



---


# CliMB: An AI-enabled Partner for Clinical Predictive Modeling


---

**Evgeny Saveliev\***   
University of Cambridge

**Tim Schubert\***   
University of Cambridge  
University Hospital Heidelberg

**Thomas Pouplin**  
University of Cambridge

**Vasilis Kosmoliaptis**   
University of Cambridge  
NIHR BTRU, Cambridge

**Mihaela van der Schaar**   
University of Cambridge  
mv472@damtp.cam.ac.uk

## Abstract

Despite its significant promise and continuous technical advances, real-world applications of artificial intelligence (AI) remain limited. We attribute this to the “domain expert-AI-conundrum”: while domain experts, such as clinician scientists, should be able to build predictive models such as risk scores, they face substantial barriers in accessing state-of-the-art (SOTA) tools. While automated machine learning (AutoML) has been proposed as a partner in clinical predictive modeling, many additional requirements need to be fulfilled to make machine learning accessible for clinician scientists.

To address this gap, we introduce *CliMB*, a no-code AI-enabled partner designed to empower clinician scientists to create predictive models using natural language. *CliMB* guides clinician scientists through the entire medical data science pipeline, thus empowering them to create predictive models from real-world data in just one conversation. *CliMB* also creates structured reports and interpretable visuals. In evaluations involving clinician scientists and systematic comparisons against a baseline GPT-4, *CliMB* consistently demonstrated superior performance in key areas such as planning, error prevention, code execution, and model performance. Moreover, in blinded assessments involving 45 clinicians from diverse specialties and career stages, more than 80% preferred *CliMB* over baseline GPT-4. Overall, by providing a no-code interface with clear guidance and access to SOTA methods in the fields of data-centric AI, AutoML, and interpretable ML, *CliMB* empowers clinician scientists to build robust predictive models.

The proof-of-concept version of *CliMB* is available as open-source software on GitHub: <https://github.com/vanderschaarlab/climb>.

---

\*These authors contributed equally.

# 1 Introduction

Individualized risk prediction is a key enabler of precision medicine [1]. There are many use cases for predictive models; we will highlight three illustrative examples. First, individualized survival predictions can guide the use of adjuvant therapy for patients with early invasive breast cancer [2]. Second, predictive models can guide clinical pathways such as identifying high-risk patients during radiotherapy. Providing these individuals with supplemental clinical evaluations reduces costs while improving patient outcomes [3]. Finally, risk predictions can also facilitate timely interventions, e.g. to reduce mortality in patients at risk of sepsis [4].

Machine learning (ML) methods offer increasingly powerful solutions for risk prediction. These include methods and tools to improve data quality [5, 6, 7, 8, 9, 10, 11], optimize model performance [12, 13, 14, 15, 16], and enhance transparency and interpretability [17, 18, 19]. However with few exceptions, current clinical risk scores are either created with classical statistical methods or a small range of ML methods (random forests appear to dominate [20, 21]). This narrow focus is a problem in prediction because, as highlighted by the “no free lunch” theorem, no single method is deemed ideal for all prediction problems [22, 1]. Overall, the breadth of ML methodology being used in clinical risk prediction (and other applications) remains limited and thus the potential of AI in medicine unfulfilled [23].

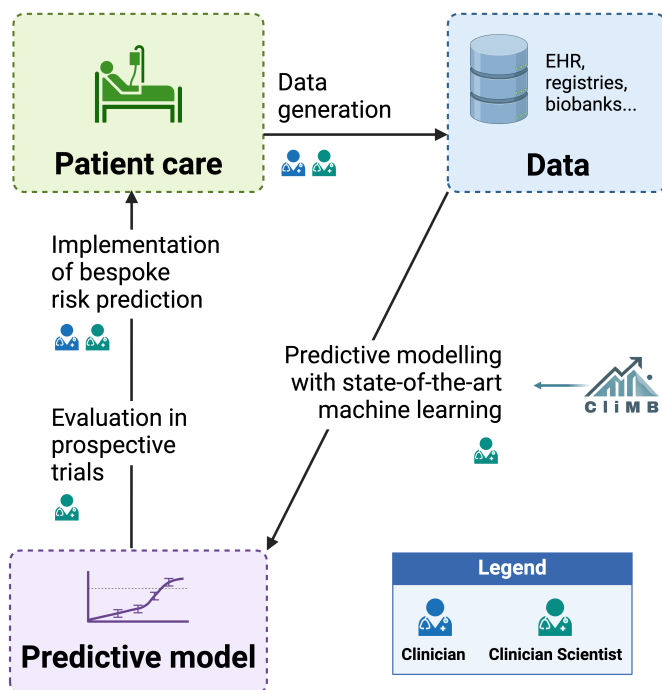
We argue that this is the result of a fundamental conundrum: on the one hand, domain experts (e.g., clinician scientists) should be able to build predictive models; on the other hand, there are largely impenetrable barriers to using SOTA methods. Solving this domain expert-AI-conundrum promises to be transformative for the medical field thanks to more individualized healthcare. Moreover, widespread application of predictive modeling in medicine will scale the identification of real-world needs that should be addressed by the ML community. Overall, a solution to the conundrum would enable progress in reality-centric AI adoption and development.

## 1.1 Clinician scientists should be able to build predictive models

**Why is clinical expertise crucial?** The primary goal of applying ML in the medical domain—and indeed in any domain—is not to explore difficult challenges or develop novel methods that achieve better performance on toy benchmarks. The true value of ML in healthcare lies in its potential to address real-world medical challenges and thereby improve patient outcomes. To ensure a predictive model will serve an actual need, the process of building it should begin with a clinical problem that requires predictions [24]. Clinicians are at the forefront of patient care with a deep understanding of medicine and awareness of clinical needs. They can identify a clear problem and setting where predictions would be promising [25]. Thus, in the clinical domain, clinicians should set the agenda for the development of predictive tools. The necessity of domain expertise is best illustrated by reflecting on the surge in ML models during the COVID-19 pandemic. At the time, hundreds of models with little to no impact were created, in part because efforts were not domain expert-driven [26]. Domain expertise is required not only to set the agenda but also throughout the development process. In addition, domain expertise is necessary for data exploration and engineering, which are usually the most time-intensive phases of data science projects [27]. In fact, many of the models created during the pandemic were built by AI researchers who lacked the medical expertise needed to identify flaws in the medical data [26]. Although the subsequent model-building process may

be more domain-agnostic, interpreting the validity of model predictions (e.g., based on feature importance) requires a deep understanding of the included clinical variables and measurements.

**Could you simply add a data scientist?** While creating dual-expert teams of clinician scientists and data scientists could be a potential solution to the domain expert-AI-conundrum, these teams are often costly, difficult to assemble, and prone to inefficiencies due to translation errors and misaligned goals [24, 1, 28, 29]. In particular, errors in translation between ML researchers and clinician scientists can lead to problems like feature leakage, which result in impaired model quality. In addition, given that demographics and available features change over time, risk scores need to be updated on a regular basis. The necessity to repeat the predictive modeling cycle aggravates the problem, particularly considering the shortage of ML experts [30].



**Figure 1: Cycle of clinical predictive modeling.** In this proposed cycle, data is generated during patient care by clinicians or clinician scientists. This data can be deposited in EHRs, registries, biobanks etc. Clinician scientists then create predictive models using SOTA machine learning. Empowering them to do so is the focus of this paper. Subsequently predictive models are evaluated and then implemented in patient care. This cycle will likely be repeated to adapt to changing features/demographics or new challenges.

Figure created with Biorender.com

**How would clinical risk scores be built in the future?** We envision a cycle in which clinicians or clinician scientists care for their patients, generate data in the process, clinician scientists use this data to build predictive models, evaluate these models in prospective trials, and implement them directly in patient care (see Figure 1). Putting domain experts in the driver’s seat is expected to (1) improve the clinical utility of ML tools, and (2) bridge the gap between method development and real-world application [24].

## 1.2 Clinician scientists face largely impenetrable access barriers

**Why is risk prediction hardly accessible?** Substantial resources and expertise are currently required to develop predictive models [1]. As explained above, no single ML method will be ideal

for the variety of risk scores being built for healthcare. Expecting even a few clinician scientists engaged in routine patient care to keep up with the latest developments in ML and make informed choices about the most promising strategies for their prediction problem is unrealistic.

**What has been done so far to make ML more accessible and why is it insufficient?**

Substantial efforts have been made to “democratize” ML for domain experts with little to no ML knowledge, particularly in the field of automated machine learning (AutoML) [15, 31, 32]. While the AutoML community has created powerful tools to optimize the model building process and make it more systematic and efficient [33], AutoML does not deliver on its original purpose of democratizing ML. Over the past decade, AutoML achievements focused on optimizing predictive performance [34]. However, current tools are not fully aligned with the needs of domain experts such as clinician scientists who seek to apply off-the-shelf ML solutions for their problems [32]. In general, user interaction with AutoML systems remains an underexplored area. In particular, the lack of sufficiently intuitive methods for users to formulate tasks, contribute domain expertise, and communicate preferences and constraints is a major obstacle for people with non-technical backgrounds [34]. For instance, most AutoML tools today require technical understanding and skills (e.g., the ability to use Python packages like AutoPrognosis [13]). As a result, these tools tend to (perhaps inadvertently) cater to ML experts seeking to enhance their workflow, rather than serving domain experts with limited AI knowledge [35]. It has thus been proposed that natural language interfaces are a prerequisite for domain experts to interact comfortably with AutoML and other data science tools [36].

**Which challenges exist beyond intuitive AutoML interfaces?** Usability is further limited because the tools required to build robust risk prediction models are fragmented rather than integrated into a comprehensive pipeline that clinician scientists could default to. The more comprehensive pipelines that do exist are focused on the model-building process but omit the critical and time-consuming phases of data exploration and engineering [35, 37]. Data quality largely determines the performance, fairness, robustness, safety, and scalability of AI systems. As a result, poor data quality can have disproportionately negative effects, particularly in high-stakes domains like medicine. Unfortunately, medical data is usually not ML-ready, making the data engineering phase critical for medical data science projects [38, 24]. Human-centric AI must also consider interpretability regarding the data, modeling process, and outcomes. Interpretability and transparency are crucial sources of trust in AutoML systems [34, 39, 40]. While advanced interpretability tools have been developed [41], they require technical skills to be used and knowledge to be understood. To truly benefit domain experts like clinician scientists, interpretability methods must be accompanied by contextual, easy-to-comprehend explanations [35, 42].

### 1.3 Requirements for human-centric AI/ML partners for predictive modeling

To resolve the domain expert-AI-conundrum and make ML accessible to clinician scientists without the need for a data scientist, an AI-enabled partner is needed that builds on developments in AutoML, data-centric AI, and ML interpretability. We refer to this kind of tool as a partner rather than a “co-pilot” because it would not merely enhance a clinician scientist’s (the pilot’s) capabilities but rather augment the clinician scientist’s skill set with a new class of capabilities. We consulted with clinician scientists, ML researchers, statisticians, and software engineers to distill



the requirements that an ideal partner for predictive modeling in medicine would fulfill. They are listed below:

- ① **Holistic support.** The partner should accompany the domain expert through all stages of their data science project. Before starting the project, an ideal partner would check the alignment between the user’s goal, their data, hardware and software. For easy access and an (albeit modular) off-the-shelf solution, tools for all phases of the predictive modeling pipeline would be combined. These phases include (1) data exploration, (2) data engineering, (3) model building, and (4) model exploitation [43]. Holistic support encompasses not only execution but also planning, which is an underexplored area [44].
- ② **Usability.** To truly empower domain experts, the AI/ML partner must be user-friendly. Usability hinges on automation and interfaces. Most basic tasks should be automated, requesting human feedback only when critical decisions must be made. Communication should occur through natural language, a universal interface [45]. Additionally, the visual presentation must be clear and easy to navigate. Improved usability will also enhance interpretability and transparency.
- ③ **Best practices.** Data science best practices should be followed to create predictive models that perform well, generalize effectively, and maintain clinical validity. E.g., model building should be optimized using AutoML. User misconceptions should be identified and interactively addressed.
- ④ **Robustness.** As evidenced in section 3.2, large language models (LLMs) often deviate from their intended plans, leading to incomplete or suboptimal outcomes. A truly effective AI/ML partner, however, must adhere to robust yet adaptable strategies to prevent it from veering off course and neglecting critical steps.
- ⑤ **Interpretability.** Interpretability and transparency should be integrated at all stages to allow the domain expert to critically appraise processes and outcomes. Interpretability metrics should be made accessible through interactive explanations, contextualization, and visualizations.
- ⑥ **Modularity.** An ideal AI/ML partner should be designed to scale with future methodologies, modularly incorporate novel tools as they emerge, and accommodate existing tools the user wishes to integrate alongside off-the-shelf solutions.
- ⑦ **Versatility.** Within reasonable boundaries, an AI/ML partner should dynamically adapt to user requests. For example, it should be possible to revisit previous phases of the data science pipeline if the need arises at a later stage of the project.
- ⑧ **Privacy.** Privacy is a key concern in medicine [46]. If sensitive medical data are used for data science tasks, appropriate privacy features must be an integral part of any tools that access these data.

## 1.4 An AI-enabled partner for clinical predictive modeling

In this article, we introduce the *Clinical predictive Model Building partner (CliMB)*, a no-code chatbot that empowers clinician scientists to build predictive models using only natural (specialist)

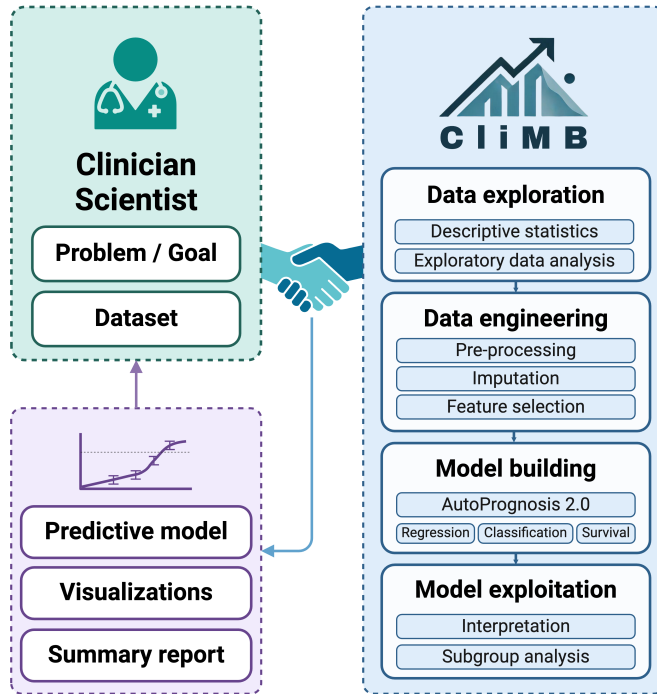


Figure 2: **An AI-enabled partner for clinical predictive model building: *CliMB*.** Clinician scientists turn to *CliMB* with a predictive problem and real-world data. *CliMB* guides the clinician scientist through all phases of the data science pipeline with robust planning and SOTA tools, including the AutoML pipeline AutoPrognosis 2.0 [13] alongside data-centric and interpretability tools (see Table 2). The clinician scientist and *CliMB* partner to generate a predictive model, visualizations, and a summary report of the methodology.

Figure created with Biorender.com

language. *CliMB* supports the entire medical data science pipeline, adhering to the requirements outlined in section 1.3. Using *CliMB*, clinician scientists can build a predictive model from real-world clinical data in one conversation. We demonstrate the advanced capabilities of *CliMB* through (1) sessions in which clinician scientists use *CliMB* to build predictive models with their own medical datasets, (2) simulated modeling sessions to systematically compare *CliMB* against a baseline GPT-4, largely similar to ChatGPT [47], and (3) a survey among a diverse group of clinicians.

The proof-of-concept implementation of *CliMB* is available as open-source software on GitHub (<https://github.com/vanderschaarlab/climb>). Comprehensive project documentation is provided at <https://climb-ai.readthedocs.io/en/latest/>. Detailed installation instructions can be accessed at <https://climb-ai.readthedocs.io/en/latest/installation.html>, and a quickstart guide is available here: <https://climb-ai.readthedocs.io/en/latest/quickstart.html>.

## 2 Methods

### 2.1 Meeting the human-centric AI/ML partner requirements

We built *CliMB* to fulfill the universal requirements for a clinician scientist-centric AI partner. *CliMB* offers holistic support for end-to-end predictive modeling. Its user-friendly natural language interface is complemented by an interactive dashboard, which allows the user to easily view data and data transformations, figures, and progress on the structured plan. This structured plan for

Requirement	Solution in <i>CliMB</i>
Holistic support	End-to-end support for medical data science pipeline with access to SOTA tools in every phase.
Usability	Intuitive user interface with interactive chat and dashboard. Automation of many basic tasks, with human feedback requested when necessary.
Best practices	Adherence to a structured plan for creating predictive models aligned with data science best practices.
Robustness	Advanced reasoning unit formalizes the AI/ML partner problem as a transparent episodic multi-armed bandit to ensure the above plan is followed (see section 2.3). Memory unit containing a working directory, structured plan, files, and logs. Use of external SOTA data science tools (see Table 2).
Interpretability	Intuitive user interface with natural language and visual explanations. Integrated data-centric tools for understanding the clinical dataset and post-hoc interpretability methods for model assessment (see Table 2).
Modularity	Easy-to-extend tool library and modular LLM component.
Versatility	Access to diverse tools and classes of predictive models. Flexible handling of user requests.
Privacy	Projects are performed locally and on secure and encrypted Azure servers.

Table 1: **Overview of requirements and solutions in *CliMB*.**

predictive modeling is aligned with data science best practices and includes all relevant subtasks (for details see Table 7). For instance, *CliMB* checks for data leakage - an issue that was regularly overlooked by clinician scientists before we added this subtask. *CliMB*'s reasoning unit, in combination with the memory unit, and the use of external tools ensure robustness. The reasoning unit is largely responsible for adherence to the structured plan. The memory unit stores logs and files but also all generated and executed code. This information can be retrieved by the reasoning unit to improve robustness, but it can also be viewed by the user to enable transparency. External tools are leveraged to process and interpret data, automate model building, and provide post-hoc interpretability (see Table 2). These tools, combined with *CliMB*'s user interface ensure the interpretability and transparency of the pipeline. While the reasoning unit ensures that a structured plan is being followed, *CliMB* can flexibly adapt to user requests and still ensure that all necessary steps of the plan will be followed. This, alongside access to diverse tools and classes of predictive models, underpins the versatility of *CliMB*.

Finally, data privacy is a critical concern in healthcare that we aim to address. In this context, clinician scientists, as data holders, must act as data stewards to ensure that data usage aligns with legal and ethical standards.[46] To decrease the risk of privacy violations, one approach is the *ex ante* limitation of data access [48]. Beyond the performance limitations of baseline GPT-4 demonstrated in section 3.2, using commercial chatbots like ChatGPT with health data poses data security risks [49]. Effectively using ChatGPT and similar tools for medical data science requires disclosing patient data. However, OpenAI uses past user interactions to retrain their models, and these logs cannot be deleted [50]. Data access could be restricted in several ways of which we will discuss three: prompt-based guardrails, explicit user confirmation, and adjustments to the computing infrastructure.

First, when relying on a third-party LLM provider like OpenAI, users may have some control over which information the LLM accesses. For example, the LLM can be instructed to avoid accessing individual data records and instead focus on summary statistics. However, this “safeguard” is weak and can be easily bypassed by LLM errors or by the user inadvertently instructing it to access the data.

Second, a mechanism can be implemented to require explicit user confirmation for any content sent to the LLM. While this approach could prevent the unintentional release of private data, it would demand significant user effort. Moreover, both of the above options restrict data access, and thus limit the LLM’s ability to perform complex reasoning based on the clinical data.

Therefore, we decided to focus on the computing infrastructure. OpenAI’s GPT-4 can be deployed on trusted servers like Azure, which provide private instances with multiple layers of data security and encryption, fully compliant with GDPR [51]. This approach ensures that confidential information is neither shared with OpenAI nor used to retrain GPT-4. We chose this solution for the initial version of *CliMB*. In the future, GPT-4 could also be substituted by other LLMs (e.g., the open source LLM LLaMA [52]) that can be run on private servers (e.g., hospital-owned) or even local machines, depending on the specific requirements of the clinician scientist. In addition to private deployment of the LLM, the dataset file is stored locally, and all code execution performed by *CliMB* occurs on the user’s machine. More information is provided in section 6.2.

Table 1 summarizes how *CliMB* fulfills the universal requirements for a human-centric AI/ML partner.

## 2.2 Design of CliMB

In this section, we describe the information flow through *CliMB*. *CliMB* adheres to data science best practices, guiding users through their projects. These projects can be divided into the following phases: (1) data exploration, (2) data engineering, (3) model building, and (4) model exploitation [43].

The objective is to build predictive models for the user’s problem using real-world medical datasets. Therefore, the clinician scientist plays a central role within *CliMB*’s human-in-the-loop framework. Other key components include the reasoning unit, the action unit, the memory, and the user interface (see Figure 3).

The reasoning unit receives information from the user, external tools invoked by the action unit, and through self-reflection. The reasoning unit (detailed in section 2.3) integrates this input,

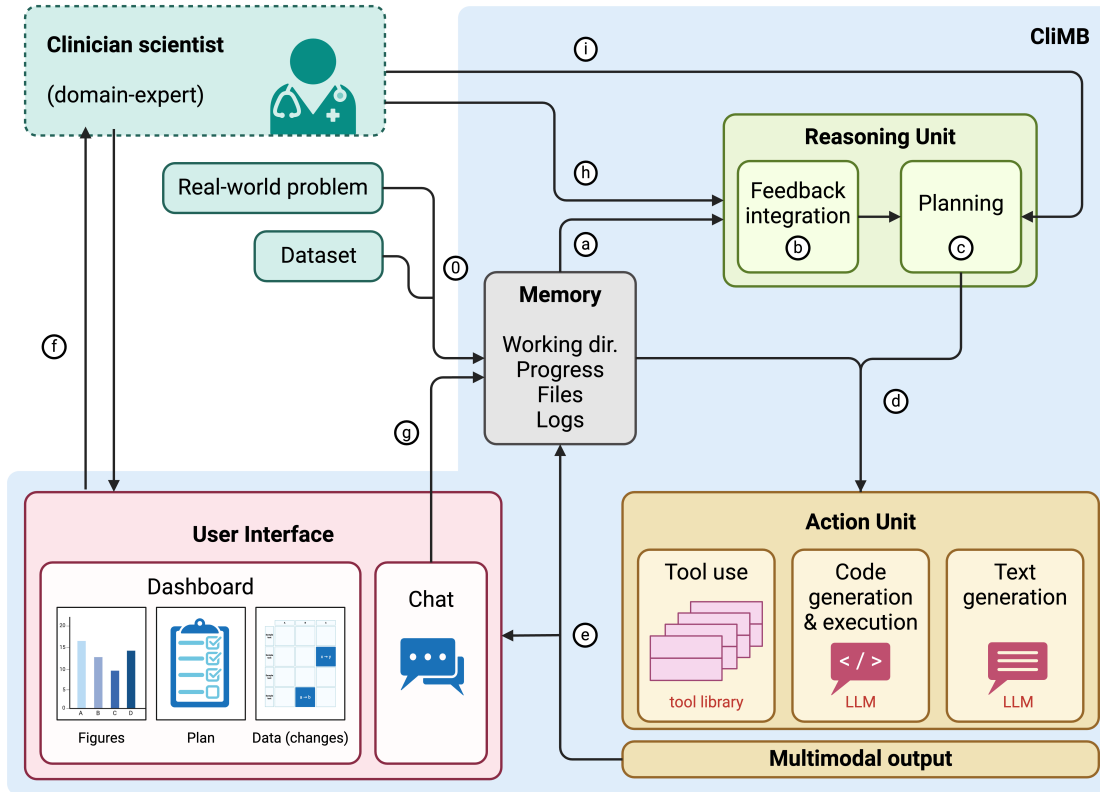


Figure 3: **Design of ClMB.** The information graph highlights the flow of information within *ClMB*. The process begins with the user describing their clinical problem and uploading a corresponding medical dataset (0). The memory unit stores a working directory for tools, continuously updated plans tracking overall progress, files, and logs of the entire user interaction (g). The reasoning unit receives information from the memory (a), user (h), and through self reflection, and integrates this feedback (b) to facilitate subsequent planning (c). The action unit executes the plan (d) and generates multimodal outputs, which are stored in the memory and displayed via the user interface (e). The clinician scientist interacts with the user interface (f) and, at the end of an episode (see section 2.3), validates and concludes the current phase (i).

Figure created with Biorender.com

updates the project plan, and assigns tasks to the action unit. It ensures robustness in following the planned stages necessary for predictive modeling. Planning assistance is necessary to make ML accessible but has been underexplored compared to execution assistance [44]. This likely stems from the fact that many AutoML tools seem to cater mostly to data scientists wanting to enhance their workflow, who are already familiar with standard data science project plans. In contrast, our focus is on clinician scientists who require more support in structuring their predictive modeling efforts.

The action unit is modular by design, currently capable of performing three classes of actions: (1) tool use, (2) code generation/execution, and (3) text generation. These actions are invoked based on tasks assigned by the reasoning unit. For code and text generation, a SOTA LLM is integrated, which can be modularly replaced with more advanced models or those offering enhanced privacy features. Additionally, the action unit has access to SOTA data science tools in three categories: (a) data-centric tools (for phases 1, 2, and 4 of the data science pipeline), (b) AutoML model-building tools (for phase 3), and (c) post-hoc interpretability tools (for phase 4). These tools are summarized in Table 2. Notably, the action unit’s modularity allows for seamless integration of a wide range of existing and emerging tools to expand the capabilities of *CliMB*.

Tool class	Available tools	Data science phases
Data-centric	Descriptive statistics, Exploratory data analysis (EDA), HyperImpute [8], Feature selection [53], DataIQ [11]	(1), (2), (4)
Model building	AutoPrognosis 2.0 [13] (Regression, Classification, Survival analysis)	(3)
Post-hoc interpretability	SHAP explainer [54], Permutation explainer [55], AutoPrognosis subgroup analysis [13]	(4)

Table 2: **Overview of tools available to the action unit in CliMB.**

The multimodal output produced by the action unit is routed through the memory unit to the reasoning unit for reflection. Following an iterative process between the reasoning and action units, the reasoning unit directs the action unit to deliver the multimodal output to the user interface and store it in the memory.

The user interface features a dynamic chat-like component alongside a stable dashboard that provides real-time updates on the current plan, highlights any dataset changes, and displays figures and other files.

Above, we have outlined the general flows of information, which over the course of the data science project, are followed to build a predictive model. In section 3.1, we present illustrative real-world clinical predictive modeling projects completed by clinician scientists using *CliMB*. The design of *CliMB* is summarized in Figure 3.

### 2.3 Reasoning unit

*CliMB*’s reasoning unit formalizes the clinician scientist–AI partner collaboration problem as a transparent episodic multi-armed bandit [56]. This formalism enables the description of both the sequence of decisions made by *CliMB* to build a predictive model with its accompanying visualizations and report, and the feedback mechanism informing the next decision. The division of the end-to-end application into planned subtasks is made possible by the episodic nature of the model.

This formalism is used to structure, through in-context learning, a locally deployed pre-trained language model  $l_{rea}$  (see Figure 4). Let  $\mathcal{T}$  be the vocabulary of tokens used by  $l_{rea}$ . Therefore, texts generated by  $l_{rea}$  belong to  $\mathcal{T}^*$ .

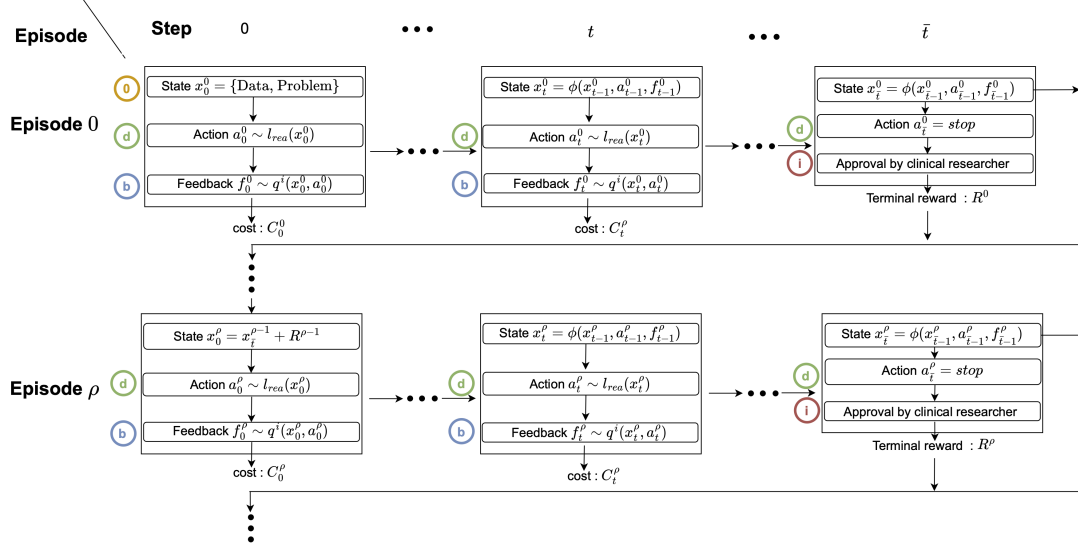


Figure 4: **The reasoning unit modeled as an episodic multi-armed bandit.** The integration of this unit in the information graph detailed in Figure 3 is as follows: (b) Initial objective given by the user, (b) Feedback mechanism, (d) Calling of actions, (i) Validation by the user.

We define a fixed set  $\mathcal{E}$  of project stages, or episode types, composing the “medical data science pipeline.” These episode types or subtasks can be categorized into the following categories: “alignment check”, “data exploration”, “data engineering”, “model building”, and “model exploitation”. All subtasks are provided in appendix 6.3. During an episode  $\rho$ , *CliMB* takes a sequence of actions  $(a_0^\rho, a_1^\rho, \dots, a_t^\rho)$  in  $\bar{\mathcal{A}}$ . These actions can be: (i) continuation actions  $\mathcal{A}$ , which progress toward the final result (model, visuals, report), or (ii) a terminal action *stop*, ending the current episode and requesting human verification. To prevent infinite iterations, we set a maximum number of actions  $l_{max}$ . If the terminal action is not selected before this limit, then  $a_{l_{max}}^\rho = stop$ .

The action  $a_t^\rho \in \bar{\mathcal{A}}$  is selected based on the state  $x_t^\rho$ , which includes the last feedback  $f_{t-1}^\rho$ :  $a_t^\rho = l_{rea}(x_t^\rho)$ . States and feedback are defined as follows:

**States.** The states  $\mathcal{X}$  describe the current progress of the task. As the task may change depending on the episode, a state  $x$  can be a textual representation ( $x \in \mathcal{T}^*$ ) of various objects such as a dataset in the “Data engineering” phase or a model in the “Model building” phase. The dynamics of the evolution of  $x$  caused by the action selected by *CliMB* is given by a state transition function  $\phi^2 : \mathcal{X} \times \mathcal{A} \times \mathcal{F} \mapsto \mathcal{X}$ . The last feedback  $f_t^\rho \in \mathcal{F}$  is concatenated to the new state  $x_{t+1}^\rho$ . For each episode, the initial state  $x_0^\rho$  corresponds to the last state of the previous episode  $x_t^{\rho-1}$  along

<sup>2</sup>Unlike in the original episodic multi-armed bandits formalism, we constrain the state transition function to be time-homogenous.

with its associated terminal reward  $R^\rho$  (see paragraph below), if  $\rho > 0$ . If  $\rho = 0$ , the initial state corresponds to the data and problem provided at the beginning by the clinician scientist.

**Feedback.** The feedback provides interpretations of the current state of the task with additional information, guiding *CliMB* in selecting the next action. The feedback is expressed in natural language:  $\mathcal{F} \subset \mathcal{T}^*$ . The feedback is produced after executing an action by querying a feedback source  $q^i : f_t^\rho \sim q^i(x_t^\rho, a_t^\rho)$ .

There are three sources for feedback ( $i \in \{1, 2, 3\}$ ):

- ① External tools
- ② Self-reflection of the LLM
- ③ Clinician scientist

We prioritize feedback from sources (i) or (ii) to maximize *CliMB*'s autonomous behavior, resorting to human interaction only when necessary. Thus, the cost of an action  $c_{x,a} \in \{0, 1\}$  for  $a \in \mathcal{A}$  and  $x \in \mathcal{X}$  is defined as  $c_{x,a} = 1$  if it queries the clinician scientist and  $c_{x,a} = 0$  if it queries another source. The cost of the stop action is always 0.

**Terminal reward and AI/ML partner objective.** In *CliMB*, the terminal action *stop* is tied to an assessment by the clinical researcher of the episode process and resulting state. The episodic terminal reward,  $R^\rho$  for the episode  $\rho$ , is binary ( $R^\rho \in \{0, 1\}$ ), provided by the clinician scientist, corresponding to a quality and trustworthiness check. Thus, the episodic reward is 1 if the clinician scientist gives their approval and 0 otherwise. *CliMB*'s objective is to maximize terminal rewards, leading to the completion of the subtasks required to obtain the final output (model, visuals, report), while minimizing action costs, i.e., human interactions.

**Robust Planning.** A plan is defined as a set of subtasks  $\mathcal{E}$ . The subtask attributed to episode  $\rho$  is denoted  $e_\rho$ . The plan is completed when all subtasks  $\hat{e} \in \mathcal{E}$  are achieved, meaning there exists an episode  $\rho \in \mathbb{N}$  where  $e_\rho = \hat{e}$  and  $R_\rho = 1$ . This guarantees robustness as it ensures that all steps of the data science pipeline are followed according to best practices. Although  $\mathcal{E}$  is fixed, the order of completing subtasks is adjusted during the process, based on the problem, current progress, and feedback:  $e_\rho = l_{\text{CliMB}}(x_t^{\rho-1}, R^{\rho-1})$ . The plan  $\mathcal{E}$  is described in a dashboard section which is regularly updated to track the progress of the different subtasks.

**Tools available.** Let  $\mathcal{A}^-$  be the set of actions available to a general-purpose LLM, including planning, text generation, ideation, etc. In *CliMB*, we provide the AI/ML partner with a code interpreter and multiple machine learning tools necessary for acting on the state. Without these coding tools,  $\mathcal{A}^-$  alone would not suffice for performing the required actions and iterating autonomously based on the tool's results. Depending on the episode type, the set of tools available  $\mathcal{G}$  may vary but can be categorized as shown in Table 2.

Thus, the set of continuation actions  $\mathcal{A}$  is enriched by the available tools  $\mathcal{G}$ , increasing the range of potential actions for the AI partner:  $\mathcal{A} = \mathcal{A}^- \cup \mathcal{G}$ .

**Scalability.** We propose *CliMB* as a modular framework, adaptable to various tasks depending on the diversity of episode types defined and the quality of available tools  $\mathcal{G}$ . Since this paper focuses on generating risk prediction models for clinician scientists, the episode types correspond to the subtasks required to achieve this goal, and the tools provided are data science tools. However,



the framework could be applied to other tasks, and the quality of results could be enhanced by improving the set of tools.

**Summary of the clinician scientist interactions.** *CliMB*'s reasoning unit has two types of interactions with the clinician scientist. First, the edge ① of the information flow graph presented in Figure 3: the feedback mechanism allows querying the researcher for intermediary approval of decisions or for additional expert knowledge, helping *CliMB* progress on the task. This type of interaction is minimized when possible by choosing other feedback sources, as it is associated with a non-zero cost. Second, the edge ② of the information flow graph in Figure 3: the validation step at the end of each episode provides the clinician scientist with the opportunity to assess the quality of the results and the trustworthiness of the process. For their decisions, the clinician scientist can rely on the information provided by the dashboard and through the chat.

## 2.4 End-to-end Pipeline

*CliMB*'s ability for robust planning, as described in section 2.3, is realized by its reasoning unit. *CliMB* guides the user through a medical data science project by following an explicit (yet adaptable) plan aligned with best practices. Each necessary stage [43] is ensured to be completed unless the user explicitly decides to skip it. In this section, we outline the standard pipeline, which can be adapted to meet specific user needs. A detailed overview of the plan is available in Table 7.

- ② **Alignment check.** *CliMB* verifies whether the hardware is suitable and whether the dataset can be loaded and used. Background information on the dataset and the clinician scientist's research question(s) is confirmed to ensure alignment between *CliMB*'s capabilities and the user's requirements.
- ① **Data exploration.** *CliMB* performs detailed exploratory data analysis, generates descriptive statistics, and creates informative figures for each variable. The meaning of each column and the expected data type is confirmed with the user.
- ② **Data engineering.** This phase includes data type conversion and the explicit discussion and handling of missing values. Additionally, any changes requested by the user are performed. For full transparency, data changes are highlighted within the dashboard. Feature selection is conducted using random-forest-based algorithms.
- ③ **Model building.** Model building is optimized using SOTA AutoML risk prediction software (AutoPrognosis 2.0 [13]). Currently, classification, regression, and time-to-event (survival) analysis can be performed.
- ④ **Model exploitation.** In this phase, model performance is evaluated, and depending on the project type, additional insights are provided to the user. Interpretability methods are employed to assess feature importance [54, 55] and, for classification tasks, stratify samples (easy, ambiguous, hard) [11]. Once all steps are completed and the clinician scientist is satisfied with the results, a summary report of the study is generated.

We show the breakdown of *CliMB* project stages' tasks and subtasks in appendix 6.3, and an illustration of the advantages of *CliMB*'s approach in robustly planning an AI-enabled data science project in appendix 6.4.

## 2.5 Hardware and Software

*CliMB* is implemented in Python, with backward compatibility up to Python 3.8. The software is compatible with all major operating systems: Windows, Linux, and macOS. The `miniconda` [57] package and environment manager is used to facilitate code and tool execution. Automatic installation of PyPI [58] (`pip`-installable) packages in a dedicated code execution environment is implemented.

The experiments were carried out using the `gpt-4-0125-preview` model with a 128,000-token context window via an Azure OpenAI Service [59] deployment. However, all OpenAI and Azure OpenAI Service provisions of the `gpt3.5-turbo`, `gpt4(-turbo)`, and `gpt4o` class models are compatible through the Python `openai` library [60]. Compatibility with other commercial and open-source LLMs is available through simple extension, hence *CliMB* is not limited to the models used here. The packages required for *CliMB* tools, such as AutoPrognosis 2.0 [13] or HyperImpute [8], are installed as part of the *CliMB* setup.

The hardware used for developing and running *CliMB* was a workstation running Ubuntu 24.04, powered by an Intel Core i9-10900K 10-core, 20-thread CPU, 64 GB of system memory, and an NVIDIA GeForce RTX 3090 GPU (24 GB VRAM). We estimate the minimum system specifications for running *CliMB* to be: a CPU with at least 4 cores, 16 GB of system RAM; optionally, a GPU with at least 4 GB of memory is advantageous for running some of the deep learning models. However, it should be noted that the appropriate computing requirements depend significantly on the size of the user’s dataset.

## 3 Results

### 3.1 Illustrative end-to-end sessions with clinician scientists

We tested *CliMB*’s capabilities to assist real-world clinician scientists in creating predictive models from their medical datasets. Predictive models for various use cases and diseases were created during these sessions. Illustrative logs are included in the appendix. We further include a video of a full *CliMB* session here <https://youtu.be/76XuR0K3F5Y>. Figure 5 provides snippets from this session in which a transplant surgeon built a predictive model for graft function at 12 months. These snippets highlight just a few of the versatile capabilities of *CliMB*.

### 3.2 Systematic comparison against baseline GPT-4

We systematically compare *CliMB* with a baseline GPT-4 software system. The latter is implemented as a `gpt4-turbo` chat interface with code execution capabilities with self-reflection. The LLM, its parameters, and the code execution environment implementation are equivalent in both the baseline and *CliMB* to ensure a fair comparison. Python code execution with self-reflection, access to (at least some) `pip`-installable libraries and a working directory, as implemented in the baseline, appears to be similar to ChatGPT’s [61] approach to code execution and data analysis as of August 2024, based on empirical observation of the ChatGPT online portal’s code generation behavior (for instance, generated code and feedback from exceptions can be confirmed by assessing

### Exploratory Data Analysis

[Explanation of project plan incl. Exploratory Data Analysis as a starting point]  
 Before proceeding, I would like to ensure that you're satisfied with this direction and if there's any specific aspect you'd like more information on or further assistance with.

Yes, this sounds good.

The Exploratory Data Analysis (EDA) has provided us with some detailed insights into your dataset `vasilis_0614_user_cols.csv`. Here are some key highlights:

- Dataset Shape:** Your dataset consists of 1000 rows and 33 columns.
- Column Types:** The dataset contains a mix of numerical and categorical columns. Certain numeric columns have been identified that might actually be categorical due to their discrete nature and low unique value count. These include demographic and graft-related features like `DSEX`, `DETHNIC`, `HYPERTENSION`, and `organ_type`.
- Descriptive Statistics:** Provided for numerical features with insights into count, mean, standard deviation, minimum, and maximum values. This includes columns such as `DAGE`, `DSEX`, `DRET_CREATE`, and `eGFR12`.
- Categorical Variables:** Detailed information on categorical variables is given, highlighting the top values and their frequencies. For example, `DETHNIC`, `GCENS`, and `TX_TYPE` show distinct categorical distributions.
- Missing Values:** A missing value analysis reveals columns like `DETHNIC`, `HYPERTENSION`, `DRET_CREATE`, and `RBMI` have missing values, with `HLA_MM` having all values missing, indicating potential areas for data cleaning.
- Outlier Identification:** Outliers have been identified in several columns, such as `DETHNIC`, `HYPERTENSION`, and `CIT_MINS`, which might require further investigation.
- Correlation Analysis:** Performed but not detailed here due to the large number of columns.
- Duplicate Records:** No duplicate records were found in your dataset.

This analysis is crucial for understanding the basic structure of your data, identifying data quality issues, and guiding subsequent data preprocessing steps.

[...]

### Dashboard

### Missing data handling

[Explanation of missing data handling]  
 Since the target variable `eGFR12` has about 16.42% missing values, it's crucial to address this without compromising the integrity of the dataset. Imputing the target variable is generally not recommended due to the risk of introducing bias. However, for other columns, imputation could be a viable strategy to retain as much data as possible.

Given the high percentage of rows with missing values, dropping rows with missing data in any column might not be the best approach, especially if we aim to maintain a substantial dataset size for analysis. Imputation could be a more suitable option, particularly for columns critical to the analysis that do not include the target variable.

[...]

Imputation makes sense for non-eGFR12 columns. For eGFR12: When the value is missing, then:

- If the corresponding `GSURV` value is between 0 and 364 (inclusive) and the corresponding `GCENS` is 1 then the `eGFR12` value should equal 10. This indicates that the graft was lost prior to the endpoint.
- If the corresponding `GSURV` value is between 0 and 364 (inclusive) and the corresponding `GCENS` is 0 then rows with missing `eGFR12` value should be deleted. This indicates that the patient died with a functioning graft.
- Leave the other missing values as they are, those are actually missing.

[Climb continues by generating the necessary code to apply these rules]

### Dashboard

### Automated Machine Learning

Yes, it is a regression study with `eGFR12` as the target variable.

Tool `AutoPrognosis2.0` completed successfully

```

Tool logs:
Setting up the regression study...
Trying regressors:
[
  "linear_regression",
  "bayesian_fidge",
  "random_forest_regressor",
  "mlp_regressor"
]
Running the regression study, this may take several minutes...
AutoPrognosis 2.0: Duration of this iteration: 0.9s
AutoPrognosis 2.0: Investigating model: data_cleanup->linear_regression
AutoPrognosis 2.0: R2 achieved: 0.339
...
AutoPrognosis 2.0: Investigating model:
0.9999999999999999 * (log-sum/fora_transform->data_cleanup->random_forest_regres
AutoPrognosis 2.0: R2 achieved: 0.447
AutoPrognosis 2.0: Study finished.
Regression study completed!
Final model summary:
    
```

### Feature importance

The feature importance analysis using the `SHAP` explainer has been completed successfully. This analysis enables us to understand which features are most influential in predicting the `eGFR12` outcome, providing valuable insights into the factors that significantly impact graft function at 12 months.

### Dashboard

Figure 5: Illustrative usage of Climb. Snippets of interactions during a full end-to-end model-building session are provided from all four phases of the medical data science project: (1) data exploration (shown: Exploratory Data Analysis), (2) data engineering (shown: data transformation), (3) model building (shown: AutoPrognosis 2.0 Survival Analysis [13]), and (4) model exploitation (shown: SHAP explainer [54] for interpreting feature importance). *Note:* illustrative examples shown do not necessarily correspond to the sessions used in experiments.

Feature	CliMB	Baseline GPT-4
Flexible adaptation to user prompts	✓	✓
Robust planning	✓	×
Advanced memory	✓	×
Access to SOTA external tools	✓	×
Informative dashboard	✓	×
Privacy features	✓	×

Table 3: **Comparison of features between CliMB and baseline GPT-4.**

the content of the “Analyzing...” snippet). We do, however, note that the exact implementation details of ChatGPT are not disclosed by OpenAI. We opt for implementing a baseline system rather than using ChatGPT online as (i) this allows us to make the baseline and *CliMB* implementations directly comparable, and (ii) this allows the use of the Azure OpenAI Service, which was necessary for data privacy when working with real-world clinical data.

From the outset, baseline GPT-4 lacks several crucial capabilities that *CliMB* integrates. These are summarized in Table 3. To systematically compare *CliMB*’s capabilities with those of baseline GPT-4, standardized sessions were conducted in which a simulated clinician scientist built predictive models from the same clinical dataset using both *CliMB* and baseline GPT-4.

The following assumptions were made for the simulated clinician scientist persona:

- ① No deeper knowledge of data science, machine learning or coding
- ② Knowledge of their clinical field
- ③ Knowledge of the variables in their dataset
- ④ A clear idea of the predictions they want to make

**Experiment setup.** To create a realistic scenario, we used a real-world clinical dataset that has not been published before and is therefore not part of the corpus GPT-4 was trained on. This dataset contains information about kidney transplants performed in UK hospitals. Variables in the dataset are categorized into donor characteristics, recipient characteristics, and graft characteristics (including survival). During the interaction, the user requested two dataset-specific steps: (1) removal of specific columns and (2) subgroup analysis by donor sex.

The objective during the end-to-end runs with both chatbots was to create a predictive model for graft function at 12 months (eGFR12). This task was performed with *CliMB* and baseline GPT-4 (5 replicates each). Full logs of these interactions were then analyzed.

The evaluation metrics were computed on a held-out test set that was not used during model development. After completing the training session on the initial dataset using either the baseline approach or the *CliMB* system, the same data preprocessing steps were applied to the separate test

dataset. The final model generated by either method was then used to make predictions on the preprocessed test set, which were subsequently evaluated.

**Failures in various categories.** Failures were categorized and are presented in Table 4. While both *CliMB* and baseline GPT-4 completed all end-to-end runs, baseline GPT-4 exhibited more frequent failures across diverse categories and stages over the five runs.

Table 4: **Failures of CliMB and baseline GPT-4 in various categories.** The table lists the proportion of runs in which a failure type occurred.

Failure category	Stage	Baseline GPT-4	CliMB
Did not finish	N/A	0/5	0/5
EDA partially failed	Data exploration	1/5	0/5
Did not save the models (unless prompted)	Model building	3/5	0/5
Did not provide an opportunity for user to review or transform data features for their needs	Data engineering	2/5	0/5
Imputed the target variable without checking with the user	Data engineering (missing values)	4/5	0/5
Dropped rows with any missing values, resulting in dropping most samples in the dataset	Data engineering (missing values)	1/5	0/5
Did not use cross validation	Model building	4/5	0/5
Subgroup analysis done by retraining instead of using existing model	Model exploitation	2/5	0/5

**Identification of dataset specific issues.** In addition, we quantified whether each chatbot would identify two issues specific to the real-world dataset used: (1) the inclusion of ID columns, which could lead to model overfitting and hamper interpretability; and (2) the presence of certain columns which, if included in the model, would result in data leakage because these values are not available at the study baseline time (these fields include a second measure of graft function at 3 months, graft survival time and censoring variables, rejection, delayed graft function, and cause of graft failure). In all instances (5/5), baseline GPT-4 missed both issues, while *CliMB* missed issue 1 in 0/5 runs and issue 2 in only 1/5 runs.

**Planning failures.** To investigate the planning capabilities of *CliMB* and baseline GPT-4, we assessed whether each chatbot completed all tasks it had planned in each phase. This evaluation included only the tasks planned by the chatbot itself during the session. *CliMB* demonstrated

robust planning, consistently completing all planned tasks in each stage. In contrast, baseline GPT-4 frequently failed to complete tasks it had planned (see Table 5).

Table 5: **Planning failures.** The table lists the proportion of runs in which the respective chatbot failed to complete all its planned tasks for the specific stage.

Stage	Baseline GPT-4	CliMB
Data exploration and engineering	3/5	0/5
Model building	0/5	0/5
Model exploitation	1/5	0/5

**Code generation and execution.** The success of code generation and execution was measured by the average number of exceptions encountered per run. *CliMB* encountered significantly fewer exceptions ( $0.4 \pm 0.4$ ) compared to baseline GPT-4 ( $4.6 \pm 2.3$ ) (unpaired t-test,  $t(8) = 3.202$ ,  $p = 0.013$ ).

**Model performance.** Finally, we measured the performance of the predictive model from each session using four metrics: mean squared error (MSE), root mean squared error (RMSE), mean absolute error (MAE), and  $R^2$ . When a data leak was not removed during the data phases, the metrics were computed after manually excluding the leak feature. *CliMB* performed significantly better than baseline GPT-4 across all metrics, as summarized in Table 6.

Table 6: **Model performance.** Comparison between *CliMB* and baseline GPT-4 across key metrics (MSE, RMSE, MAE,  $R^2$ ). The mean and standard deviation for each metric are reported. *CliMB* significantly outperforms baseline GPT-4 across all metrics. Statistical significance is indicated by asterisks based on unpaired t-tests (following outlier removal where applicable): \*\*  $p < 0.01$ , \*\*\*  $p < 0.001$ , \*\*\*\*  $p < 0.0001$ . †: 1 outlier identified and removed using the ROUT method ( $Q = 1\%$ ) before performing unpaired t-tests.

	MSE	RMSE	MAE	$R^2$
baseline GPT-4	433.7 (173.6)†	20.55 (3.78)†	15.50 (3.69)†	0.148 (0.208)
CliMB	286.3 (5.7)***	16.81 (0.09)****	12.03 (0.46)***	0.485 (0.014)**

### 3.3 Clinician feedback

To get a first impression of *CliMB*'s perceived usefulness in real-world application, we gathered feedback from clinicians at the *Cambridge AI in Medicine Summer School* [62]. As part of the summer school curriculum, video demonstrations (see appendix 6.6) were shown to participants, initially without any additional commentary, allowing them to focus fully on the videos. In these videos, the same type of predictive model was built from the same dataset twice: once with baseline GPT-4, once with *CliMB*. While the goal of the summer school session was to demonstrate, how *CliMB* truly enables the attendees to build predictive models, there was no indication as to which tool was being used in either demonstration, effectively resulting in a blinded evaluation. Following

the demonstrations, participants voted in a survey to compare *CliMB* against baseline GPT-4. Consent was obtained (see appendix section 6.7 for the full consent form and information provided to participants).

Clinicians from diverse career stages participated - from medical students to consultants/attending physicians. Overall, about 82% of these clinicians preferred *CliMB* over baseline GPT-4. The most frequent reasons for choosing *CliMB* over baseline GPT-4 included its clear guidance and user interface (see Figure 6).

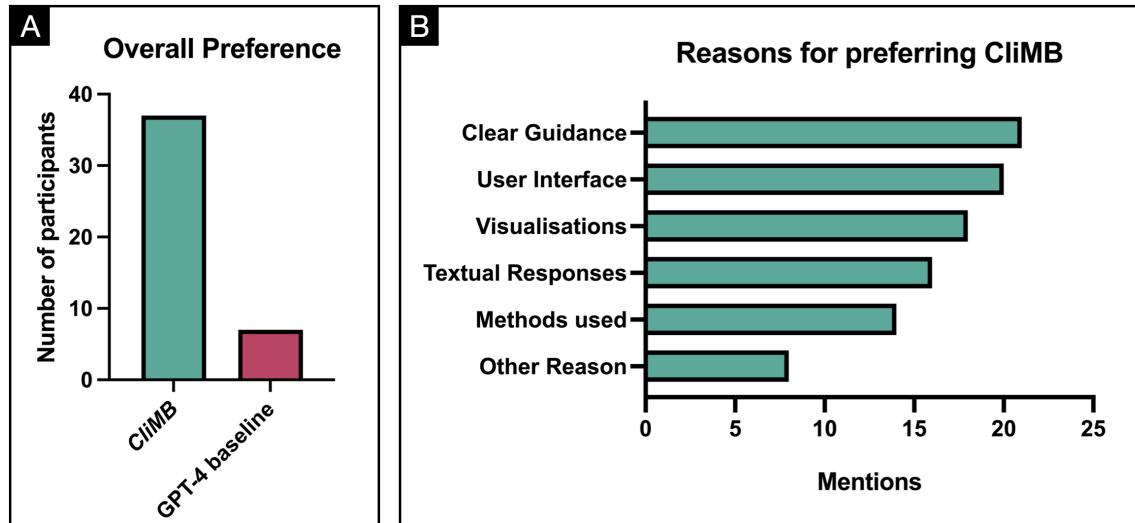


Figure 6: **Clinician preferences in a direct comparison of *CliMB* and baseline GPT-4.** A) Overall, 37/45 participants preferred *CliMB* to baseline GPT-4. B) The bar plot on the right summarizes the reasons mentioned by participants who preferred *CliMB*.

## 4 Discussion

*CliMB* is a first-of-its-kind solution to the domain expert-AI-conundrum for clinical research. It meets the universal requirements outlined in section 1.3, as explained in section 2.1.

### 4.1 CliMB outperforms baseline GPT-4

In section 3.2, we demonstrated that *CliMB* consistently outperforms baseline GPT-4 across multiple dimensions. First, *CliMB* typically avoids a variety of potential failures during different stages of the data science project. It also exhibits a strong capability to identify dataset-specific issues, such as potential data leaks or risks of overfitting. By successfully flagging these issues in nearly every instance, *CliMB* minimizes risks that could compromise the validity of the models. In contrast, baseline GPT-4 consistently overlooked these issues, which raises concerns about its reliability in real-world applications.

These differences reflect the advantages of the advanced reasoning unit of *CliMB*. *CliMB* demonstrated robust planning by ensuring that all planned steps of the workflow were completed. The planned steps themselves follow data science best practices. In contrast, baseline GPT-4 frequently failed to complete even the tasks it had planned itself, which could lead to invalid analyses, downstream failures, and missed quality checks.

Moreover, *CliMB*'s code generation and execution capabilities were markedly superior, with significantly fewer exceptions encountered during runs.

Finally, the models generated by *CliMB* consistently outperformed those created by baseline GPT-4 across key performance metrics that reflect higher accuracy and predictive power.

Most importantly, *CliMB* is preferred by clinicians as demonstrated in section 3.3. Interestingly, the most common reason for this preference was the clear guidance provided by *CliMB*, which it can provide thanks to its structured plan. These statements also highlight the demand for planning assistance.

To sum up, *CliMB*'s infrastructure, with its robust planning and SOTA tools, delivers a more dependable and effective solution for medical data science. *CliMB* represents a significant advancement over baseline GPT-4, all while incorporating data privacy.

## 4.2 Limitations

Our work has several limitations that are outlined below.

- ① While the need for technical skills is mostly eliminated, clinician scientists who want to build and use predictive models still require other skills and knowledge. Specifically, they need a basic understanding of ML and a thorough understanding of their dataset to avoid overreliance on the system.
- ② The predictive models created with *CliMB* require further validation and must be tested for clinical effectiveness and safety before deployment [63]. Therefore, clinicians who wish to deploy predictive models in practice need a proficient level of AI expertise. By that, we mean an understanding of the ethical implications of integrating AI in medicine and the ability to critically evaluate the predictive models created with *CliMB*.
- ③ Finally, *CliMB* should undergo more systematic evaluation in collaboration with clinician scientists from various disciplines. This process should encompass all types of predictive problems and utilize datasets of varying sizes with different dataset-specific challenges.

## 4.3 Future vision

In this work, we present a core technology that is both versatile, making it applicable across various specialties, and modular, allowing it to scale with ease. *CliMB* can be augmented with additional tools to handle multi-modal data. Moreover, future versions can expand to include machine learning methodology beyond risk prediction (e.g., causal machine learning for CATE estimation [64]) or handle competing risks during survival analysis [65]. In addition to expanding the core version, we



envision more specialized versions of *CliMB*-like tools. To provide an example, these could include multi-modal chatbots offered by registries to interact with the registry data.

*CliMB* was built with the vision of an ideal AI/ML partner for clinical risk prediction in mind (see section 2.1). As a result, *CliMB* democratizes predictive modeling in medicine. It is one of the first realizations of the vision of a collaborative design that integrates the complementary strengths of human expertise and AutoML methodologies, as articulated by Lindauer and colleagues [34]. *CliMB* extends this vision by also incorporating data-centric and post-hoc interpretability methods (see Table 2) and by offering robust planning assistance. This development enables a larger and more diverse group of clinician scientists to build predictive models for the real-world problems they face in patient care. On one hand, it bridges the gap between powerful existing solutions and their application in medicine. On the other hand, by scaling the use of ML in real-world settings, new problems will be identified by domain experts and relayed to the ML community, which can focus on finding technical solutions to unsolved problems. *CliMB* facilitates a dialogue that will accelerate progress toward reality-centric AI.

Ultimately, *CliMB* brings the community closer to realizing AI’s full potential in healthcare. Our goal is to empower clinician scientists through *CliMB* to build accurate, trustworthy, and interpretable predictive models that have a tangible positive impact on patient care. We want to empower clinician scientists to build machine learning that matters [66].

## 5 Acknowledgments

We thank Andres Floto, Tim Oosterlinck, Andriy Melnik, Tom Callender, Janet Allen, Rachel Smith, Will Brown, and Courtney Kremler for their invaluable clinical knowledge and feedback on *CliMB* and Andrew Rashbass, Nabeel Seedat, Jiashuo Liu, Nicolás Astorga, Alicia Curth, Paulius Rauba, Robert Davis, Claudio Fanconi, Paul Schubert, and Favian Bauer for helpful discussions. This work was supported by Microsoft’s Accelerate Foundation Models Academic Research initiative. TS acknowledges scholarships from the Medical Faculty at Heidelberg University and the German Academic Scholarship Foundation. VK acknowledges funding as a Paul I. Terasaki Scholar (G106170).

## References

- [1] Thomas Callender and Mihaela van der Schaar. Automated machine learning as a partner in predictive modelling. *Lancet Digit. Health*, 5(5):e254–e256, May 2023.
- [2] Ahmed M Alaa, Deepti Gurdasani, Adrian L Harris, Jem Rashbass, and Mihaela van der Schaar. Machine learning to guide the use of adjuvant therapies for breast cancer. *Nature Machine Intelligence*, 3(8):716–726, June 2021.
- [3] Divya Natesan, Eric L Eisenstein, Samantha M Thomas, Neville C W Eclow, Nicole H Dalal, Sarah J Stephens, Mary Malicki, Stacey Shields, Alyssa Cobb, Yvonne M Mowery, Donna Niedzwiecki, Jessica D Tenenbaum, Manisha Palta, and Julian C Hong. Health care cost reductions with machine learning-directed evaluations during radiation therapy - an economic analysis of a randomized controlled study. *NEJM AI*, 1(4), April 2024.
- [4] Kamran Fahad, Tjandra Donna, Heiler Andrew, Virzi Jessica, Singh Karandeep, King Jessie E., Valley Thomas S., and Wiens Jenna. Evaluation of sepsis prediction models before onset of treatment. *NEJM AI*, 1(3):AIoa2300032, February 2024.
- [5] Nabeel Seedat, Fergus Imrie, and Mihaela van der Schaar. Navigating data-centric artificial intelligence with DC-check: Advances, challenges, and opportunities. *IEEE Trans. Artif. Intell.*, 5(6):2589–2603, June 2024.
- [6] Geoff Pleiss, Tianyi Zhang, Ethan Elenberg, and Kilian Q Weinberger. Identifying mislabeled data using the area under the margin ranking. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, number Article 1430 in NIPS '20, pages 17044–17056, Red Hook, NY, USA, December 2020. Curran Associates Inc.
- [7] Jinsung Yoon, James Jordon, and Mihaela Schaar. GAIN: Missing data imputation using generative adversarial nets. In *International Conference on Machine Learning*, pages 5689–5698. PMLR, July 2018.
- [8] Daniel Jarrett, Bogdan C Cebere, Tennison Liu, Alicia Curth, and Mihaela van der Schaar. HyperImpute: Generalized iterative imputation with automatic model selection. In *International Conference on Machine Learning*, pages 9916–9937. PMLR, June 2022.
- [9] Swabha Swayamdipta, Roy Schwartz, Nicholas Lourie, Yizhong Wang, Hannaneh Hajishirzi, Noah A Smith, and Yejin Choi. Dataset cartography: Mapping and diagnosing datasets with training dynamics. In *Conference on Empirical Methods in Natural Language Processing*, 2020.
- [10] Michael R Smith, Tony Martinez, and Christophe Giraud-Carrier. An instance level analysis of data complexity. *Mach. Learn.*, 95(2):225–256, May 2014.
- [11] Nabeel Seedat, Jonathan Crabbe, Ioana Bica, and M Schaar. Data-IQ: Characterizing subgroups with heterogeneous outcomes in tabular data. *Neural Inf Process Syst*, abs/2210.13043:23660–23674, October 2022.
- [12] Matthias Feurer, Katharina Eggensperger, Stefan Falkner, Marius Lindauer, and Frank Hutter. Auto-sklearn 2.0: Hands-free AutoML via meta-learning. *Journal of Machine Learning Research*, 23(261):1–61, 2022.

- [13] Fergus Imrie, Bogdan Cebere, Eoin F McKinney, and Mihaela van der Schaar. AutoPrognosis 2.0: Democratizing diagnostic and prognostic modeling in healthcare with automated machine learning. *PLoS Digit. Health*, 2(6):e0000276, June 2023.
- [14] Nick Erickson, Jonas Mueller, Alexander Shirkov, Hang Zhang, Pedro Larroy, Mu Li, and Alexander Smola. AutoGluon-tabular: Robust and accurate AutoML for structured data. *arXiv [stat.ML]*, March 2020.
- [15] Lars Kotthoff, C Thornton, H Hoos, F Hutter, and Kevin Leyton-Brown. Auto-WEKA 2.0: Automatic model selection and hyperparameter optimization in WEKA. *J. Mach. Learn. Res.*, 18(25):25:1–25:5, March 2017.
- [16] M Lindauer, Katharina Eggensperger, Matthias Feurer, André Biedenkapp, Difan Deng, C Benjamins, René Sass, and F Hutter. SMAC3: A versatile bayesian optimization package for hyperparameter optimization. *J. Mach. Learn. Res.*, 23(54):54:1–54:9, September 2021.
- [17] Amirata Ghorbani and James Zou. Data shapley: Equitable valuation of data for machine learning. In *International Conference on Machine Learning*, pages 2242–2251. PMLR, May 2019.
- [18] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Why should I trust you?: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, New York, NY, USA, August 2016. ACM.
- [19] Amirata Ghorbani, James Wexler, James Y Zou, and Been Kim. Towards automatic concept-based explanations. *Neural Inf Process Syst*, pages 9273–9282, February 2019.
- [20] Wenjuan Wang, Martin Kiik, Niels Peek, Vasa Curcin, Iain J Marshall, Anthony G Rudd, Yanzhong Wang, Abdel Douiri, Charles D Wolfe, and Benjamin Bray. A systematic review of machine learning models for predicting outcomes of stroke with structured data. *PLoS One*, 15(6):e0234722, June 2020.
- [21] Benjamin A Goldstein, Ann Marie Navar, Michael J Pencina, and John P A Ioannidis. Opportunities and challenges in developing risk prediction models with electronic health records data: a systematic review. *J. Am. Med. Inform. Assoc.*, 24(1):198–208, January 2017.
- [22] D H Wolpert and W G Macready. No free lunch theorems for optimization. *IEEE Trans. Evol. Comput.*, 1(1):67–82, April 1997.
- [23] Pranav Rajpurkar, Emma Chen, Oishi Banerjee, and Eric J Topol. AI in health and medicine. *Nat. Med.*, 28(1):31–38, January 2022.
- [24] Aparna Balagopalan, Ioana Baldini, Leo Anthony Celi, Judy Gichoya, Liam G McCoy, Tristan Naumann, Uri Shalit, Mihaela van der Schaar, and Kiri L Wagstaff. Machine learning for healthcare that matters: Reorienting from technical novelty to equitable impact. *PLoS Digit. Health*, 3(4):e0000474, April 2024.
- [25] Laure Wynants, Ben Van Calster, Gary S Collins, Richard D Riley, Georg Heinze, Ewoud Schuit, Marc M J Bonten, Darren L Dahly, Johanna A A Damen, Thomas P A Debray, Valentijn M T de Jong, Maarten De Vos, Paul Dhiman, Maria C Haller, Michael O Harhay, Liesbet

- Henckaerts, Pauline Heus, Michael Kammer, Nina Kreuzberger, Anna Lohmann, Kim Luijken, Jie Ma, Glen P Martin, David J McLernon, Constanza L Andaur Navarro, Johannes B Reitsma, Jamie C Sergeant, Chunhu Shi, Nicole Skoetz, Luc J M Smits, Kym I E Snell, Matthew Sperin, René Spijker, Ewout W Steyerberg, Toshihiko Takada, Ioanna Tzoulaki, Sander M J van Kuijk, Bas van Bussel, Iwan C C van der Horst, Florian S van Royen, Jan Y Verbakel, Christine Wallisch, Jack Wilkinson, Robert Wolff, Lotty Hooft, Karel G M Moons, and Maarten van Smeden. Prediction models for diagnosis and prognosis of covid-19: systematic review and critical appraisal. *BMJ*, 369:m1328, April 2020.
- [26] Will Douglas Heaven. Hundreds of AI tools have been built to catch covid. none of them helped. <https://www.technologyreview.com/2021/07/30/1030329/machine-learning-ai-failed-covid-hospital-diagnosis-pandemic/>, July 2021. Accessed: 2024-7-5.
- [27] Florian Pfisterer, Janek Thomas, and B Bischl. Towards human centered AutoML. *ArXiv*, abs/1911.02391, November 2019.
- [28] Yaoli Mao, Dakuo Wang, Michael Muller, Kush R Varshney, Ioana Baldini, Casey Dugan, and Aleksandra Mojsilović. How data scientists work together with domain experts in scientific collaborations: To find the right answer or to ask the right question? *Proc. ACM Hum. Comput. Interact.*, 3(GROUP):1–23, December 2019.
- [29] Youyang Hou and Dakuo Wang. Hacking with NPOs: Collaborative analytics and broker roles in civic data hackathons. *Proc. ACM Hum. Comput. Interact.*, 1(CSCW):1–16, December 2017.
- [30] Koen Van der Blom, Alex Serban, Holger Hoos, and Joost Visser. AutoML adoption in ML software. In *8th ICML Workshop on Automated Machine Learning (AutoML)*, 2021.
- [31] Matthias Feurer, Aaron Klein, Katharina Eggenberger, Jost Springenberg, Manuel Blum, and Frank Hutter. Efficient and robust automated machine learning. *Advances in Neural Information Processing Systems*, 28, 2015.
- [32] M Lindauer and A Tornede. Rethinking AutoML: Advancing from a machine-centered to human-centered paradigm. <https://www.automl.org/rethinking-automl-advancing-from-a-machine-centered-to-human-centered-paradigm/>, 2022. Accessed: 2024-7-1.
- [33] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren, editors. *Automated Machine Learning*. Springer International Publishing, 2019.
- [34] Marius Lindauer, Florian Karl, Anne Klier, Julia Moosbauer, Alexander Tornede, Andreas C Mueller, Frank Hutter, Matthias Feurer, and Bernd Bischl. Position: A call to action for a human-centered AutoML paradigm. In *International Conference on Machine Learning*, pages 30566–30584. PMLR, July 2024.
- [35] Alexander Tornede, Difan Deng, Theresa Eimer, Joseph Giovanelli, Aditya Mohan, Tim Ruhkopf, Sarah Segel, Daphne Theodorakopoulos, Tanja Tornede, Henning Wachsmuth, and M Lindauer. AutoML in the age of large language models: Current challenges, future opportunities and risks. *Transactions on Machine Learning Research*, abs/2306.08107, June 2023.

- [36] Shubhra Kanti Karmaker (“Santu”), Md Mahadi Hassan, Micah J Smith, Lei Xu, Chengxiang Zhai, and Kalyan Veeramachaneni. AutoML to date and beyond: Challenges and opportunities. *ACM Comput. Surv.*, 54(8):1–36, October 2021.
- [37] Sushant Kumar, Sumit Datta, Vishakha Singh, Sanjay Kumar Singh, and Ritesh Sharma. Opportunities and challenges in data-centric AI. *IEEE Access*, 12:33173–33189, 2024.
- [38] Nithya Sambasivan, Shivani Kapania, Hannah Highfill, Diana Akrong, Praveen Paritosh, and Lora M Aroyo. “everyone wants to do the model work, not the data work”: Data cascades in high-stakes AI. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, New York, NY, USA, May 2021. ACM.
- [39] Dakuo Wang, Justin D Weisz, Michael Muller, Parikshit Ram, Werner Geyer, Casey Dugan, Yla Tausczik, Horst Samulowitz, and Alexander Gray. Human-AI collaboration in data science: Exploring data scientists’ perceptions of automated AI. *Proc. ACM Hum.-Comput. Interact.*, 3(CSCW):1–24, November 2019.
- [40] Jaimie Drozdal, Justin Weisz, Dakuo Wang, Gaurav Dass, Bingsheng Yao, Changruo Zhao, Michael Muller, Lin Ju, and Hui Su. Trust in AutoML: exploring information needs for establishing trust in automated machine learning systems. In *Proceedings of the 25th International Conference on Intelligent User Interfaces*, IUI ’20, pages 297–307, New York, NY, USA, March 2020. Association for Computing Machinery.
- [41] Namita Agarwal and Saikat Das. Interpretable machine learning tools: A survey. In *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1528–1534. IEEE, December 2020.
- [42] Julia Moosbauer, Julia Herbing, Giuseppe Casalicchio, Marius Lindauer, and Bernd Bischl. Explaining hyperparameter optimization via partial dependence plots. *Advances in Neural Information Processing Systems*, 34:2280–2291, December 2021.
- [43] Tijn De Bie, Luc De Raedt, José Hernández-Orallo, Holger H Hoos, Padhraic Smyth, and Christopher K I Williams. Automating data science. *Commun. ACM*, 65(3):76–87, February 2022.
- [44] Ken Gu, Madeleine Grunde-McLaughlin, Andrew McNutt, Jeffrey Heer, and Tim Althoff. How do data analysts respond to AI assistance? a wizard-of-oz study. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, New York, NY, USA, May 2024. ACM.
- [45] Shujian Zhang, Chengyue Gong, Lemeng Wu, Xingchao Liu, and Mingyuan Zhou. AutoML-GPT: Automatic machine learning with GPT. *arXiv [cs.CL]*, May 2023.
- [46] W Nicholson Price, 2nd and I Glenn Cohen. Privacy in the age of medical big data. *Nat. Med.*, 25(1):37–43, January 2019.
- [47] OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, Irwan Bello, Jake Berdine, Gabriel Bernadett-Shapiro, Christopher

Berner, Lenny Bogdonoff, Oleg Boiko, Madelaine Boyd, Anna-Luisa Brakman, Greg Brockman, Tim Brooks, Miles Brundage, Kevin Button, Trevor Cai, Rosie Campbell, Andrew Cann, Brittany Carey, Chelsea Carlson, Rory Carmichael, Brooke Chan, Che Chang, Fotis Chantzis, Derek Chen, Sully Chen, Ruby Chen, Jason Chen, Mark Chen, Ben Chess, Chester Cho, Casey Chu, Hyung Won Chung, Dave Cummings, Jeremiah Currier, Yunxing Dai, Cory Decareaux, Thomas Degry, Noah Deutsch, Damien Deville, Arka Dhar, David Dohan, Steve Dowling, Sheila Dunning, Adrien Ecoffet, Atty Eleti, Tyna Eloundou, David Farhi, Liam Fedus, Niko Felix, Simón Posada Fishman, Juston Forte, Isabella Fulford, Leo Gao, Elie Georges, Christian Gibson, Vik Goel, Tarun Gogineni, Gabriel Goh, Rapha Gontijo-Lopes, Jonathan Gordon, Morgan Grafstein, Scott Gray, Ryan Greene, Joshua Gross, Shixiang Shane Gu, Yufei Guo, Chris Hallacy, Jesse Han, Jeff Harris, Yuchen He, Mike Heaton, Johannes Heidecke, Chris Hesse, Alan Hickey, Wade Hickey, Peter Hoeschele, Brandon Houghton, Kenny Hsu, Shengli Hu, Xin Hu, Joost Huizinga, Shantanu Jain, Shawn Jain, Joanne Jang, Angela Jiang, Roger Jiang, Haozhun Jin, Denny Jin, Shino Jomoto, Billie Jonn, Heewoo Jun, Tomer Kaftan, Łukasz Kaiser, Ali Kamali, Ingmar Kanitscheider, Nitish Shirish Keskar, Tabarak Khan, Logan Kilpatrick, Jong Wook Kim, Christina Kim, Yongjik Kim, Jan Hendrik Kirchner, Jamie Kiros, Matt Knight, Daniel Kokotajlo, Łukasz Kondraciuk, Andrew Kondrich, Aris Konstantinidis, Kyle Kosic, Gretchen Krueger, Vishal Kuo, Michael Lampe, Ikai Lan, Teddy Lee, Jan Leike, Jade Leung, Daniel Levy, Chak Ming Li, Rachel Lim, Molly Lin, Stephanie Lin, Mateusz Litwin, Theresa Lopez, Ryan Lowe, Patricia Lue, Anna Makanju, Kim Malfacini, Sam Manning, Todor Markov, Yaniv Markovski, Bianca Martin, Katie Mayer, Andrew Mayne, Bob McGrew, Scott Mayer McKinney, Christine McLeavey, Paul McMillan, Jake McNeil, David Medina, Aalok Mehta, Jacob Menick, Luke Metz, Andrey Mishchenko, Pamela Mishkin, Vinnie Monaco, Evan Morikawa, Daniel Mossing, Tong Mu, Mira Murati, Oleg Murk, David Mély, Ashvin Nair, Reiichiro Nakano, Rajeev Nayak, Arvind Neelakantan, Richard Ngo, Hyeonwoo Noh, Long Ouyang, Cullen O’Keefe, Jakub Pachocki, Alex Paino, Joe Palermo, Ashley Pantuliano, Giambattista Parascandolo, Joel Parish, Emy Parparita, Alex Passos, Mikhail Pavlov, Andrew Peng, Adam Perelman, Filipe de Avila Belbute Peres, Michael Petrov, Henrique Ponde de Oliveira Pinto, Michael, Pokorny, Michelle Pokrass, Vitchyr H Pong, Tolly Powell, Alethea Power, Boris Power, Elizabeth Proehl, Raul Puri, Alec Radford, Jack Rae, Aditya Ramesh, Cameron Raymond, Francis Real, Kendra Rimbach, Carl Ross, Bob Rotsted, Henri Roussez, Nick Ryder, Mario Saltarelli, Ted Sanders, Shibani Santurkar, Girish Sastry, Heather Schmidt, David Schnurr, John Schulman, Daniel Selsam, Kyla Sheppard, Toki Sherbakov, Jessica Shieh, Sarah Shoker, Pranav Shyam, Szymon Sidor, Eric Sigler, Maddie Simens, Jordan Sitkin, Katarina Slama, Ian Sohl, Benjamin Sokolowsky, Yang Song, Natalie Staudacher, Felipe Petroski Such, Natalie Summers, Ilya Sutskever, Jie Tang, Nikolas Tezak, Madeleine B Thompson, Phil Tillet, Amin Tootoonchian, Elizabeth Tseng, Preston Tuggle, Nick Turley, Jerry Tworek, Juan Felipe Cerón Uribe, Andrea Vallone, Arun Vijayvergiya, Chelsea Voss, Carroll Wainwright, Justin Jay Wang, Alvin Wang, Ben Wang, Jonathan Ward, Jason Wei, C J Weinmann, Akila Welihinda, Peter Welinder, Jiayi Weng, Lilian Weng, Matt Wiethoff, Dave Willner, Clemens Winter, Samuel Wolrich, Hannah Wong, Lauren Workman, Sherwin Wu, Jeff Wu, Michael Wu, Kai Xiao, Tao Xu, Sarah Yoo, Kevin Yu, Qiming Yuan, Wojciech Zaremba, Rowan Zellers, Chong Zhang, Marvin Zhang, Shengjia Zhao, Tianhao Zheng, Juntang Zhuang, William Zhuk, and Barret Zoph. GPT-4 technical report. *arXiv [cs.CL]*, March 2023.

- [48] Nicolas P Terry. Protecting patient privacy in the age of big data. *SSRN Electron. J.*, September 2012.

- [49] Jingquan Li. Security implications of AI chatbots in health care. *J. Med. Internet Res.*, 25(1):e47551, November 2023.
- [50] Soroosh Tayebi Arasteh, Tianyu Han, Mahshad Lotfinia, Christiane Kuhl, Jakob Nikolas Kather, Daniel Truhn, and Sven Nebelung. Large language models streamline automated machine learning for clinical studies. *Nat. Commun.*, 15(1):1603, February 2024.
- [51] Data residency in azure. <https://azure.microsoft.com/en-us/explore/global-infrastructure/data-residency>. Accessed: 2024-8-15.
- [52] Llama 3.1. <https://llama.meta.com/>. Accessed: 2024-8-15.
- [53] Beatriz Remeseiro and Veronica Bolon-Canedo. A review of feature selection methods in medical applications. *Computers in Biology and Medicine*, 112:103375, 2019.
- [54] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. *Advances in Neural Information Processing Systems*, 30, 2017.
- [55] Leo Breiman. Random forests. *Machine learning*, 45:5–32, 2001.
- [56] Cem Tekin and Mihaela van der Schaar. Episodic multi-armed bandits. *arXiv [cs.LG]*, August 2015.
- [57] Anaconda Inc. Anaconda software distribution, 2016.
- [58] Python package index - PyPI. <https://pypi.org/>. Accessed: 2021-3-28.
- [59] Azure OpenAI service documentation. <https://learn.microsoft.com/en-us/azure/ai-services/openai/>. Accessed: 2024-8-23.
- [60] openai-python: The official python library for the OpenAI API.
- [61] OpenAI. ChatGPT. <https://openai.com/chatgpt/>. Accessed: 2024-8-29.
- [62] The cambridge AI in medicine summer school. <https://www.vanderschaar-lab.com/cambridge-ai-in-medicine-summer-school/>. Accessed: 2024-5-10.
- [63] Christopher A Longhurst, Karandeep Singh, Aneesh Chopra, Ashish Atreja, and John S Brownstein. A call for artificial intelligence implementation science centers to evaluate clinical effectiveness. *NEJM AI*, 1(8), July 2024.
- [64] Stefan Feuerriegel, Dennis Frauen, Valentyn Melnychuk, Jonas Schweisthal, Konstantin Hess, Alicia Curth, Stefan Bauer, Niki Kilbertus, Isaac S Kohane, and Mihaela van der Schaar. Causal machine learning for predicting treatment outcomes. *Nat. Med.*, 30(4):958–968, April 2024.
- [65] Ahmed Alaa and Mihaela van der Schaar. Deep multi-task gaussian processes for survival analysis with competing risks. *Adv. Neural Inf. Process. Syst.*, pages 2329–2337, 2017.
- [66] Kiri L Wagstaff. Machine learning that matters. In *Proceedings of the 29th International Conference on International Conference on Machine Learning*, ICML’12, pages 1851–1856, Madison, WI, USA, June 2012. Omnipress.

- [67] Robert C Martin. Design principles and design patterns. *Object Mentor*, 1(34):597, 2000.
- [68] ChrisHMSFT. Data, privacy, and security for azure OpenAI service. <https://learn.microsoft.com/en-us/legal/cognitive-services/openai/data-privacy>. Accessed: 2024-9-13.



## 6 Appendix

### 6.1 Software architecture

The software architecture of *ClimB* is founded on the following design principles:

1. **LLM, tool, and environment-agnostic.** *ClimB* is not tied to a particular LLM in its design, set of tools (locally executable, predefined code snippets with a human readable documentation), or code execution environment. The LLM used can be changed via simple extension of the **Engine** class, new tools can be defined as needed, and the code execution environment is compartmentalized, flexible and managed via **conda**. A notable limitation currently is that only Python code execution environments are supported, but this choice is based on the observation that the most extensive set of ML and data science libraries are available primarily in Python.
2. **Modular.** The *ClimB* software design is modular, broadly following the SOLID [67] design principles. The main concepts in the architectures are: **Engine**, which defines how reasoning and action units work via the LLM; **Message**, which stores the details of each interaction step (between the LLM, user, tools, and code execution); **UI** which defines a user interface compatible with the rest of the architecture; and the *code execution environment* logic.
3. **Extensible.** The two preceding points make *ClimB* extensible - the key components of the system can be built upon in a straight-forward fashion. In addition, a structured plan that defines the details of the project to be performed between the LLM and the user can be customized easily, via a JSON configuration file.

### 6.2 Data privacy considerations

The approach to data privacy considerations used in *ClimB* is further elaborated in this section. This approach was chosen in order to ensure a level of data privacy that is compatible with the requirements of certain non-public datasets. The use of each dataset utilized in the experiments reported was approved with the clinician scientist acting as the data steward. In addition, no personally identifiable fields were present in any of the datasets, and all identification reference fields (e.g. database IDs that may correspond to data records of a specific patient, procedure, treatment etc.) were replaced with randomly generated values prior to use with *ClimB*. The following key points characterize *ClimB*'s current approach to data privacy:

1. **Local storage of data.** All dataset files (original or modified in any way by *ClimB*) are stored locally on the user's machine. Hence, the data files are never uploaded to any third-party (LLM provider's, or other) servers.
2. **Local code execution.** All code execution performed by *ClimB*, either through code generation or predefined tool invocation occurs locally on the user's machine. Hence, no working directory files of any kind (including saved predictive models, image files, tool output artifacts etc.) leave the user's machine.

3. **Use of Azure OpenAI Service as LLM provider.** Azure OpenAI Service, deployed within the researchers' Azure tenancy, was used to access LLM text generation. In addition to customer data ownership control in Azure [51], Azure OpenAI Service has additional data privacy considerations [68]. Namely, the user-LLM text interactions (composed of input prompts and output completions):

- are not available to other Azure customers,
- are not available to OpenAI,
- are not used to improve OpenAI models,
- are not used to train, retrain, or improve Azure OpenAI Service foundation models,
- are not used to improve any Microsoft or third party products or services without the customer's permission or instruction.

While Azure OpenAI may store some data that is necessary for stateful features to function [68], in the case of *CliMB* this is minimal; while it may apply to the functionality of tool calling, in the form of the temporary tool call IDs, which we use, we do not utilize the major stateful features such as the Assistant API, Files API, or vector store. Furthermore, any data that is stored:

- is stored at rest in the Azure OpenAI resource in the customer's Azure tenant, within the same geography as the Azure OpenAI resource (for all datasets used in the experiments, the geography was set to correspond to that of the dataset and the clinician scientist);
- was double encrypted at rest, by default with Microsoft's AES-256 encryption;
- can be deleted by the customer at any time.

The Azure subscriptions hosting the OpenAI resources used for the experiments have also been set up under "modified abuse monitoring" exception [68], which additionally entirely prevents temporary storage of the prompt history and ensures no human review of the data.

### 6.3 Structured plan

This section provides additional details of the structured plan used in *CliMB* in order to improve project planning robustness. The plan is displayed in Table 7. In addition to the task and subtask names, the plan contains a more detailed description for each. The “subtask selection” field indicates if the subtask is to be issued always (“mandatory”), or depending on certain conditions checked against the project memory (“conditional”) – conditional subtasks have a corresponding condition specification, e.g. “applicable to survival analysis”.

Project stage	Task name	Subtask name	Subtask selection
Alignment check	Getting the environment ready	Upload data file Check hardware Check data file can be loaded	mandatory mandatory mandatory
	Getting information from the user	High-level information from the user Experiment setup and research question from the user Assess data suitability and tool support	mandatory mandatory mandatory
Data exploration	Exploratory Data Analysis	Exclude/keep columns Perform EDA Generate descriptive statistics Warn about small sample size if necessary Show Kaplan-Meier plot	mandatory mandatory mandatory conditional mandatory
Data engineering	Data processing - before missing	Column background information	mandatory
	Data processing - missing data handling	Represent missing data as NaN Consider dropping columns with high missing values Consider dropping rows with missing values Impute missing values	mandatory mandatory mandatory conditional
	Data processing - after missing	Discuss data preprocessing with the user Feature selection	mandatory mandatory
	Machine learning study pre-checks	Confirm ML problem type Check time and event columns Check for data leakage Check for irrelevant columns	mandatory conditional mandatory mandatory
Model building	Machine learning study	Machine learning study - classification Machine learning study - regression Machine learning study - survival analysis	conditional conditional conditional
	ML iteration	Iterate the machine learning study with the user	mandatory
Model exploitation	Machine learning study exploitation	Feature importance plots Insights on classification Subgroup analysis	mandatory conditional mandatory
End of Study	Finish up the project	Discuss the project and finish up	mandatory

Table 7: **Structured project plan.** This table summarizes all stages, tasks, and subtasks.

## 6.4 Illustration of the planning process

CliMB’s reasoning unit coordinates the project flow through iterative execution of the underlying subtasks. The structured plan defined for the project provides “checkpoints” that represent *guardrails* (consider, “can we proceed to the next stage of the workflow given the current state of the data?”) and *best practices* (“the recommended approach in situation *X* is *Y*”).

This is well-illustrated through the scenario shown in Fig 7. Given that the “model building” project stage requires, among other points, that there are no missing values present, the plan for the “data engineering” stage contains the appropriate missing data handling checkpoint. This manifests itself as an evaluation of the dataset for missing values and execution of several subtask actions (all informed by user-guided decisions through feedback): column-wise missingness evaluation, row-wise missingness evaluation, investigation of the target variable’s missing values, and imputation. On the output of this process, the dataset is devoid of missing values, and it is safe to proceed to the subsequent “model building” work.

In the absence of this approach, we find that the stochastic nature of LLMs leads to frequent inconsistencies in the state of the project as it progresses (including the dataset, model, and other files in the working directory, as well as the actual tasks undertaken), and consequently machine learning project best practices are sometimes not followed.

### CliMB structured planning

– ensures the requirements of each stage in the workflow are met: serves as *guardrails* and ensures *best practices*

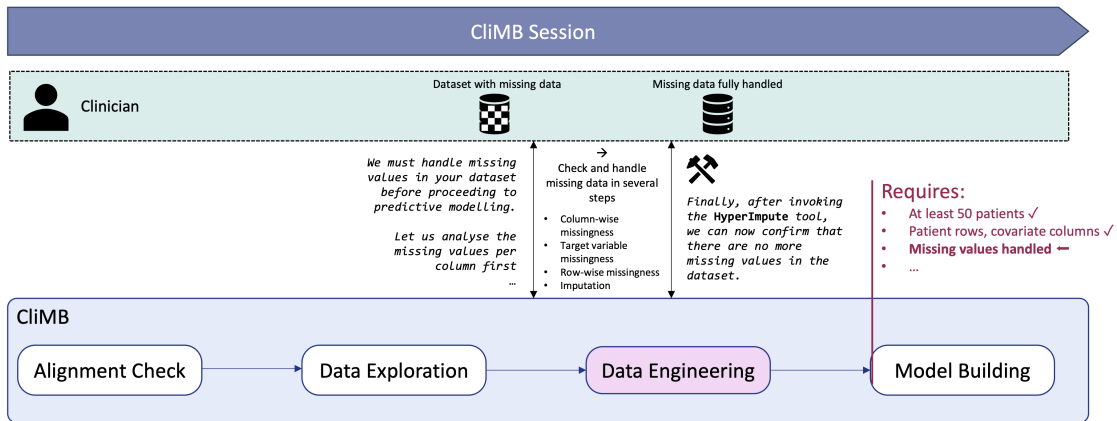


Figure 7: *CliMB* planning checkpoint illustration. Illustration of the “data engineering” stage’s checkpoint within the project flow. The reasoning unit ensures the dataset is evaluated for missing values before transitioning to the “model building” stage. The checkpoint involves several subtasks that ensure the dataset is prepared and complete for subsequent steps in the workflow.

## 6.5 Interaction log examples

### 6.5.1 Baseline GPT-4 failure cases

#### Example 1: incorrect missing value handling

In the following example we see the failure case of baseline GPT-4 deciding to drop all missing values in order to avoid a code execution error (ultimately caused by missing values not being supported by the model to be fitted). This resulted in dropping the vast majority of the rows, leaving only 14 samples. The failure was not noticed by the system.

---

**Summary of work up to this point:** An initial EDA was carried out, including some descriptive statistics, a correlogram; imputation of columns more than 80% missing data was carried out; a number of columns were removed by user request. The dataset at this point has 39 columns and 1200 rows.

...

**Assistant:** Would you like to start with Linear Regression, or do you have a preference for another regression algorithm?

**User:** Let's start with that. I do not have data science expertise so I will need your guidance here.

**Assistant:** Absolutely, starting with Linear Regression is a sensible choice for its simplicity and interpretability. As we proceed, I'll guide you through each step, explaining the process and decisions along the way.

Here's the plan for building and evaluating a Linear Regression model:

1. Model Training: We'll use the training dataset (`X_train.csv` and `y_train.csv`) to train the Linear Regression model.
2. Model Evaluation: After the model is trained, we'll use the testing dataset (`X_test.csv` and `y_test.csv`) to evaluate the model's performance. We'll look at metrics like RMSE (Root Mean Squared Error), which tells us how far off our predictions are from the actual values, on average; and ( $R^2$ ) (Coefficient of Determination), which gives us an idea of how well our model explains the variation in the target variable.

Let's start by training the Linear Regression model with our training data.

#### Generated code:

```
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from math import sqrt
```

```
# Loading the training data
X_train = pd.read_csv('X_train.csv')
y_train = pd.read_csv('y_train.csv')

# Initializing the Linear Regression model
model = LinearRegression()

# Training the model with the training data
model.fit(X_train, y_train)

# Saving the trained model
import joblib
joblib.dump(model, 'linear_regression_model.pkl')

print("Linear Regression model trained and saved as 'linear_regression_model.pkl'")
```

### Code execution output:

Code execution failed ❌

```
Traceback (most recent call last):
[...]
ValueError: could not convert string to float: 'Bw6'
```

**Assistant:** It appears we encountered an issue during the model training step because the dataset contains categorical variables, which Linear Regression cannot handle directly. Before training the model, it's essential to convert these categorical variables into a form that the model can process. One common approach is one-hot encoding, which converts categorical variables into a format of binary columns.

Here's how we can adjust our approach:

1. Preprocess the Data: Perform one-hot encoding on categorical variables.
2. Re-train the Model: With our data now fully numeric, we can train the Linear Regression model again.

Let's start by preprocessing the training data with one-hot encoding for the categorical variables and then proceed to train the model.

### Generated code:

```
import pandas as pd
from sklearn.compose import ColumnTransformer
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import train_test_split
from math import sqrt
```

```
import joblib

# Load the data
data = pd.read_csv('data_filtered.csv')

# Identify categorical columns
categorical_columns = data.select_dtypes(include=['object']).columns

# Preprocessing: One-hot encoding for categorical variables
preprocessor = ColumnTransformer(
    transformers=[
        ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_columns)
    ],
    remainder='passthrough')

# Define the model
model = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('regressor', LinearRegression())
])

# Split the data into features (X) and target variable (y)
X = data.drop(columns=['eGFR12'])
y = data['eGFR12']

# Splitting the data back into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
                                                    random_state=42)

# Fit the model
model.fit(X_train, y_train)

# Save the trained model
joblib.dump(model, 'linear_regression_model.pkl')

print("Linear Regression model trained with one-hot encoding and saved as '
      linear_regression_model.pkl'.")
```

### Code execution output:

Code execution failed **X**

```
Traceback (most recent call last):
[...]
ValueError: Input X contains NaN.
LinearRegression does not accept missing values encoded as NaN natively. For
supervised learning, you might want to consider sklearn.ensemble.
HistGradientBoostingClassifier and Regressor which accept missing values
encoded as NaNs natively. Alternatively, it is possible to preprocess the
data, for instance by using an imputer transformer in a pipeline or drop
samples with missing values. See https://scikit-learn.org/stable/modules/
impute.html You can find a list of all estimators that handle NaN values at
the following page: https://scikit-learn.org/stable/modules/impute.html#
estimators-that-handle-nan-values
```

**Assistant:** The code execution encountered an issue due to the presence of NaN values in the data. This suggests that some steps in the data preprocessing phase were not fully addressed, particularly in dealing with all missing values before model training.

Given this situation, the most straightforward path forward involves ensuring that our dataset is free of any NaN values and that categorical variables are properly encoded. Since our initial approach to preprocessing within the model pipeline encountered challenges, I suggest we take a step back and manually preprocess the dataset to address NaN values and encode categorical variables before fitting the model again.

First, we'll handle NaN values across the entire dataset and encode categorical variables using one-hot encoding. After these preprocessing steps, we'll proceed with model training again.

Let's start by preprocessing the dataset.

#### Generated code:

```
import pandas as pd

# Load the filtered dataset
data = pd.read_csv("data_filtered.csv")

# Drop rows with any NaN values to ensure clean data for modeling
data_cleaned = data.dropna()

# Generating one-hot encodings for categorical variables
data_cleaned = pd.get_dummies(data_cleaned)

# Saving the preprocessed data
data_cleaned.to_csv("data_cleaned_preprocessed.csv", index=False)

print("Data cleaned and preprocessed. Saved as 'data_cleaned_preprocessed.csv'.")
```

Note the `df.dropna()` in the above generated code snippet causes this problem. The dataset at this point has 39 columns and only 14 rows.

#### Code execution output:

Code execution finished successfully ✓

```
Data cleaned and preprocessed. Saved as 'data_cleaned_preprocessed.csv'.
```

...

**Summary of work after this point:** The system and user moved on to feature importance analysis without realising the significant information loss.



**Example 2: failure to fully complete planned work**

In the following example, we observe the system failing to fully complete its intended work steps. In particular, during exploratory data analysis, an attempt to generate a correlation matrix is made, but abandoned due to an exception encountered. In addition, while the the work plan initially intends to perform feature selection, this ends up boiling down to simply listing the categorical and numeric variables and using them in the following predictive study stage, rather than using any feature selection method, or querying the user's domain knowledge.

---

**Summary of work up to this point:** The user's file was loaded, the columns were listed. The user then queried what the system suggests for next steps.

...

**User:** What do you suggest?

**Assistant:** Given the dataset and typical steps in a medical research project, I recommend proceeding with the following steps in order:

- **Data Cleaning:** To handle missing values, which is crucial for ensuring the quality of the data for analysis. This includes deciding whether to impute missing values based on the distribution of the data or to drop columns/rows with a high percentage of missing values.
- **Exploratory Data Analysis (EDA):** To get a deeper understanding of the data, including the distribution of key variables, potential correlations between variables, and identifying any outliers. This step can help in formulating hypotheses for your study.
- **Feature Selection:** To identify the most relevant variables for the predictive modeling. This can be based on the results of the EDA, domain knowledge, or using statistical methods to find the most predictive features.
- **Preparing for Machine Learning:** This includes encoding categorical variables (if any), normalizing/scaling numeric variables to ensure that no variable dominates others due to its scale, and splitting the data into training and test sets.
- **Machine Learning:** Based on your research question, choosing a predictive task (classification, regression, survival analysis) and applying appropriate machine learning models. This also includes evaluating model performance and possibly iterating over different models or tuning parameters to improve results.

Would you like to start with Data Cleaning, move directly to Exploratory Data Analysis, or do you have another preference?

**User:** Let's do as you suggested above.

...

The system then proceeds with working with the user on removing highly missing columns user and imputing the missing target variable cases.

...

**Assistant:** The missing values in the `eGFR12` column have been imputed with the median, and columns with more than 50% missing values have been removed. The dataset is now saved as `kidney_prepared.csv`, ready for further analysis and modeling.

Next steps could include:

- Exploratory Data Analysis (EDA): To further understand the distribution of the `eGFR12` variable, relationships between features, and identify any potential outliers.
- Feature Selection: Identifying which variables are most relevant to predicting `eGFR12`.
- Preparing for Machine Learning: Encoding categorical variables, normalizing/scaling numeric variables, etc.
- Machine Learning: Building and evaluating a predictive model for `eGFR12`.

Would you like to proceed with Exploratory Data Analysis, or would you prefer to move directly into feature selection or model building?

**User:** Yes, do that (EDA)

**Assistant:** Let's proceed with Exploratory Data Analysis (EDA) to better understand the dataset, especially focusing on the `eGFR12` variable, which is our target for prediction. EDA will help us to identify patterns, detect outliers, and understand the relationship between different variables.

For this EDA, we will:

1. Describe the dataset statistically, including the central tendency, dispersion, and shape of the dataset's distribution.
2. Visualize the distribution of `eGFR12` to understand its spread and identify any potential outliers.
3. Visualize correlations between `eGFR12` and other variables to identify potential predictors.

We'll start with the statistical description and distribution visualization of `eGFR12`.

**Generated code:**

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load the prepared data
```

```

data = pd.read_csv("kidney_prepared.csv")

# Statistical description of the dataset
print("Statistical description of the dataset:")
print(data.describe())

# Distribution of eGFR12
plt.figure(figsize=(10, 6))
sns.histplot(data['eGFR12'], kde=True)
plt.title('Distribution of eGFR12')
plt.xlabel('eGFR12')
plt.ylabel('Frequency')
plt.savefig('eGFR12_distribution.png')
plt.close()

# Correlation matrix heatmap
plt.figure(figsize=(12, 10))
correlation_matrix = data.corr()
sns.heatmap(correlation_matrix, annot=False, cmap='coolwarm')
plt.title('Correlation Matrix Heatmap')
plt.savefig('correlation_matrix.png')
plt.close()

print("Distribution of eGFR12 visualized in 'eGFR12_distribution.png'.")
print("Correlation matrix heatmap saved as 'correlation_matrix.png'.")

```

### Code execution output:

Code execution failed ✖

```

Statistical description of the dataset:
      DONOR_ID  DAGE  ...  PAST_DIABETES  PAST_HYPERTENSION
count  1200.000000  1200.000000  ...    1162.000000         1153.000000
mean    64055.581667    44.575833  ...         1.036145         1.193408
std     13818.714125    16.002531  ...         0.186730         0.395142
min     47359.000000     5.000000  ...         1.000000         1.000000
25%     54447.250000    34.000000  ...         1.000000         1.000000
50%     59765.500000    47.000000  ...         1.000000         1.000000
75%     69865.250000    57.000000  ...         1.000000         1.000000
max     111574.000000    77.000000  ...         2.000000         2.000000

[8 rows x 46 columns]

Traceback (most recent call last):
  [...]
ValueError: could not convert string to float: 'A2'

```

**Assistant:** The statistical description of the dataset was successful, providing insights into the central tendencies and dispersions of numerical variables. However, generating the correlation matrix heatmap failed due to the presence of non-numeric (string) data in the dataset, which cannot be directly correlated using the method attempted.

Given this, we successfully visualized the distribution of `eGFR12`, which should help in understanding its spread and identifying any potential outliers. The distribution visualization has been saved as `eGFR12.distribution.png`.

Moving forward, we can:

- Focus on numerical features for correlation analysis with `eGFR12`, excluding non-numeric columns to avoid errors.
- Proceed with feature selection and preparation for machine learning, considering only the variables relevant for predicting `eGFR12`.

Would you like to attempt a revised correlation analysis excluding non-numeric columns, or would you prefer to move on to the next steps in the analysis?

**User:** Proceed to with the option you think makes more sense

Here we notice that the system moves on to the next steps without fixing the correlation matrix, citing the error encountered as the reason.

**Assistant:** Given the goal of predicting `eGFR12` and the error encountered with non-numeric data in the correlation analysis, it makes sense to proceed with feature selection and preparation for machine learning, focusing on variables relevant for predicting `eGFR12`. This approach will allow us to concentrate on preparing the data for modeling without further delay.

Steps for proceeding:

1. Feature Selection: Identify numerical variables and potentially relevant categorical variables for predicting `eGFR12`. We'll consider variables that are likely to influence kidney function based on domain knowledge or previous studies.
2. Data Preparation for Machine Learning:
  - Encode Categorical Variables: Convert categorical variables into a format that can be provided to machine learning models, using techniques such as one-hot encoding.
  - Normalize/Scale Numerical Variables: Ensure numerical variables are on a similar scale to improve model performance and convergence.

Let's start with feature selection, focusing on numerical variables and identifying key categorical variables for encoding. We'll prepare a list of features to be included in the model based on their potential relevance to `eGFR12`.

**Generated code:**

```
import pandas as pd

# Load the prepared data
data = pd.read_csv("kidney_prepared.csv")
```

```
# Identify numerical columns for potential inclusion
numerical_cols = data.select_dtypes(include=['int64', 'float64']).columns.tolist()

# Identify categorical columns for potential inclusion and encoding
categorical_cols = data.select_dtypes(include=['object']).columns.tolist()

# Print identified columns
print("Numerical Columns:")
print(numerical_cols)
print("\nCategorical Columns:")
print(categorical_cols)

# Note: This is an initial identification step. Further selection may be based
# on domain knowledge, correlation with 'eGFR12', or other statistical
# analyses.
```

### Code execution output:

Code execution failed ❌

```
Numerical Columns:
['DONOR_ID', 'DAGE', 'DSEX', [...], 'ORGANS_TXD', 'PAST_DIABETES', '
  PAST_HYPERTENSION']

Categorical Columns:
['Donor_First_A_Broad', 'Donor_Second_A_Broad', 'Donor_Second_A_Split', [...], '
  Recip_First_DQ_Split', 'Recip_Second_DQ_Broad', 'recip_first_Bw']
```

We see that here, despite the system outlining that “this is an initial identification step” for feature selection, it proceeds directly to the dataset preparation without further action.

**Assistant:** The dataset contains a mix of numerical and categorical columns. Given our goal of predicting eGFR12, which is a numerical variable, we’ve identified the following columns for potential use in our predictive model:

- Numerical Columns: These include donor and recipient demographic information, medical history, and pre- and post-transplant measurements. These variables are directly usable in regression models after scaling.
- Categorical Columns: These primarily consist of genetic markers and other categorical medical data. These will require encoding to be used in predictive modeling.

Next Steps:

1. Encode Categorical Variables: We will convert categorical variables into a format suitable for machine learning models. Given the nature of the categorical data, mainly consisting of genetic markers, one-hot encoding is a suitable approach as it transforms categorical data into a format that can be provided to ML models without imposing ordinality.

2. Normalize/Scale Numerical Variables: This step is crucial to ensure that all numerical variables contribute equally to the model's predictions. We'll scale the numerical variables to have a mean of 0 and a standard deviation of 1.
3. Split the Data: Before training models, we'll split the data into training and test sets to evaluate the model's performance on unseen data.

Let's start with encoding the categorical variables and scaling the numerical variables. This preparation will finalize the dataset for machine learning.

#### Generated code:

```

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
import pandas as pd

# Load the prepared data
data = pd.read_csv("kidney_prepared.csv")

# Separate features and target variable
X = data.drop('eGFR12', axis=1)
y = data['eGFR12']

# Encoding categorical variables
categorical_cols = ['Donor_First_A_Broad', 'Donor_Second_A_Broad', '
Donor_First_B_Broad', 'Donor_Second_B_Broad', 'Donor_First_C_Broad', '
Donor_First_DR_Broad', 'Donor_Second_DR_Broad', 'Donor_First_DQ_Broad', '
Donor_Second_DQ_Broad', 'Recip_First_A_Broad', 'Recip_Second_A_Broad', '
Recip_First_B_Broad', 'Recip_Second_B_Broad', 'Recip_First_DR_Broad', '
Recip_Second_DR_Broad', 'Recip_First_DQ_Broad', 'Recip_Second_DQ_Broad']
encoder = OneHotEncoder(sparse=False, handle_unknown='ignore')
X_encoded = pd.DataFrame(encoder.fit_transform(X[categorical_cols]))
X_encoded.columns = encoder.get_feature_names(categorical_cols)
X = X.drop(categorical_cols, axis=1).reset_index(drop=True)
X = pd.concat([X, X_encoded], axis=1)

# Scaling numerical variables
scaler = StandardScaler()
numerical_cols = X.select_dtypes(include=['int64', 'float64']).columns
X[numerical_cols] = scaler.fit_transform(X[numerical_cols])

# Splitting the dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Save the processed data
X_train.to_csv('X_train.csv', index=False)
X_test.to_csv('X_test.csv', index=False)
y_train.to_csv('y_train.csv', index=False)
y_test.to_csv('y_test.csv', index=False)

print("Categorical variables encoded, numerical variables scaled, and data split
into training and test sets.")

```

...

**Summary of work after this point:** The system then proceeds to work with the user on model building and utilization, without returning to feature selection.

**Example 3: failure to check for data leakage**

In this example, we observe that baseline GPT-4 tends not to pick up on the presence of likely data leakage (or, indeed, the identifier) columns, as it lacks a structured set of checks in the workflow that would ensure this.

---

**Summary of work up to this point:** The user's data was uploaded, an exploratory data analysis performed, and missing values were handled. No discussion of the features that should be removed from the outset was initiated by the system, so the set of features being used at this point is large, at 274.

...

**Assistant:** The dataset now has 274 columns after encoding categorical variables and imputing missing values for both continuous and categorical variables. This transformation ensures that the dataset is ready for regression analysis, with all variables in a format suitable for machine learning models.

The next steps in our analysis will be:

1. **Feature Selection:** We'll identify the most relevant features for predicting eGFR12. This can be done through various methods such as correlation analysis, or more sophisticated techniques like feature importance from tree-based models. Given the expanded feature set post-encoding, this step is crucial to simplify the model and potentially improve its performance.
2. **Model Training:** We'll select a regression model to train on our dataset. Common choices include linear regression for its simplicity and interpretability, or tree-based models like Random Forest or Gradient Boosting for potentially better performance at the cost of some interpretability.
3. **Model Evaluation:** We'll evaluate the model using appropriate metrics. For regression tasks, common metrics include Mean Absolute Error (MAE), Mean Squared Error (MSE), and R-squared ( $R^2$ ).

Would you like to proceed with feature selection next, or do you have any specific preferences for the model training step?

**User:** Yes

**Assistant:** Great! Let's proceed with feature selection to identify the most relevant features for predicting eGFR12. Given the large number of features created after one-hot encoding, feature selection will help us reduce the dimensionality of the dataset and potentially improve the model's performance by focusing on the most informative variables.



A straightforward and effective approach for feature selection in regression tasks is to use a tree-based model, like a Random Forest, to assess feature importance. Random Forests are useful for this purpose because they provide a measure of how much each feature contributes to the prediction accuracy, without requiring the data to be linearly separable or normally distributed.

Here's the plan:

1. Train a basic Random Forest regressor on the dataset.
2. Use the feature importances provided by the model to select a subset of the most important features.

Let's start by training a Random Forest regressor and obtaining the feature importances.

### Generated code:

*Dependencies:*

```
pandas  
sklearn
```

```
import pandas as pd  
from sklearn.ensemble import RandomForestRegressor  
from sklearn.model_selection import train_test_split  
  
# Load the encoded dataset  
data = pd.read_csv("preprocessed_data_encoded.csv")  
  
# Separate the features and target variable  
X = data.drop("eGFR12", axis=1)  
y = data["eGFR12"]  
  
# Split the data into training and testing sets  
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,  
                                                    random_state=42)  
  
# Initialize and train the Random Forest regressor  
rf = RandomForestRegressor(n_estimators=100, random_state=42)  
rf.fit(X_train, y_train)  
  
# Get feature importances  
feature_importances = rf.feature_importances_  
  
# Create a DataFrame to view the feature importances  
features_df = pd.DataFrame({'Feature': X.columns, 'Importance':  
                           feature_importances})  
features_df = features_df.sort_values(by='Importance', ascending=False)  
  
# Save the feature importances to a CSV for review  
features_df.to_csv("feature_importances.csv", index=False)  
  
print("Feature importances saved to feature_importances.csv")
```

**Code execution output:**

Code execution failed ✖

```
error: subprocess-exited-with-error
[...]
  The 'sklearn' PyPI package is deprecated, use 'scikit-learn' rather than '
  sklearn' for pip commands.
[...]
note: This is an issue with the package mentioned above, not pip.
```

The system encountered an error in dependency specification but recovered by correcting the dependency name from `sklearn` to `scikit-learn`.

**Code execution output:**

Code execution finished successfully ✔

```
Feature importances saved to feature_importances.csv
```

**Assistant:** The feature importances have been successfully calculated and saved to `feature_importances.csv`. This file contains a list of features ranked by their importance in predicting eGFR12, as determined by the Random Forest regressor.

With this information, we can:

- Select a subset of the most important features for the regression model to simplify the model and potentially improve performance.
- Proceed to model training with the selected features.

Would you like to review the feature importances and select a specific number of top features for the model, or should we proceed with all the features available?

**User:** Yes, review them

**Assistant:** To review the feature importances, I'll generate code to load and display the top features from the `feature_importances.csv` file. This will help us decide how many features we might want to include in our regression model based on their importance scores.

**Generated code:**

```
import pandas as pd

# Load the feature importances
feature_importances = pd.read_csv("feature_importances.csv")

# Display the top 20 features by importance
print("Top 20 features by importance:")
print(feature_importances.head(20))
```

**Code execution output:**

Code execution finished successfully ✓

```
Top 20 features by importance:
  Feature      Importance
0      eGFR3      0.506512
1         RAGE      0.170737
2       GSURV      0.027731
3         DAGE      0.024606
4         DBMI      0.020644
5     CIT_MINS      0.015304
6     RECIP_ID      0.013962
7    WAIT_TIME      0.012750
8    DRET_CREAT      0.011237
9       TX_ID      0.008558
10      DCOD      0.008022
11      PRD      0.006450
12      DSEX      0.005084
13    RETHNIC      0.004640
14 Donor_First_C_Broad_Cw7      0.004598
15      DONOR_ID      0.004161
16      CRF_TX      0.002980
17 Recip_First_B_Broad_B40      0.002926
18    organs_dntd      0.002925
19    ORGANS_TXD      0.002867
```

Notice that the columns eGFR3 and GSURV appear as important features, which is unsurprising, since these represent a data leakage for the target eGFR12. The identifier columns such as RECIP.ID are also present.

The system then continues through to the model building stage with the user, using the top 20 features identified here.

...

After the model building stage, the user requested the feature importance values of the final trained model.

**User:** I would like to see feature importances

**Assistant:** To provide insights into which features are most important for predicting eGFR12 according to the optimized Random Forest model, let's extract and display the feature importances from the trained model.

We'll generate code to load the optimized model, extract the feature importances, and then display them in descending order of importance.

**Generated code:**

```
import pandas as pd
import joblib
```

```

# Load the optimized Random Forest model
rf_optimized = joblib.load('rf_optimized_model.joblib')

# Load the dataset to get the feature names
data = pd.read_csv("data_selected_features.csv")
feature_names = data.drop('eGFR12', axis=1).columns

# Extract feature importances from the model
feature_importances = rf_optimized.feature_importances_

# Create a DataFrame to display them
importances_df = pd.DataFrame({'Feature': feature_names, 'Importance':
    feature_importances})
importances_df = importances_df.sort_values(by='Importance', ascending=False)

# Display the feature importances
print(importances_df)

```

### Code execution output:

Code execution finished successfully ✓

Feature	Importance
0	eGFR3 0.370966
1	RAGE 0.182343
3	DAGE 0.114246
12	DSEX 0.047448
2	GSURV 0.047023
4	DBMI 0.038885
7	WAIT_TIME 0.032089
9	TX_ID 0.030701
8	DRET_CREAT 0.030360
6	RECIP_ID 0.030002
5	CIT_MINS 0.027119
10	DCOD 0.022366
11	PRD 0.017126
13	RETHNIC 0.005409
14	Donor_First_C_Broad_Cw7 0.003918

Once again, we notice that in the final trained model, the data leakage columns (eGFR3, GSURV) persist, making the model not practically applicable for new predictions, as these variables are not going to be available for patients at transplant time. Since they were not removed, some ID columns (RECIP\_ID, TX\_ID) are also present, which may represent the model overfitting on the particular set of IDs used in the training data, and likely misleading feature importance values.

...

**Summary of work after this point:** The system then proceeded to work with the user on the final task, that is, subgroup analysis by the DSEX variable.

### 6.5.2 CliMB examples

An example of an entire *CliMB* session can be watched here: <https://youtu.be/76XuR0K3F5Y>. We provide examples below that show the advantages of the advanced reasoning approach and other features of *CliMB*. The below examples are sourced from a different session from the recording and hence there are minor differences.

#### Example 1: Data exploration

This example demonstrates *CliMB*'s approach to getting a detailed and relevant data analysis via feedback from the user and the available tools (EDA and descriptive statistics).

---

**Summary of work up to this point:** The user's file has been loaded, and the alignment of the system's capabilities and the user's intended research question has been confirmed.

...

#### Assistant: Planning step

**Assistant:** To begin our exploratory data analysis, let's first identify all the columns in your dataset. This will allow us to understand the structure of your data better and decide if there are any columns you would like to exclude from the analysis or if you prefer to only keep certain columns.

#### Generated code:

```
import pandas as pd

# Load the dataset
data = pd.read_csv("kidney_train.csv")

# Print the names of all the columns, one per line for clarity
print("List of all columns in the dataset:")
for column in data.columns:
    print(column)
```

#### Code execution output:

Code execution finished successfully ✓

```
List of all columns in the dataset:
DONOR_ID
DAGE
DSEX
... shortened for brevity ...

Recip_Second_DQ_Broad
Recip_Second_DQ_Split
```

```
recip_first_Bw
```

**Assistant:** Given the list of columns in your dataset, would you like to exclude certain columns from the analysis, or would you prefer to only keep certain columns? If so, could you specify which columns you would like to exclude or keep? This will help tailor the dataset to your specific analysis needs.

The system provides an opportunity for the user to review the features in the dataset early in the pipeline, and to select a subset of these as needed. In this example, the clinician scientist required a fairly complex preprocessing step in terms of which columns were to be removed; this was requested and accomplished by the system.

**User:** I want to remove some unnecessary columns. These are: all columns that have a name like <letter>.<number>, all columns that start with Recip\_ or Donor\_, and also remove columns "ALI\_0", "OKI\_0", "recip\_first\_Bw".

**Assistant:** I'll now generate the code to exclude the specified columns from your dataset. This includes removing all columns with names matching the pattern <letter>.<number>, all columns starting with Recip\_ or Donor\_, and specifically the columns ALI\_0, OKI\_0, and recip\_first\_Bw. The modified dataset will be saved with a suffix \_user\_cols in the filename.

#### Generated code:

```
import pandas as pd
import re

# Load the dataset
data = pd.read_csv("kidney_train.csv")

# Define the patterns for columns to be removed
patterns = [r'^[A-Z]_\d$', r'^Recip_', r'^Donor_', r'^ALI_0$', r'^OKI_0$', r'^
            recip_first_Bw$']

# Compile the patterns into regex objects
regexes = [re.compile(pattern) for pattern in patterns]

# Identify columns to remove based on the patterns
columns_to_remove = [column for column in data.columns if any(regex.match(column
    ) for regex in regexes)]

# Remove the identified columns
data_modified = data.drop(columns=columns_to_remove)

# Save the modified dataset
data_modified.to_csv("kidney_train_user_cols.csv", index=False)

print(f"Modified dataset saved as 'kidney_train_user_cols.csv'. Removed {len(
    columns_to_remove)} columns.")
```

**Code execution output:**

Code execution finished successfully ✓

Modified dataset saved as 'kidney\_train\_user\_cols.csv'. Removed 52 columns.

**Tool EDA completed successfully ✓****Tool logs:**

```
Getting dataset basic info...
Getting descriptive statistics for numerical features...
Getting detailed information on categorical variables...
Performing missing values analysis...
Identified numeric columns that should be considered categorical:
['DSEX', 'DETHNIC', 'HYPERTENSION', 'DGRP', 'RSEX', 'RETHNIC', 'GRAFT_NO', '
  TX_YR', 'HLA_MM', 'organ_type', 'REJECTION', 'DGF', 'GCENS', 'COF', 'TX_TYPE
  ', 'organs_dntd', 'ORGANS_TXD', 'PAST_DIABETES', 'PAST_HYPERTENSION']
Performing correlation analysis...
Performing potential outliers identification...
Performing duplicate records analysis...
```

**Tool output:**

```
Dataset Shape: 1200 rows and 36 columns
Column Names and Types:
DONOR_ID          object
DAGE              int64
DSEX              int64
... shortened for brevity ...
ORGANS_TXD        int64
PAST_DIABETES     float64
PAST_HYPERTENSION float64

Descriptive Statistics for Numerical Features:
[shortened here for brevity]
          DONOR_ID          DAGE          DSEX          ...          ORGANS_TXD
          PAST_DIABETES  PAST_HYPERTENSION
count      1200.000000      1200.000000      1200.000000      ...      1200.000000
          1162.000000          1153.000000
mean        64055.581667          44.575833          1.464167      ...          3.371667
          1.036145          1.193408
std         13818.714125          16.002531          0.498922      ...          1.168904
          0.186730          0.395142
[...]
```

Identified numeric value columns that should most likely be considered categoricals:

```
['DSEX', 'DETHNIC', 'HYPERTENSION', 'DGRP', 'RSEX', 'RETHNIC', 'GRAFT_NO', '
  TX_YR', 'HLA_MM', 'organ_type', 'REJECTION', 'DGF', 'GCENS', 'COF', 'TX_TYPE
  ', 'organs_dntd', 'ORGANS_TXD', 'PAST_DIABETES', 'PAST_HYPERTENSION'].
```

This is done by checking whether the column contains only integers and has a low number of unique values (<20 or <5% of total examples).

Detailed Information on Categorical Variables:  
GRAFT\_NO - Unique Values: 4  
Top 5 Values:

```
GRAFT_NO
1      870
2      279
3       48
4        3

TX_TYPE - Unique Values: 1
Top 5 Values:
TX_TYPE
10      1200

RECIP_ID - Unique Values: 1200
Top 5 Values:
RECIP_ID
ZGZWPAG      1
QUUFRYX      1
95WCXRJ      1
NJ96Z1C      1
TYVD858      1
Rare Categories:
RECIP_ID
9IV3SFC      1
I1PX0JP      1
QWJIG75      1
A0IBUOM      1
EMHWBRM      1

... shortened for brevity ...

organ_type - Unique Values: 2
Top 5 Values:
organ_type
12      608
11      592

DGF - Unique Values: 3
Top 5 Values:
DGF
1.0     546
2.0     230
3.0      27

PAST_DIABETES - Unique Values: 2
Top 5 Values:
PAST_DIABETES
1.0     1135
2.0      28

Missing Values Analysis:
DETHNIC           6
HYPERTENSION      79
DRET_CREAT       109
DBMI              59
RETHNIC           16
RBM              855
HLA_MM           1200
CIT_MINS          18
```



```

eGFR3                217
eGFR12               197
REJECTION            72
DGF                  397
GSURV                2
GCENS                 2
COF                  774
RCOD                 1017
PAST_DIABETES        37
PAST_HYPERTENSION    49

Count of columns with all NaN values: 1
Correlation Analysis:

Most Positively Correlated Features:
  Feature 1      Feature 2      Correlation
0  organs_dntd    ORGANS_TXD    0.910547
1    eGFR3        eGFR12    0.879036
2  GRAFT_NO      CRF_TX    0.658768
3  WAIT_TIME     TX_YR    0.446634
4    TX_YR        CRF_TX    0.405210
5  GRAFT_NO      TX_YR    0.395091
6    DAGE         RAGE    0.375703
7    TX_YR        DGF    0.362208
8  WAIT_TIME     CRF_TX    0.358320
9  PAST_DIABETES PAST_HYPERTENSION 0.325089

Most Negatively Correlated Features:
  Feature 1      Feature 2      Correlation
0    TX_YR        GSURV    -0.486558
1    DAGE         eGFR12    -0.460534
2    DAGE         eGFR3    -0.435481
3  HYPERTENSION  DGRP    -0.427371
4  GSURV         GCENS    -0.426957
5    RBMI         RCOD    -0.392979
6    DSEX         eGFR12    -0.369204
7    DSEX         eGFR3    -0.354643
8    RAGE         eGFR12    -0.350899
9    RAGE         eGFR3    -0.345405

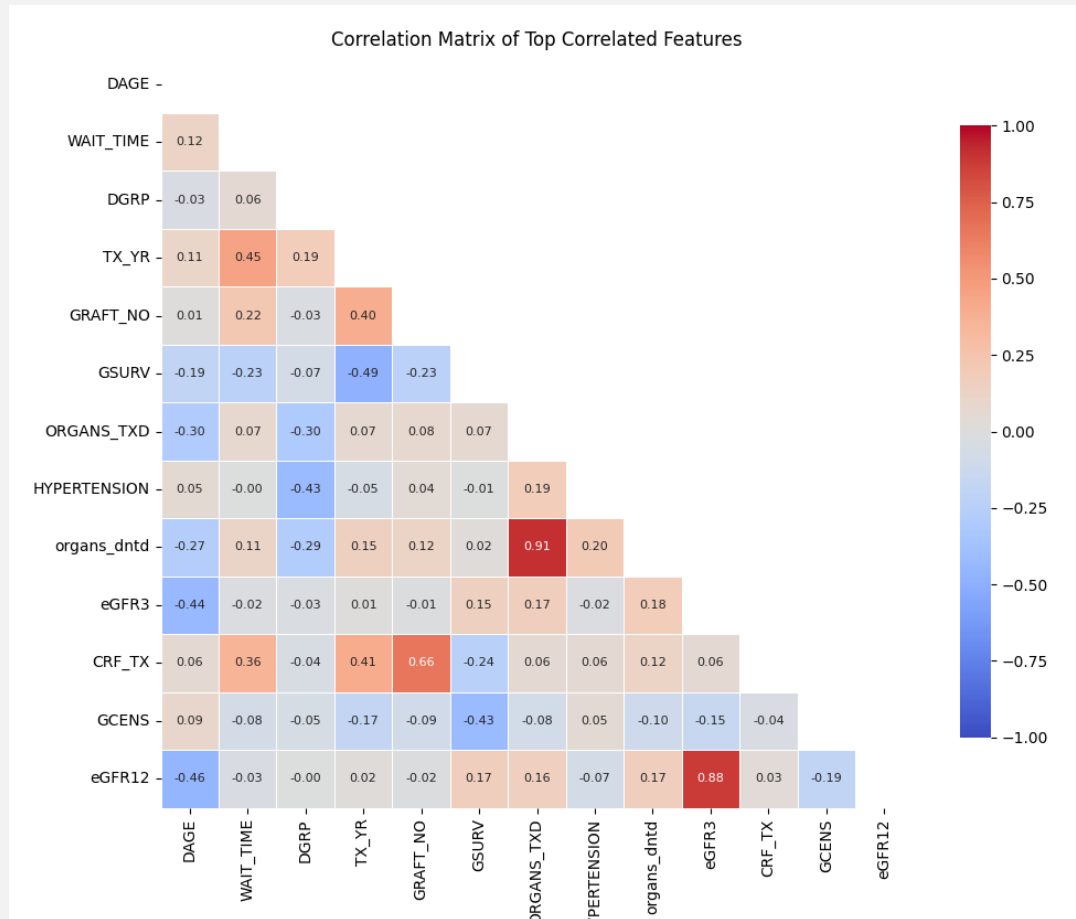
Outlier Identification for Numerical Features:
DAGE - Outliers Count: 0
[Lower Bound: -4.5, Upper Bound: 94.5]
DSEX - Outliers Count: 0
[Lower Bound: -1.75, Upper Bound: 2.75]
DETHNIC - Outliers Count: 41
[Lower Bound: 0, Upper Bound: 0]
... shortened for brevity ...
ORGANS_TXD - Outliers Count: 121
[Lower Bound: 0.25, Upper Bound: 4.75]
PAST_DIABETES - Outliers Count: 65
[Lower Bound: 0, Upper Bound: 0]
PAST_HYPERTENSION - Outliers Count: 275
[Lower Bound: 0, Upper Bound: 0]

Duplicate Records: 0

```

The tool produced the following report:

Here is a correlogram showing the correlation between features:



A correlogram is a visual tool that shows the relationships between different variables (or features) in a dataset. It presents a grid of color-coded squares, where each square represents the strength and direction of the relationship between two variables. Brighter or darker colors indicate stronger relationships. Positive relationships (where variables increase together) and negative relationships (where one variable increases as the other decreases) are shown with different colors. Here we use reds to denote positive, and blue to denote negative relationships. This makes it easy to see which pairs of variables are related, and how closely they are connected.

Notice that *ChMB* tools are able to produce a report that may include text and figures, in order to enhance the user’s understanding of the results of the tool execution.

**Assistant:** The exploratory data analysis (EDA) has provided us with a wealth of information about your dataset. Here are some key insights:

1. **Dataset Overview:** Your dataset consists of 1200 rows and 36 columns after some preprocessing. It includes a mix of numerical and categorical variables.

2. **Numerical Feature Analysis:**

- The descriptive statistics for numerical features suggest varying ranges and distributions, which is typical for clinical data. For example, `DAGE` (donor age) and `RAGE` (recipient age) show a broad range of values, indicating a diverse sample of donors and recipients.
- Some variables like `eGFR3` and `eGFR12` (estimated Glomerular Filtration Rate at 3 and 12 months) are crucial for your analysis as they directly relate to the kidney function post-transplant.

3. **Categorical Variable Analysis:**

- Several categorical variables have been identified, such as `DSEX` (donor sex), `RETHNIC` (recipient ethnicity), and `TX_TYPE`. These variables could play significant roles in your predictive model.
- The dataset includes many unique identifiers (`DONOR_ID`, `RECIP_ID`, `TX_ID`), which are not useful for modeling but essential for data management.

4. **Missing Values Analysis:**

- Certain columns have a significant number of missing values, such as `RBMI` (recipient BMI) with 855 missing values and `RCOD` (reason for coding) with 1017 missing values. This indicates a need for careful handling of missing data.

5. **Correlation Analysis:**

- The most positively correlated features include variables like `organs_dntd` and `ORGANS_TXD`, which is expected. Interestingly, `eGFR3` and `eGFR12` are highly correlated, suggesting that early graft function is predictive of the 12-month outcome.

6. **Outlier Identification:**

- Several features have outliers, which could be genuine variations or errors. For instance, `DRET_CREAT` (donor creatinine) and `CIT_MINS` (cold ischemia time in minutes) show outliers that might need further investigation.

7. **Duplicate Records Analysis:**

- No duplicate records were found, indicating good data integrity.

This analysis has provided a foundational understanding of your dataset's structure, variables, and potential challenges like missing values and outliers. It's a crucial step towards building a robust predictive model.

Would you like to proceed with generating descriptive statistics for a more detailed analysis of each variable, including visualizations?

**User:** Yes, let's see descriptive statistics.

**Tool** descriptive\_statistics completed successfully ✓

**Tool logs:**

```

Creating the descriptive statistics table...
Saving the summary table to:
[...]/session_2024-09-17_18-58-36/kidney_train_user_cols.csv__descriptive_stats.csv
Identified numeric columns that should be considered categorical:
['DSEX', 'DETHNIC', 'HYPERTENSION', 'DGRP', 'RSEX', 'RETHNIC', 'GRAFT_NO', '
TX_YR', 'HLA_MM', 'organ_type', 'REJECTION', 'DGF', 'GCENS', 'COF', 'TX_TYPE',
'organs_dntd', 'ORGANS_TXD', 'PAST_DIABETES', 'PAST_HYPERTENSION']
Creating plots for the data...
Normally distributed features:
['RBMI']
Not normally distributed features:
['CIT_MINS', 'CRF_TX', 'DAGE', 'DBMI', 'DCOD', 'DRET_CREAT', 'GSURV', 'PRD', '
RAGE', 'RCOD', 'WAIT_TIME', 'eGFR12', 'eGFR3']
Plotted a bar plot for: 'COF'
Plotted a bar plot for: 'DETHNIC'
Plotted a bar plot for: 'DGF'
... shortened for brevity ...
Plotted a histogram and box plot for: 'WAIT_TIME'
Plotted a histogram and box plot for: 'eGFR12'
Plotted a histogram and box plot for: 'eGFR3'

```

**Tool output:**

Variable	Summary
DONOR_ID	
WEE6KOG	1/1200 (0.1)
5LC7OU5	1/1200 (0.1)
ZCGNZ7F	1/1200 (0.1)
QHOA7GH	1/1200 (0.1)
FRPIDON	1/1200 (0.1)
Other	1195/1200 (99.6)
DAGE	47.0 (34.0 - 56.0)
DSEX	
1	632/1200 (52.7)
2	568/1200 (47.3)
...	shortened for brevity ...
ORGANS_TXD	
3	563/1200 (46.9)
2	210/1200 (17.5)
4	208/1200 (17.3)

```

 5                98/1200 (8.2)
 6                84/1200 (7.0)
 Other            37/1200 (3.1)
PAST_DIABETES
 1.0             1135/1163 (97.6)
 2.0              28/1163 (2.4)
PAST_HYPERTENSION
 1.0             925/1151 (80.4)
 2.0             226/1151 (19.6)

The following plots have also been created and saved:
- COF: descr__bar_plot__COF.png
- DETHNIC: descr__bar_plot__DETHNIC.png
- DGF: descr__bar_plot__DGF.png
... shortened for brevity ...
- WAIT_TIME: descr__hist_box_plot__WAIT_TIME.png
- eGFR12: descr__hist_box_plot__eGFR12.png
- eGFR3: descr__hist_box_plot__eGFR3.png

```

**The tool produced the following report:**

To view the plots, please select any of the images in the "Working Directory" tab on the right.

**Assistant:** The descriptive statistics have provided a detailed overview of each variable in your dataset. Here's a summary of the key findings:

### 1. Variable Summaries:

- Categorical variables such as DSEX (donor sex), RETHNIC (recipient ethnicity), and TX\_TYPE (transplant type) show a clear distribution of values, which could be important for your analysis.
- Numerical variables such as DAGE (donor age), RAGE (recipient age), and eGFR12 (estimated Glomerular Filtration Rate at 12 months) have been summarized with median and interquartile ranges, providing insights into their distribution.

### 2. Outliers and Missing Values:

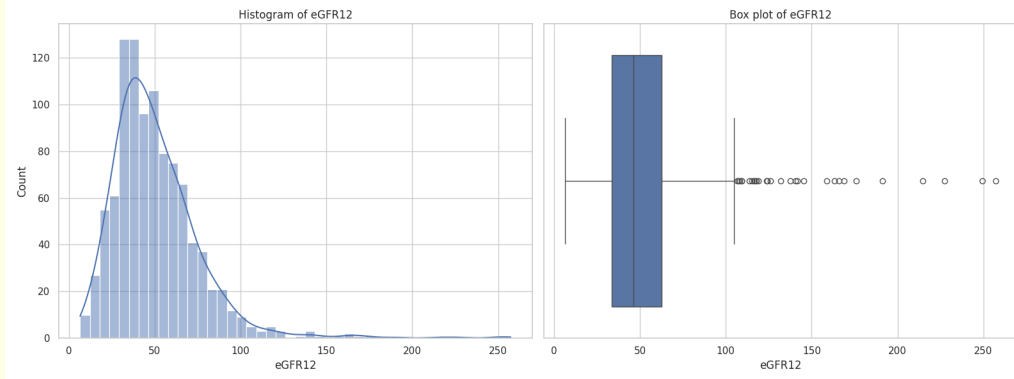
- The analysis has identified potential outliers and missing values in several variables. This is crucial for data cleaning and preprocessing to ensure the quality of your predictive model.

### 3. Plots Generated:

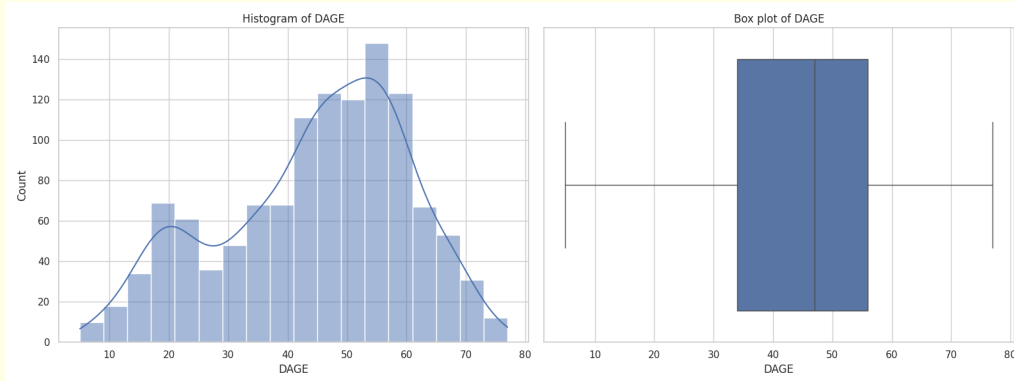
- Various plots have been generated to visually summarize the data, including bar plots for categorical variables and histograms and box plots for numerical variables. These visualizations can help identify patterns, outliers, and distributions in your dataset.

Here are five important plots that you might find useful:

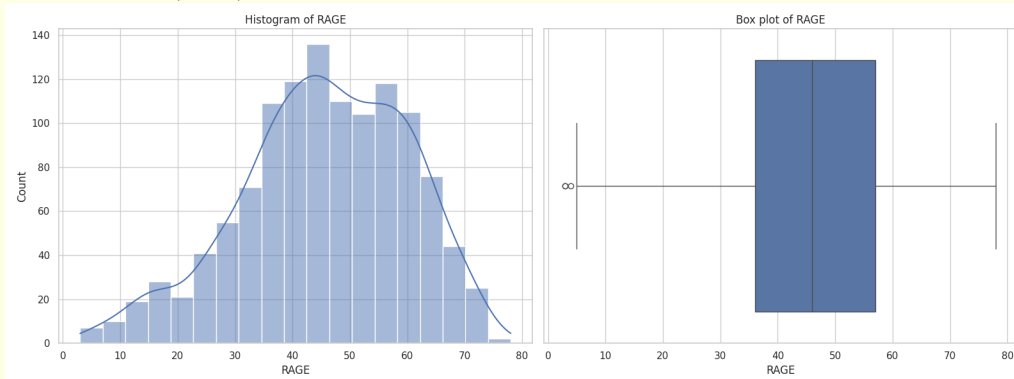
• Estimated Glomerular Filtration Rate at 12 months (eGFR12):



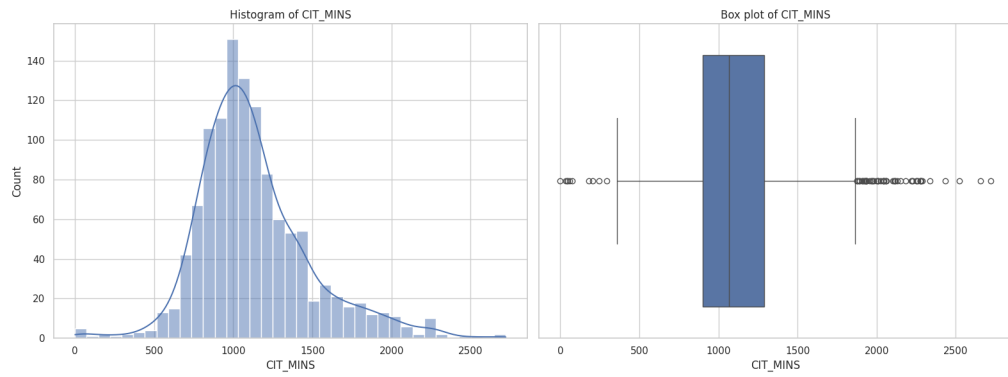
• Donor age (DAGE):



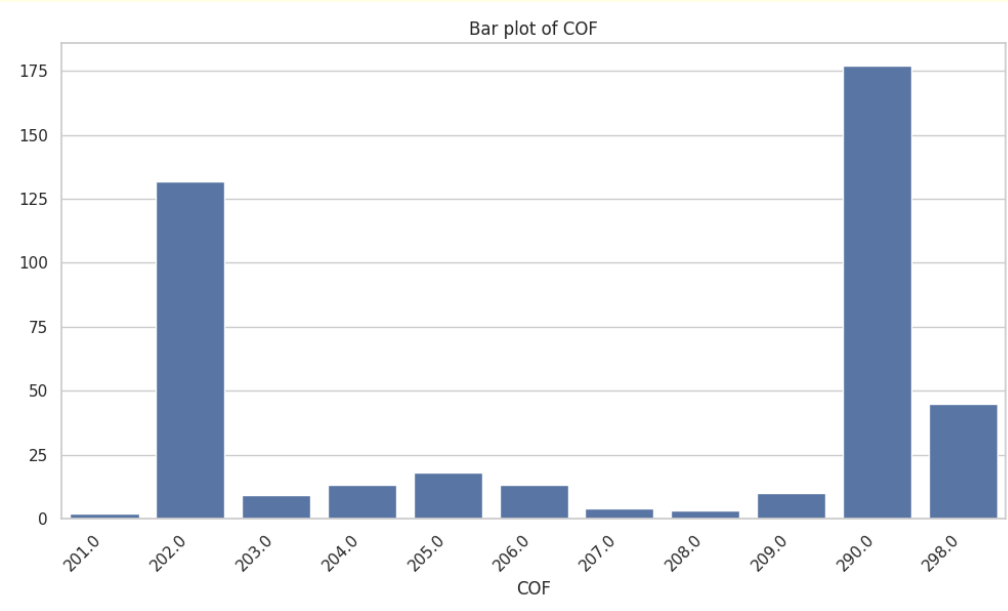
• Recipient age (RAGE):



- Cold Ischemia Time in minutes (CIT\_MINS):



- Cause of Failure (COF):



These plots provide a visual representation of the key variables in your dataset, offering insights into their distribution and potential impact on graft function at 12 months post-transplant.

Please take a moment to review these plots and the descriptive statistics. If you have any questions or need further analysis, feel free to ask.

**User:** That makes sense. Let's continue.

**Assistant:** ✓ Task completed

...

**Summary of work after this point:** The system then proceeded to work with the user on the remaining stages of the project: data engineering, model building, and model exploitation.

### Example 2: Handling of missing values

The following example demonstrates how *CliMB* utilizes a multi-step approach to handling missing data, avoiding common missing data pitfalls, and querying user input when necessary using code generation and available tools.

---

**Summary of work up to this point:** User data has been loaded, the alignment between the system’s capabilities and the user’s intended research question has been confirmed, exploratory data analysis and descriptive statistics generation have been completed, and background information about different columns has been discussed with the user.

At this point, the dataset has 36 columns, 1200 rows, and 5106 (11.82% of total) missing values.

...

#### Assistant: Planning step

**Assistant:** To begin addressing the missing data in your dataset, we’ll first need to represent any missing data as `numpy.nan` values. This standardization is crucial for the subsequent steps of handling missing data effectively.

Based on the EDA performed earlier, we identified several columns with missing values, such as `DETHNIC`, `HYPERTENSION`, `DRET_CREAT`, `DBMI`, `RETHNIC`, `RBMI`, `HLA_MM`, `CIT_MINS`, `eGFR3`, `eGFR12`, `REJECTION`, `DGF`, `GSURV`, `GCENS`, `COF`, `RCOD`, `PAST_DIABETES`, and `PAST_HYPERTENSION`. It was also noted that there are columns with significant amounts of missing data, which might be represented in non-standard formats.

Before proceeding with converting non-standard NaN representations to `numpy.nan`, I’d like to confirm if there are any specific non-standard values used to represent missing data in these columns (e.g., placeholders like “NA”, “unknown”, “-1”, etc.). Could you confirm if such non-standard missing value indicators are present in the dataset?

Note that the workflow allows the system to make reasonable inferences in terms of what placeholders, if any, are used to represent missing values in the dataset; this is done by analysing the relevant project steps completed up to this point (especially exploratory data analysis results). Since in this case, there are in fact no placeholders that represent missing values in the dataset, the system did not suggest any particular placeholders to convert to `numpy.nan`, however, this was first explicitly checked with the user.

**User:** I believe we don’t have such placeholder values in our dataset.



**Assistant:** Given that there are no non-standard placeholders for missing values in your dataset, we can proceed directly with the next steps of handling missing data.

The next step is to generate code to display per-column percentages of missing values in descending order. This will help us identify columns with a high percentage of missing data, which might need to be dropped or require special attention for imputation.

Let's start by displaying the percentage of missing values for each column:

#### Generated code:

```
import pandas as pd
import numpy as np

# Load the dataset
data = pd.read_csv("kidney_train_user_cols.csv")

# Calculate the percentage of missing values for each column
missing_percentage = data.isnull().mean() * 100

# Display the percentages in descending order
missing_percentage_sorted = missing_percentage.sort_values(ascending=False)
print("Percentage of missing values per column, in descending order:")
print(missing_percentage_sorted)

# Save this information into a CSV file for reference
missing_percentage_sorted.to_csv("missing_percentage.csv", header=True)

# Set pandas options to display all rows and columns
pd.options.display.max_rows = None
pd.options.display.max_columns = None
```

#### Code execution output:

Code execution finished successfully ✓

```
Percentage of missing values per column, in descending order:
HLA_MM          100.000000
RCOD            84.750000
RBMI            71.250000
COF            64.500000
DGF            33.083333
eGFR3          18.083333
eGFR12         16.416667
DRET_CREAT      9.083333
HYPERTENSION    6.583333
REJECTION       6.000000
DBMI            4.916667
PAST_HYPERTENSION 4.083333
PAST_DIABETES   3.083333
CIT_MINS        1.500000
RETHNIC         1.333333
DETHNIC         0.500000
GCENS           0.166667
```

```
GSURV                0.166667
organs_dntd          0.000000
TX_TYPE              0.000000
ORGANS_TXD           0.000000
organ_type           0.000000
DONOR_ID             0.000000
CRF_TX               0.000000
DAGE                 0.000000
TX_ID                0.000000
GRAFT_NO             0.000000
WAIT_TIME            0.000000
PRD                  0.000000
RSEX                 0.000000
RAGE                 0.000000
RECIP_ID             0.000000
DCOD                 0.000000
DGRP                 0.000000
DSEX                 0.000000
TX_YR                0.000000
dtype: float64
```

**Assistant:** Based on the percentage of missing values per column, we have identified that the column HLA\_MM has 100% missing values, making it a candidate for removal. Additionally, columns like RCOD, RBMI, COF, and DGF have high percentages of missing values, which may impact the dataset's integrity and the predictive model's performance.

Considering the rule of thumb that columns with 80%+ missing values should generally be removed due to the limited information they can provide, we should consider dropping the HLA\_MM and RCOD columns immediately.

Before proceeding with the removal of these columns or any others with a high percentage of missing values, I'd like to ask:

1. Would you like to set a different threshold for missing values percentage before deciding to remove a column? The default suggestion is 80%.
2. Are there any specific columns with high missing values you would like to keep for further analysis or imputation, despite exceeding the missing values threshold?

Please let me know your preferences regarding these questions, and then we can move forward with the appropriate actions.

**User:** Let us use the 80% missing threshold as you suggest.

**Assistant:** Based on our agreement to use an 80% missing threshold for deciding whether to remove a column, we will proceed to drop the HLA\_MM column due to 100% missing values, and the RCOD column due to 84.75% missing values. This step will help streamline the dataset by removing columns that are unlikely to contribute valuable information to the analysis due to

the high proportion of missing data.

Let's generate the code to remove these columns and save the modified dataset:

#### Generated code:

```
import pandas as pd

# Load the dataset
data = pd.read_csv("kidney_train_user_cols.csv")

# Drop columns with more than 80% missing values
columns_to_drop = ['HLA_MM', 'RCOD']
data_modified = data.drop(columns=columns_to_drop)

# Save the modified dataset
data_modified.to_csv("kidney_train_user_cols_nan.csv", index=False)

print(f"Columns {columns_to_drop} were dropped. Modified dataset saved as '
      kidney_train_user_cols_nan.csv'.")
```

#### Code execution output:

Code execution finished successfully ✓

```
Columns ['HLA_MM', 'RCOD'] were dropped. Modified dataset saved as '
      kidney_train_user_cols_nan.csv'.
```

The system then proceeds to investigating the missingness with respect to rows, and also applies specific focus on the target column values.

**Assistant:** With the columns HLA\_MM and RCOD removed due to high percentages of missing values, we've streamlined the dataset for more effective analysis and modeling. The next step involves considering whether to drop rows with missing values or to use imputation techniques to fill in these gaps.

Given the updated dataset, let's generate code to show:

- The per-column percentage of missing values,
- The percentage of total rows that have missing values.

This information will help us decide how to proceed with rows containing missing data, especially concerning the target variable eGFR12. It's important to handle missing values in the target variable carefully, as imputing these could introduce bias or inaccuracies.

Let's get an overview of the current state of missing data in the dataset:

#### Generated code:

```
import pandas as pd

# Load the modified dataset
data = pd.read_csv("kidney_train_user_cols_nan.csv")

# Calculate the percentage of missing values for each column
missing_percentage = data.isnull().mean() * 100

# Calculate the percentage of rows with any missing values
rows_with_missing = (data.isnull().any(axis=1).mean()) * 100

# Display the percentages in descending order for columns
print("Percentage of missing values per column, in descending order:")
print(missing_percentage.sort_values(ascending=False))

# Display the percentage of rows with any missing values
print(f"\nPercentage of total rows with any missing values: {rows_with_missing}%")

# Set pandas options to display all results
pd.options.display.max_rows = None
pd.options.display.max_columns = None
```

#### Code execution output:

Code execution finished successfully ✓

```
Percentage of missing values per column, in descending order:
RBMI                71.250000
COF                 64.500000
DGF                 33.083333
eGFR3               18.083333
eGFR12              16.416667
DRET_CREAT          9.083333
HYPERTENSION        6.583333
REJECTION            6.000000
DBMI                 4.916667
PAST_HYPERTENSION   4.083333
PAST_DIABETES       3.083333
CIT_MINS             1.500000
RETHNIC              1.333333
DETHNIC              0.500000
GCENS                0.166667
GSURV               0.166667
organ_type          0.000000
TX_TYPE             0.000000
organs_dntd         0.000000
ORGANS_TXD          0.000000
DONOR_ID            0.000000
CRF_TX              0.000000
TX_YR               0.000000
DAGE                0.000000
GRAFT_NO            0.000000
WAIT_TIME           0.000000
PRD                 0.000000
RSEX                0.000000
```

```
RAGE          0.000000
RECIP_ID      0.000000
DCOD          0.000000
DGRP          0.000000
DSEX          0.000000
TX_ID         0.000000
dtype: float64

Percentage of total rows with any missing values: 97.25%
```

**Assistant:** Given the significant amount of missing data across various columns and that 97.25% of the rows contain at least one missing value, it's crucial to carefully consider our next steps to preserve as much valuable data as possible.

For the target variable **eGFR12**, which is essential for your predictive model, it's particularly important to handle missing values judiciously. Imputing the target variable is generally not recommended due to the risk of introducing bias.

Considering the high percentage of rows with missing values, dropping all such rows could lead to a substantial reduction in dataset size, potentially compromising the study's validity. Therefore, imputation might be a preferable approach for columns other than the target variable, especially if the missing data is not overwhelmingly concentrated in a few rows.

Given these considerations:

- Are you okay with dropping rows that have missing values in the target variable **eGFR12**? This step is necessary as imputing the target variable is not recommended.
- For other columns with missing data, would you prefer to attempt imputation to preserve as much data as possible? Imputation can be particularly beneficial if missing data is spread across many rows, minimizing the loss of valuable information.

Please let me know how you'd like to proceed with these two points, and then we can take the appropriate actions.

**User:** I think imputation makes sense for non-eGFR12 columns.

I want to work with the eGFR12 column to prepare it. When the eGFR12 value is missing, then:

1. If the corresponding GSURV value is between 0 and 364 (inclusive) and the corresponding GCENS is 1 then the eGFR12 value should equal 10. This indicates that the graft was lost prior to the endpoint.
2. If the corresponding GSURV value is between 0 and 364 (inclusive) and the corresponding GCENS is 0 then rows with missing eGFR12 value should be deleted. This indicates that the patient died with a functioning graft.

3. Leave the other missing values as they are, those are actually missing.

In this case, the clinician scientist decides to perform a fairly complex, custom handling of missing values in the target column `eGFR12` – which either manually imputes the value, discards the row, or leaves the missing value as is for later imputation, depending on logical conditions in other columns. The system successfully completes this request and then proceeds to the final step in the process, automated imputation.

**Assistant:** Let's proceed with preparing the `eGFR12` column according to the conditions you've outlined. We'll handle the missing `eGFR12` values based on the associated `GSURV` and `GCENS` values, with the specific logic you've provided:

1. If `eGFR12` is missing and `GSURV` is between 0 and 364 (inclusive) and `GCENS` is 1, then set `eGFR12` to 10 to indicate graft loss prior to the endpoint.
2. If `eGFR12` is missing and `GSURV` is between 0 and 364 (inclusive) and `GCENS` is 0, then delete these rows to indicate the patient died with a functioning graft.
3. Leave other missing `eGFR12` values as they are, treating them as genuinely missing.

This approach allows us to handle missing data in a way that respects the clinical context of your study. Let's generate the code to implement this logic:

#### Generated code:

```
import pandas as pd
import numpy as np

# Load the dataset
data = pd.read_csv("kidney_train_user_cols_nan.csv")

# Apply conditions for handling missing eGFR12 values
condition1 = (data['eGFR12'].isnull()) & (data['GSURV'] >= 0) & (data['GSURV']
    <= 364) & (data['GCENS'] == 1)
condition2 = (data['eGFR12'].isnull()) & (data['GSURV'] >= 0) & (data['GSURV']
    <= 364) & (data['GCENS'] == 0)

# Set eGFR12 to 10 under condition 1
data.loc[condition1, 'eGFR12'] = 10

# Drop rows under condition 2
data = data.drop(data[condition2].index)

# Save the modified dataset
data.to_csv("kidney_train_user_cols_nan_prepared.csv", index=False)

print("eGFR12 column prepared and dataset saved as '
    kidney_train_user_cols_nan_prepared.csv'.")
```

**Code execution output:**

Code execution finished successfully ✓

```
eGFR12 column prepared and dataset saved as 'kidney_train_user_cols_nan_prepared.csv'.
```

**Assistant:** With the eGFR12 column now prepared according to the specified conditions and the dataset saved as 'kidney\_train\_user\_cols\_nan\_prepared.csv', we're ready to move forward with the next steps in handling missing data for the rest of the dataset.

Given the considerations and your preference for imputation for non-eGFR12 columns, we'll proceed with discussing and preparing for the imputation process. Imputation is a critical step to ensure that our dataset is as complete as possible, which can significantly improve the performance of the predictive model.

However, before we can invoke any imputation tool, it's essential to ensure there are no entirely empty columns left in the dataset, as they can cause issues during imputation. We've already addressed columns with a high percentage of missing values, so we should be in a good position to proceed.

The next step is to discuss the imputation process itself. Imputation can be done using various methods, ranging from simple approaches like mean or median imputation to more complex ones like model-based imputation. Given the complexity of your dataset and the importance of accurately predicting graft function, a more sophisticated imputation method might be appropriate.

**HyperImpute** is a powerful tool that can handle complex imputation tasks by leveraging multiple imputation models and algorithms to find the best approach for the given data. Before we proceed to use HyperImpute, I'll explain what it does:

- **HyperImpute** automatically explores a range of imputation models and techniques to fill in missing values in the dataset.
- It considers the relationships between variables, the pattern of missingness, and the distribution of data to select the most suitable imputation method.
- This process is aimed at minimizing the bias and error introduced by missing data, making the imputed dataset more reliable for subsequent analysis.

Would you like to proceed with using HyperImpute for imputing the remaining missing values in your dataset, excluding the eGFR12 column which we have already addressed?

**User:** Yes, please impute.

Tool hyperimpute\_imputation completed successfully ✓

Tool logs:

```
Imputing all columns
Setting up HyperImpute Imputer...
Imputing data...

Field "model_name" has conflict with protected namespace "model_".

You may be able to resolve this warning by setting 'model_config['
  protected_namespaces'] = ()'.

Missing values before imputation:
DONOR_ID          0
DAGE              0
DSEX              0
DETHNIC           6
HYPERTENSION      75
DGRP              0
DRET_CREAT        106
DCOD              0
DBMI              56
RECIP_ID          0
RAGE              0
RSEX              0
RETHNIC           16
RBMI              833
PRD               0
WAIT_TIME         0
GRAFT_NO          0
TX_ID             0
TX_YR             0
CIT_MINS          18
CRF_TX            0
organ_type        0
eGFR3             199
eGFR12            64
REJECTION         67
DGF               386
GSURV             2
GCENS             2
COF               759
TX_TYPE           0
organs_dntd       0
ORGANS_TXD        0
PAST_DIABETES     34
PAST_HYPERTENSION 48
dtype: int64

HyperImpute: Iterations completed 1/3
HyperImpute: Iterations completed 2/3
HyperImpute: Iterations completed 3/3
Saving imputation model to: [...]session_2024-09-17_18-58-36/
  hyperimpute__kidney_train_user_cols_nan_prepared_imputed.pkl

Missing values after imputation:
```



```

DONOR_ID      0
DAGE          0
DSEX          0
DETHNIC       0
HYPERTENSION  0
DGRP          0
DRET_CREAT    0
DCOD          0
DBMI          0
RECIP_ID      0
RAGE          0
RSEX          0
RETHNIC       0
RBMI          0
PRD           0
WAIT_TIME     0
GRAFT_NO      0
TX_ID         0
TX_YR         0
CIT_MINS      0
CRF_TX        0
organ_type    0
eGFR3         0
eGFR12        0
REJECTION     0
DGF           0
GSURV         0
GCENS         0
COF           0
TX_TYPE       0
organs_dntd   0
ORGANS_TXD    0
PAST_DIABETES 0
PAST_HYPERTENSION 0
dtype: int64

Saving imputed data...

Tool output:

2671 missing values were imputed. The imputed data has been saved to [...] /
  session_2024-09-17_18-58-36/kidney_train_user_cols_nan_prepared_imputed.csv

```

At this point, the dataset has 34 columns (the two 80%+ missing columns having been removed), 1163 rows (the missing target rows having been removed), and no missing values - as one would expect after a successful missing data imputation step.

...

**Summary of work after this point:** The system then proceeds to work with the user through the machine learning study, and subsequent post-hoc interpretability including feature importance and subgroup analysis by DSEX.

## 6.6 Video demonstrations

The two video demonstrations used in the *Cambridge AI in Medicine Summer School* [62] in order to gather clinician feedback are referenced below.

- **Example 1:** Baseline GPT-4 session. <https://youtu.be/Kyppni0KvoE>,
- **Example 2:** CliMB session. <https://youtu.be/9u411eFPMm8>.

## 6.7 Consent form

Before participating in the polls described in section 3.3, participants were provided with the following information:

### 1. Introduction

You are invited to participate in a research study that aims to understand how an AI-enabled, interactive chatbot can empower clinicians to build their own predictive models. We are particularly interested in your perceptions of the responses provided by this chatbot. Before you decide to participate, it is important that you understand the purpose of this research and what your participation will involve. Please take the time to read the following information carefully.

### 2. What is the purpose of this research?

The primary goal of this research project is to assess the utility of various AI-powered, interactive chatbots designed to assist clinicians like you in building predictive models from clinical datasets. Predictive models can serve multiple purposes. For example, they can help identify patients at high risk for requiring acute care during radiotherapy, enabling clinicians to provide targeted attention. Other models have previously been developed to predict the survival benefits of adjuvant therapy for breast cancer patients. In this study, the chatbot you will see is designed to assist in building such predictive models. However, our focus is on understanding your perception of how the chatbot communicates and displays information. Specifically, we are interested in how you evaluate different tools across several metrics. Please note that you will not be aware of which tool generated which response. This study does not assess your abilities or intelligence; we are solely interested in your opinions on the chatbot responses.

### 3. What data will be collected and how?

During this study, we will collect data related to your preferences. You will be presented with a set of questions that clinicians might ask the chatbot, and you will evaluate the responses generated by two different chatbots. The answers will be anonymized, and you will not know the identity of the chatbot that provided the response. The rating will be done via Zoom's integrated poll feature. If you are unsure how to respond, please select the option that most aligns with your preference. We will ask general questions about your prior experience with ML and your career stage. We will not collect data on demographic information like age or gender, and no sensitive information will be requested.

### 4. How will the data be used?

The data collected will be used to evaluate the effectiveness of different chatbots in assisting clinicians in interactively building predictive models. The aggregated results may be published in

academic journals, presented at conferences, and included in technical reports. Individual responses will remain anonymous and will not be identifiable in any published or presented materials.

#### **5. How and for how long will the data be stored?**

All data collected during the study will be securely stored in an encrypted format on secure servers. Data will be retained for a period of five years after the conclusion of the study, as required by our data retention policy, after which it will be securely deleted.

#### **6. Anonymity of Responses**

Your participation in this study will remain anonymous. No personally identifiable information will be associated with your responses in any reports generated from this research. Data will only be presented in aggregate form.

#### **7. Data Sharing with Other Researchers**

Anonymized, aggregated data may be shared with other researchers at a later date. Again, individual responses will not be identifiable.

#### **8. Withdrawal of Consent and Data**

You have the right to withdraw from the study at any time without providing reasons. If you choose to withdraw, all data associated with your participation will be deleted. Your refusal to participate or later withdrawal from the study will not result in any penalty. To withdraw your consent and request the deletion of your data, please contact [es583@cam.ac.uk](mailto:es583@cam.ac.uk) via email.

#### **9. Legal Framework**

Your data will be handled according to the principles and rules set by the General Data Protection Regulation (GDPR).

#### **10. Consent**

Please confirm that you have read and understood the information provided above regarding your participation in this research study. By clicking the box, you confirm that you:

- Understand the nature and purpose of the study.
- Agree to the collection, use, and storage of your data as described above.
- Understand that your participation is entirely voluntary, and that you may withdraw at any time without penalty.
- Agree to participate in this study.