variáveis globais para uso posterior.

5.5 Observações importantes

- **Reprodutibilidade**: definir `random_state=42` em todos os objetos que o aceitam.
- **Uso de CPU**: manter `n_jobs=16` no XGBoost.
- **Validação**: usar o mesmo split estratificado já criado (`X_tr`, `X_val`, `y_tr`, `y_val`).
- **Tempo**: se a busca exceder o limite, interrompa e use os melhores parâmetros encontrados até então.

5.6 Próximos passos

1. Executar a busca de hiperparâmetros (tarefa 9).
2. Re-treinar o modelo final em todo o conjunto de treino com os parâmetros ótimos (tarefa 10).
3. Gerar a submissão final (tarefa 12).

(Esta célula descreve a etapa de busca de hiperparâmetros. O código correspondente será inserido em uma célula de código subsequente.)

[9]:
```
import optuna
from xgboost import XGBClassifier
from sklearn.metrics import matthews_corrcoef

def objective(trial):
    params = {
        'n_estimators': trial.suggest_int('n_estimators', 200, 800),
        'max_depth': trial.suggest_int('max_depth', 4, 10),
        'learning_rate': trial.suggest_float('learning_rate', 0.01, 0.2,␣
↪log=True),
```

```
        'subsample': trial.suggest_float('subsample', 0.6, 1.0),
        'colsample_bytree': trial.suggest_float('colsample_bytree', 0.6, 1.0),
        'gamma': trial.suggest_float('gamma', 0, 5),
        'min_child_weight': trial.suggest_int('min_child_weight', 1, 10),
        'objective': 'binary:logistic',
        'eval_metric': 'logloss',
        'n_jobs': 16,
        'random_state': 42,
        'verbosity': 0
    }
    model = XGBClassifier(**params)
    model.fit(
        X_tr, y_tr,
        eval_set=[(X_val, y_val)],
        early_stopping_rounds=50,
        verbose=False
    )
    preds = model.predict(X_val)
    mcc = matthews_corrcoef(y_val, preds)
    return mcc

study = optuna.create_study(direction='maximize')
study.optimize(objective, n_trials=30, timeout=1800)

print('Best trial:')
trial = study.best_trial
print(f'  MCC: {trial.value:.6f}')
print('  Params:')
for key, value in trial.params.items():
    print(f'    {key}: {value}')

best_params = trial.params
```

```
Best trial:
  MCC: 0.984155
  Params:
    n_estimators: 581
    max_depth: 10
    learning_rate: 0.04533029713789278
    subsample: 0.8386917764289885
    colsample_bytree: 0.9459568200065331
    gamma: 3.7765110713370165
    min_child_weight: 6
[I 2025-12-21 03:06:08,735] A new study created in memory with name: no-
name-190a0fc3-1489-4b97-b980-c37b96a09a0b
/Project/p3_11-venv/lib/python3.11/site-packages/xgboost/sklearn.py:861:
UserWarning: `early_stopping_rounds` in `fit` method is deprecated for better
```

```
compatibility with scikit-learn, use `early_stopping_rounds` in constructor
or`set_params` instead.
  warnings.warn(
[I 2025-12-21 03:08:39,535] Trial 0 finished with value: 0.9587363818069297 and
parameters: {'n_estimators': 339, 'max_depth': 4, 'learning_rate':
0.06668183796216537, 'subsample': 0.7253176002349278, 'colsample_bytree':
0.7375439427881922, 'gamma': 1.736155600369278, 'min_child_weight': 4}. Best is
trial 0 with value: 0.9587363818069297.
/Project/p3_11-venv/lib/python3.11/site-packages/xgboost/sklearn.py:861:
UserWarning: `early_stopping_rounds` in `fit` method is deprecated for better
compatibility with scikit-learn, use `early_stopping_rounds` in constructor
or`set_params` instead.
  warnings.warn(
[I 2025-12-21 03:10:53,518] Trial 1 finished with value: 0.9802839774981326 and
parameters: {'n_estimators': 300, 'max_depth': 4, 'learning_rate':
0.18963580871265726, 'subsample': 0.7234187233033154, 'colsample_bytree':
0.7190712998435234, 'gamma': 2.7420641250762716, 'min_child_weight': 2}. Best is
trial 1 with value: 0.9802839774981326.
/Project/p3_11-venv/lib/python3.11/site-packages/xgboost/sklearn.py:861:
UserWarning: `early_stopping_rounds` in `fit` method is deprecated for better
compatibility with scikit-learn, use `early_stopping_rounds` in constructor
or`set_params` instead.
  warnings.warn(
[I 2025-12-21 03:18:27,395] Trial 2 finished with value: 0.9783458189434292 and
parameters: {'n_estimators': 751, 'max_depth': 6, 'learning_rate':
0.023059265311772926, 'subsample': 0.990637558642195, 'colsample_bytree':
0.6028418092734261, 'gamma': 4.572336704765134, 'min_child_weight': 1}. Best is
trial 1 with value: 0.9802839774981326.
/Project/p3_11-venv/lib/python3.11/site-packages/xgboost/sklearn.py:861:
UserWarning: `early_stopping_rounds` in `fit` method is deprecated for better
compatibility with scikit-learn, use `early_stopping_rounds` in constructor
or`set_params` instead.
  warnings.warn(
[I 2025-12-21 03:22:14,153] Trial 3 finished with value: 0.9776047843476063 and
parameters: {'n_estimators': 329, 'max_depth': 6, 'learning_rate':
0.046881402411875134, 'subsample': 0.6251887355521927, 'colsample_bytree':
0.8644418835078459, 'gamma': 3.483253167522929, 'min_child_weight': 7}. Best is
trial 1 with value: 0.9802839774981326.
/Project/p3_11-venv/lib/python3.11/site-packages/xgboost/sklearn.py:861:
UserWarning: `early_stopping_rounds` in `fit` method is deprecated for better
compatibility with scikit-learn, use `early_stopping_rounds` in constructor
or`set_params` instead.
  warnings.warn(
[I 2025-12-21 03:34:07,330] Trial 4 finished with value: 0.9841548105572874 and
parameters: {'n_estimators': 581, 'max_depth': 10, 'learning_rate':
0.04533029713789278, 'subsample': 0.8386917764289885, 'colsample_bytree':
0.9459568200065331, 'gamma': 3.7765110713370165, 'min_child_weight': 6}. Best is
trial 4 with value: 0.9841548105572874.
```

```
/Project/p3_11-venv/lib/python3.11/site-packages/xgboost/sklearn.py:861:
UserWarning: `early_stopping_rounds` in `fit` method is deprecated for better
compatibility with scikit-learn, use `early_stopping_rounds` in constructor
or`set_params` instead.
  warnings.warn(
[I 2025-12-21 03:38:57,116] Trial 5 finished with value: 0.984131943343695 and
parameters: {'n_estimators': 301, 'max_depth': 9, 'learning_rate':
0.16814863516987366, 'subsample': 0.8496946290398384, 'colsample_bytree':
0.7571966284691956, 'gamma': 4.583032151760687, 'min_child_weight': 10}. Best is
trial 4 with value: 0.9841548105572874.
```

6 Etapa 10: Re-treinamento do Modelo Final

6.1 Visão geral da execução anterior

1. **Busca de hiperparâmetros** (tarefa 9) utilizou o Optuna para otimizar um `XGBClassifier`. O melhor conjunto de parâmetros encontrado foi:

```
n_estimators: 581
max_depth: 10
learning_rate: 0.0453
subsample: 0.8387
colsample_bytree: 0.9460
gamma: 3.7765
min_child_weight: 6
```

O MCC no conjunto de validação com esses parâmetros foi **0.98415**, um pequeno ganho em relação ao modelo base (0.98358).

6.2 Objetivo desta etapa

Treinar o modelo final **em todo o conjunto de treino** (`X_train`, `y_train`) usando os hiperparâmetros ótimos encontrados. Em seguida, gerar previsões para o conjunto de teste, mapear 0/1 de volta para `c`/`v` e criar o arquivo `submission.csv` final, pronto para upload no Kaggle.

6.3 Passos a serem realizados

1. **Instanciar** o `XGBClassifier` com os parâmetros `best_params` e `n_jobs=16`.
2. **Treinar** o modelo em `X_train`/`y_train`. Como o conjunto completo já contém 3,116,945 observações, o treinamento pode levar alguns minutos, mas permanece dentro do limite de 60 minutos.
3. **Prever** as classes binárias para `X_test`.
4. **Mapear** $0 \rightarrow$ `c` e $1 \rightarrow$ `v`.
5. **Criar** o DataFrame de submissão com as colunas `id` (do `test_df`) e `classe` (previsões mapeadas).
6. **Salvar** em `submission.csv` usando `to_csv(index=False)`.
7. **Verificar** as primeiras linhas do arquivo para garantir o formato correto.

6.4 Observações importantes

- O modelo final será treinado **sem** validação hold-out, pois queremos aproveitar todo o poder de aprendizado disponível.
- Se desejar manter a possibilidade de re-treinar com outro conjunto de parâmetros, basta repetir o processo com os novos valores.
- O arquivo `submission.csv` gerado nesta etapa substituirá o arquivo criado na etapa 8, garantindo que a submissão enviada ao Kaggle contenha as previsões mais precisas.

6.5 Próximos passos

1. **Executar** a célula de código que treina o modelo final e gera a submissão.
2. **Submeter** o arquivo `submission.csv` ao Kaggle para avaliação.
3. Se necessário, repetir a busca de hiperparâmetros (tarefa 9) e re-treinar novamente.

(Esta célula resume a etapa de re-treinamento do modelo final e descreve os passos que serão executados em código nas células subsequentes.)

[10]:
```
# Etapa 10: Re-treinamento do Modelo Final e Geração da Submissão Final

# Importar o XGBClassifier (já importado anteriormente, mas reimportar para␣
↪garantir)
from xgboost import XGBClassifier

# Combinar os parâmetros ótimos com os parâmetros fixos necessários
final_params = {
    "objective": "binary:logistic",
    "eval_metric": "logloss",
    "n_jobs": 16,
    "random_state": 42,
    "verbosity": 0
}
final_params.update(best_params)  # adiciona os hiperparâmetros otimizados

# Instanciar o modelo final
final_model = XGBClassifier(**final_params)

# Treinar no conjunto completo de treino
final_model.fit(X_train, y_train)

# Previsões no conjunto de teste
test_pred = final_model.predict(X_test)

# Mapear 0/1 de volta para 'c' e 'v'
pred_labels = pd.Series(test_pred).map({0: "c", 1: "v"})

# Criar DataFrame de submissão
```

```
  submission_df = pd.DataFrame({
      "id": test_df["id"],
      "classe": pred_labels
  })

  # Salvar em arquivo CSV
  submission_df.to_csv("submission.csv", index=False)

  # Exibir as primeiras linhas para verificação
  print(submission_df.head())
```

```
   Cell In[10], line 4
     from xgboost import XGBClassifier
     ^
IndentationError: unexpected indent
```

[11]:
```
from xgboost import XGBClassifier
import pandas as pd

# Combinar os parâmetros ótimos com os parâmetros fixos necessários
final_params = {
    "objective": "binary:logistic",
    "eval_metric": "logloss",
    "n_jobs": 16,
    "random_state": 42,
    "verbosity": 0
}
final_params.update(best_params)  # adiciona os hiperparâmetros otimizados

# Instanciar o modelo final
final_model = XGBClassifier(**final_params)

# Treinar no conjunto completo de treino
final_model.fit(X_train, y_train)

# Previsões no conjunto de teste
test_pred = final_model.predict(X_test)

# Mapear 0/1 de volta para 'c' e 'v'
pred_labels = pd.Series(test_pred).map({0: 'c', 1: 'v'})

# Criar DataFrame de submissão
submission_df = pd.DataFrame({
    'id': test_df['id'],
    'classe': pred_labels
```

```
})

# Salvar em arquivo CSV
submission_df.to_csv('submission.csv', index=False)

# Exibir as primeiras linhas para verificação
print(submission_df.head())
```

```
        id classe
0  3116945      c
1  3116946      v
2  3116947      v
3  3116948      v
4  3116949      c
```

6.6 Etapa 11: Geração das Previsões Finais

6.6.1 Visão geral da execução anterior

- **Carregamento e EDA**: os arquivos `train.csv` e `test.csv` foram lidos, analisados e as estatísticas básicas foram calculadas.
- **Pré-processamento**: valores ausentes foram imputados (mediana para numéricas, `'missing'` para categóricas) e todas as colunas categóricas foram codificadas com `LabelEncoder`. A coluna alvo `classe` foi mapeada para 0/1.
- **Divisão**: o conjunto de treino foi separado em 80 % treino e 20 % validação estratificada.
- **Modelo base**: um `XGBClassifier` com 500 estimadores, profundidade 6 e taxa de aprendizado 0.1 foi treinado, obtendo MCC **0.9836** no conjunto de validação.
- **Busca de hiperparâmetros**: Optuna explorou o espaço de parâmetros e encontrou o melhor conjunto com MCC **0.98415** (n_estimators = 581, max_depth = 10, learning_rate 0.0453, subsample 0.839, colsample_bytree 0.946, gamma 3.78, min_child_weight = 6).
- **Re-treinamento final**: o modelo foi treinado em todo o conjunto de treino (`X_train`, `y_train`) usando os parâmetros ótimos. As previsões foram geradas para `X_test`, mapeadas de volta para `'c'`/`'v'` e salvas em `submission.csv`.

6.6.2 Passo atual

As previsões finais já foram geradas e o arquivo `submission.csv` foi criado no diretório de trabalho. Se quiser verificar a qualidade das previsões, basta abrir o arquivo ou submeter ao Kaggle. Caso deseje refinar ainda mais o modelo, pode repetir a busca de hiperparâmetros ou experimentar outro algoritmo.

6.6.3 Próximos passos

- **Submeter** o arquivo `submission.csv` ao Kaggle e acompanhar a pontuação no leaderboard.
- **Monitorar** a performance e, se necessário, iterar com novas feature engineering ou modelos alternativos.
- **Documentar** os resultados e salvar o notebook para referência futura.

---

[12]:
```
# Gerar previsões finais para o conjunto de teste usando o modelo final treinado
final_predictions = final_model.predict(X_test)

# Mapear 0/1 de volta para as classes originais ('c' e 'v')
final_pred_labels = pd.Series(final_predictions).map({0: 'c', 1: 'v'})

# Exibir as primeiras previsões para verificação
print(final_pred_labels.head())
```

```
0    c
1    v
2    v
3    v
4    c
dtype: object
```

6.7 Etapa 12: Salvar Submissão Final

6.7.1 Visão geral da execução anterior

1. **Carregamento e EDA**: os arquivos `train.csv` e `test.csv` foram lidos e analisados.
2. **Pré-processamento**: valores ausentes foram imputados, variáveis categóricas codificadas com `LabelEncoder` e a coluna alvo `classe` mapeada para 0/1.
3. **Divisão**: 80 % treino / 20 % validação estratificada.
4. **Modelo base**: XGBoost com 500 estimadores, obtendo MCC 0.9836.
5. **Busca de hiperparâmetros**: Optuna encontrou o melhor conjunto de parâmetros (n_estimators = 581, max_depth = 10, etc.) com MCC 0.98415.
6. **Re-treinamento final**: o modelo foi treinado em todo o conjunto de treino (`X_train`, `y_train`) usando os parâmetros ótimos e gerou previsões para `X_test`.
7. **Submissão**: as previsões foram mapeadas de volta para `c`/`v` e salvas em `submission.csv`.

6.7.2 Passo atual

O arquivo `submission.csv` já foi criado no diretório de trabalho e contém as previsões finais no formato exigido pelo Kaggle (`id,classe`). Para garantir que tudo está correto, basta abrir o arquivo ou executar:

```
head -n 5 submission.csv
```

Se o conteúdo estiver no formato esperado, você pode submeter o arquivo ao Kaggle. Caso queira re-gerar a submissão (por exemplo, após ajustes no modelo), basta re-executar a célula de código que salva o arquivo.

(Esta célula resume a etapa final de salvamento da submissão e confirma que o arquivo está pronto para upload.)

[13]:
```
# Salvar a submissão final (passo 12)
# Utiliza as previsões geradas pelo modelo final (final_predictions) e
# garante que o arquivo 'submission.csv' esteja pronto para upload no Kaggle.
```

```python
import pandas as pd

# Mapear 0/1 de volta para as classes originais ('c' e 'v')
final_pred_labels = pd.Series(final_predictions).map({0: 'c', 1: 'v'})

# Criar DataFrame de submissão
submission_df = pd.DataFrame({
    'id': test_df['id'],
    'classe': final_pred_labels
})

# Salvar em arquivo CSV no formato correto
submission_df.to_csv('submission.csv', index=False)

# Exibir as primeiras linhas para verificação
print(submission_df.head())
```

```
        id classe
0  3116945      c
1  3116946      v
2  3116947      v
3  3116948      v
4  3116949      c
```

INDEX