BLADE: Benchmarking Language Model Agents for Data-Driven Science

Ken Gu[1] Ruoxi Shang[1] Ruien Jiang[2]* Keying Kuang[2]* Richard-John Lin[3]* Donghe Lyu[4]* Yue Mao[5]* Youran Pan[3]* Teng Wu[6]* Jiaqian Yu[7]* Yikun Zhang[1]* Tianmai M. Zhang[1]* Lanyi Zhu[1]* Mike A. Merrill[1] Jeffrey Heer[1] Tim Althoff[1]

[1]University of Washington [2]UC Berkeley [3]New York University [4]Stanford University
[5]University of British Columbia [6]Microsoft [7]George Washington University

https://github.com/behavioral-data/BLADE

Abstract

Data-driven scientific discovery requires the iterative integration of scientific domain knowledge, statistical expertise, and an understanding of data semantics to make nuanced analytical decisions, e.g., about which variables, transformations, and statistical models to consider. LM-based agents equipped with planning, memory, and code execution capabilities have the potential to support data-driven science. However, evaluating agents on such open-ended tasks is challenging due to multiple valid approaches, partially correct steps, and different ways to express the same decisions. To address these challenges, we present BLADE, a benchmark to automatically evaluate agents' multifaceted approaches to open-ended research questions. BLADE consists of 12 datasets and research questions drawn from existing scientific literature, with ground truth collected from independent analyses by expert data scientists and researchers. To automatically evaluate agent responses, we developed corresponding computational methods to match different representations of analyses to this ground truth. Though language models possess considerable world knowledge, our evaluation shows that they are often limited to basic analyses. However, agents capable of interacting with the underlying data demonstrate improved, but still non-optimal, diversity in their analytical decision making. Our work enables the evaluation of agents for data-driven science and provides researchers deeper insights into agents' analysis approaches.

1 Introduction

Scientific data continues to accumulate rapidly, driven by advancements in scientific instrumentation and the digitization of information. However, practicing *data-driven science* (i.e., answering research questions from data) remains difficult, requiring rigorous methodologies, an understanding of data values and semantics, statistical and domain expertise, and critical thinking to validate hypotheses and draw meaningful and justifiable conclusions (Jun et al., 2021; Breznau et al., 2022; Baker, 2016; Collaboration, 2015).

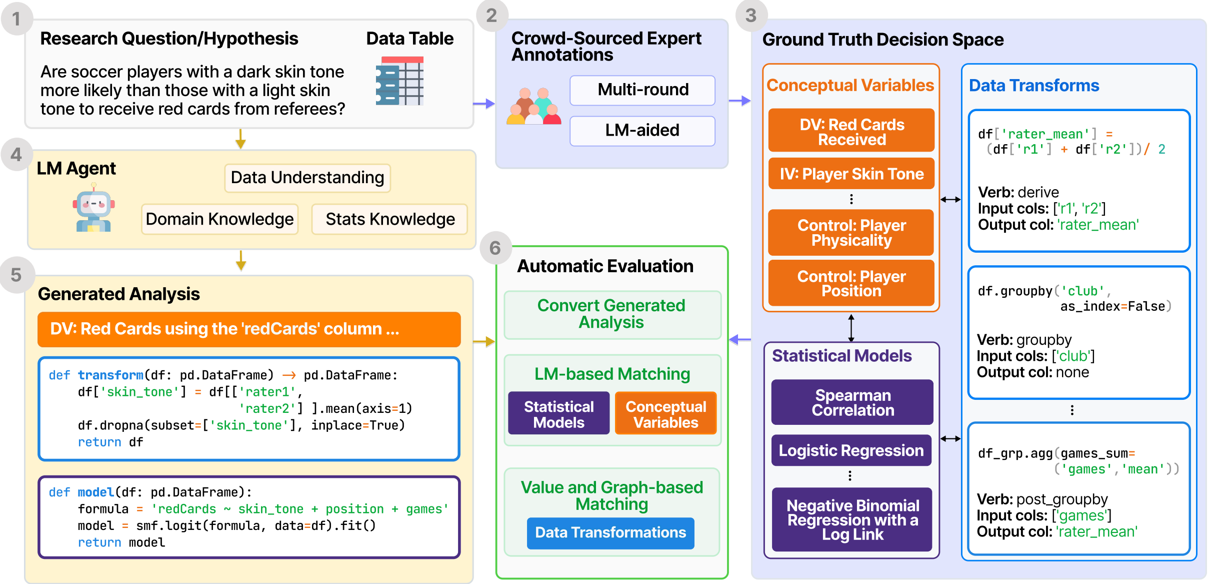
Language model (LM)-based agents (Sumers et al., 2023; Wu et al., 2023; Wang et al., 2023), pre-trained on web-scale data and equipped with memory and tool usage capabilities (Schick et al., 2023), have the potential to conduct and support data-driven science. They can reason about and interact with heterogeneous data representing subjects, objects, and processes of study in the "external" world (Majumder et al., 2024b). However, to facilitate their progress, *we need a reliable method to evaluate and measure their performance*.

Recent benchmarks have enabled progress. However, they focus on either (1) data analysis execution with straightforward tasks containing a single, final, easily evaluated answer (e.g., *Calculate the mean and standard deviation of the "Mar.2019" column* (Hu et al., 2024a; Yin et al., 2022; Liu et al., 2024a)) or (2) tasks for machine learning (ML) (e.g., *improve the accuracy of an ML model* (Hong et al., 2024a; Huang et al., 2023b; Guo et al., 2024b)). For scientific analyses, these tasks require limited integration of external knowledge, limited understanding of data semantics, and limited grounding in external scientific knowledge. In addition, these benchmarks evaluate only on single metrics, such as ML model accuracy or completion rate. However, in the process of data-driven scientific discovery, the many intermediary decisions in a multi-step analysis are themselves critical to identify, meaningfully assess, and differentiate in order to improve agent performance.

Evaluating agent performance on open-ended data-driven analyses, especially automatically, poses specific challenges. First, the *natural flexibility in making analysis decisions* (Gelman and Loken, 2014, 2019; Simmons et al., 2011) makes

[*] These authors contributed equally to this work.

Findings of the Association for Computational Linguistics: EMNLP 2024, pages 13936–13971
November 12-16, 2024 ©2024 Association for Computational Linguistics


1
Research Question/Hypothesis
Data Table
Are soccer players with a dark skin tone more likely than those with a light skin tone to receive red cards from referees?
2
Crowd-Sourced Expert Annotations
Multi-round
LM-aided
3
Ground Truth Decision Space
Conceptual Variables
DV: Red Cards Received
IV: Player Skin Tone
Control: Player Physicality
Control: Player Position
Statistical Models
Spearman Correlation
Logistic Regression
Negative Binomial Regression with a Log Link
Data Transforms
4
LM Agent
Data Understanding
Domain Knowledge
Stats Knowledge
5
Generated Analysis
DV: Red Cards using the 'redCards' column ...
6
Automatic Evaluation
Convert Generated Analysis
LM-based Matching
Statistical Models
Conceptual Variables
Value and Graph-based Matching
Data Transformations

```
def transform(df: pd.DataFrame) → pd.DataFrame:
    df['skin_tone'] = df[['rater1',
                          'rater2'] ].mean(axis=1)
    df.dropna(subset=['skin_tone'], inplace=True)
    return df
```

```
def model(df: pd.DataFrame):
    formula = 'redCards ~ skin_tone + position + games'
    model = smf.logit(formula, data=df).fit()
    return model
```

```
df['rater_mean'] =
 (df['r1'] + df['r2'])/ 2
```

Verb: derive
Input cols: ['r1', 'r2']
Output col: 'rater_mean'

```
df.groupby('club',
           as_index=False)
```

Verb: groupby
Input cols: ['club']
Output col: none

```
df_grp.agg(games_sum=
           ('games','mean'))
```

Verb: post_groupby
Input cols: ['games']
Output col: 'rater_mean'

Figure 1: Overview of BLADE. We gathered research questions and datasets from existing research papers, crowd-sourced analysis studies and statistic textbooks as well as analyses from expert annotators (boxes 1-2-3, and Sec. 3). Given a research question and dataset, LM agents generate a full analysis containing the relevant conceptual variables, a data transform function, and a statistical modeling function (boxes 1-4-5, and Sec. 4.2). BLADE automatically evaluates this against the ground truth (box 6 and Sec. 5).

it hard to establish a single ground truth that encompasses all justifiable choices. Second, the *heterogeneity of decisions* (e.g., regarding hyperparameters of a statistical model, variables choices, high-level approaches, etc.) complicates efforts to decide on the representation and abstraction of meaningful decisions. Finally, given multiple valid decisions and approaches, determining the *criteria and method to assess the correctness and soundness of the agent's analysis* is difficult to quantify.

In this work, we introduce BLADE, a benchmark for the principled evaluation of LM agents used for data-driven scientific analyses. Given a research question (e.g., *"Are soccer players with a dark skin tone more likely than those with a light skin tone to receive red cards from referees?"* (Silberzahn et al., 2018; Auspurg and Brüderl, 2021)) and a dataset, BLADE evaluates agents' ability to integrate external scientific and statistical knowledge with an understanding of the data to conduct rigorously justifiable data analyses.

To build BLADE, we *collected a set of actual research questions and datasets* (Fig. 1.1) from research papers, crowd-sourced analysis studies, and statistics textbooks (Sec. 3). Then, inspired by prior crowd-sourced analysis studies (Silberzahn et al., 2018; Schweinsberg et al., 2021), we *recruited expert data analysts and collected high-quality data analyses* (Fig. 1.2) through a crowd-sourced analysis for each research question (i.e., multiple analysts independently performing a single analysis). To ensure our benchmark captured a broad variety of defensible analysis approaches, we *asked analysts to validate alternative decisions* from their peers and LM-generated decisions seeded by analysts' own decisions. For this process, we also collected negative examples of *"unjustifiable" decisions* to use when testing agents' ability to discern justifiable ones. We then combined all unique decisions to form the *ground truth* (Fig. 1.3).

Next, based on studies outlining decision steps in the data analysis process (Gu et al., 2023a; Liu et al., 2019, 2020a; Jun et al., 2021), we formulated tasks. These tasks tested the discernment and formulation of *analytical decisions that reflect multiple levels of abstraction*, ranging from executable code implementing data transformations to higher-level planning of conceptual variables requiring external scientific knowledge (Sec. 4).

Finally, given our data and task, we developed *representations* and matching criteria for different types of analysis decisions. We also developed corresponding *computational methods to enable automatic evaluation of agent responses* (Sec. 5).

Overall, BLADE contains 188 multiple choice and 536 ground truth analysis decisions encompassing multiple justifiable analysis approaches across 12 real-world datasets and research questions. To

Requirements	**Data Interpreter** (Hong et al., 2024b)	**MLAgentBench** (Huang et al., 2023a)	**QRData** (Liu et al., 2024b)	**DS-Agent** (Guo et al., 2024b)	**DABench** (Hu et al., 2024b)	**Ours**
Agent abilities tested						
(1) comprehend data semantics	–	–	✔	–	–	✔
(2) integrate domain knowledge	–	–	✗	–	–	✔
(3) conduct multi-step reasoning	✔	✔	–	✔	–	✔
(4) discern justifiable decisions	✗	✗	✗	✗	✗	✔
Evaluation characteristics						
(5) automatic	✔	✔	✔	✔	✔	✔
(6) decision-based	✗	✗	✗	✗	✗	✔
(7) flexible to decision input	✗	✗	✗	✗	✗	✔

Table 1: Comparing BLADE against existing data analysis evaluation datasets and benchmarks for conducting *scientific analyses* based on the requirements specified in Section 2. – indicates partial satisfaction (e.g., data understanding is only on ML model building). See Table 4 for examples from BLADE and recent benchmarks.

illustrate its utility and assess benchmark performance, we evaluate different LMs and a standard ReAct agent (Yao et al., 2023) that interacts with a sandbox notebook environment (Sec. 6).

In our results (Sec. 7), we find most LMs are decent at discerning decisions and generating non-empty executable analyses. However, these analysis are basic and lack diversity. In particular, LM's coverage of the ground truth for forming statistical models with conceptual variables is below 13%, and for operationalizing variables, it is below 27% (Fig. 4). The baseline ReAct agent shows a consistent improvement in coverage, though with plenty of room for improvement.

Our main contributions are: (1) a rigorously expert-annotated benchmark and the first of its kind to evaluate agents' analytical decisions on open-ended scientific research questions; (2) an evaluation framework to automatically assess agent responses on fine-grained aspects of the analysis; and (3) results on various LMs and a ReAct agent indicating their current strengths and limitations.

Our work takes the first step in evaluating agents for open-ended data-driven scientific discovery, advancing our understanding of their capabilities to collect data, generate hypotheses, conduct analyses, and interpret results to form valid, justifiable scientific conclusions. To support further research and development, we open source our benchmark and evaluation framework[1].

2 Benchmark Requirements

Our benchmark evaluates agents on answering open-ended, data-driven scientific questions, advancing current efforts that execute analysis code based on precise single-answer instructions. Existing benchmarks are limited in their ability to assess agent decision-making during analysis and often do not capture the full scope of their approaches. Our benchmark addresses these limitations by focusing on the following key requirements (Table 1).

We maintain that the ideal benchmark would evaluate an agent's abilities to (1) *comprehend data semantics*, understanding the semantic relationships between variables and what the data represents relative to the external world, (2) *integrate domain knowledge*, i.e., findings from related literature and an understanding of a "world model", (3) *conduct multi-step reasoning* and planning at different levels of abstraction, i.e., high level planning vs. lower level code execution, given domain knowledge, an understanding of the data, and execution outputs, and (4) *differentiate justifiable decisions* with firm theoretical or statistical support (Simonsohn et al., 2020) from unjustifiable ones.

Additionally, the benchmark evaluation should be (5) *automatic*, requiring no human intervention, (6) *decision-based*, with the ground truth reflecting the intermediary decisions, and (7) *flexible to decision input*, being aware of multiple ways to specify the same decision. Requirements 1 through 4 inform our *data collection process* (Sec. 3) and task formulation (Sec. 4). Requirements 5 through 7 inform our *evaluation procedure* (Sec. 5). Ultimately, we assess whether agents can plan, develop, and execute a justifiable analysis to answer a real-world research question.

3 Benchmark Data Collection

We now describe our data collection process for research questions (RQs), data, and ground-truth analyses.

RQs and Data. We selected scientific-grade datasets and RQs directly from scientific publications, particularly those studied in meta-analysis

[1] https://github.com/behavioral-data/BLADE

papers (Silberzahn et al., 2018; Simonsohn et al., 2020; Young and Holsteen, 2017) and reproduced in statistics textbooks (Mcelreath, 2020; Kleiber and Zeileis, 2008). We chose these sources because they provide a multitude of complex analyses, and relevant properties that make analyses non-trivial and revealing of statistical knowledge. Table 3 summarizes these RQs, datasets, source papers, and meta-analysis papers. During this process, we ensured that the datasets were clearly documented and sufficiently complex to require non-trivial analyses, i.e., expert annotators would be required to distinguish defensible from indefensible decisions.

Annotation Process. To gather ground truth analyses and ensure the highest quality annotations, we followed a procedure similar to those used in previous crowd-sourced analysis studies (Silberzahn et al., 2018; Schweinsberg et al., 2021). We recruited 11 trained analysis experts and engaged them in a multi-stage process to ensure quality. Our experts had a self-reported average of 6 years of experience, with 6 pursuing or holding a Ph.D. in a scientific field. Since one of our key contributions is the corpus of ground truth analyses, we invited our expert annotators to be co-authors of this paper. See Appendix A.1 for details on recruitment.

We gave each expert an RQ, dataset, and dataset description, including details of each column. For each dataset, experts independently conducted their analyses, recording all decisions they made. This naturally resulted in multiple analytical approaches. To broaden the scope of possible strategies, we used an LM (GPT-4) to generate additional decisions (prompts shown in Fig. 9).

To ensure high data quality, experts validated and annotated each other's and LM-generated decisions as justified or unjustified. Agreement rates among expert annotators were relatively high: 75% for transformations and 80% for conceptual variables. In contrast, agreement on LM-generated decisions was much lower, at 27% for transformations and 13% for conceptual variables, highlighting low agent performance on decision generation. Many of these lower-agreement decisions were excluded from the final ground truth dataset.

Finally, we brought the team of experts together to discuss their decisions, resolve ambiguities, and establish consensus. Our ground truth thus reflects alternative approaches validated by multiple experts. See Appendix A.2 for details of our annotation process.

4 Benchmark Tasks

We want the benchmark tasks to represent decisions that are vital to the analysis and to evaluate the key skills needed to conduct data-driven science (i.e., requirements 1-4 in Section 2). We draw from prior studying the scientific analysis process (Gu et al., 2023b; Liu et al., 2019, 2020a) and focus on agents' ability to *discern* (Sec. 4.1) and *make* (Sec. 4.2) *planning* decisions, i.e., those requiring a process of reasoning about and then synthesizing the data, scientific domain, and statistical knowledge. Specifically we test the following decisions:

(1) *Formulating Conceptual Variables.* Agents should recognize independent variables (IVs) , dependent variables (DVs), and control variables based on domain knowledge and multi-step reasoning (requirements 2 and 3), e.g., "Prior literature suggests player physicality influences the referee's perception. We can consider physicality a control."

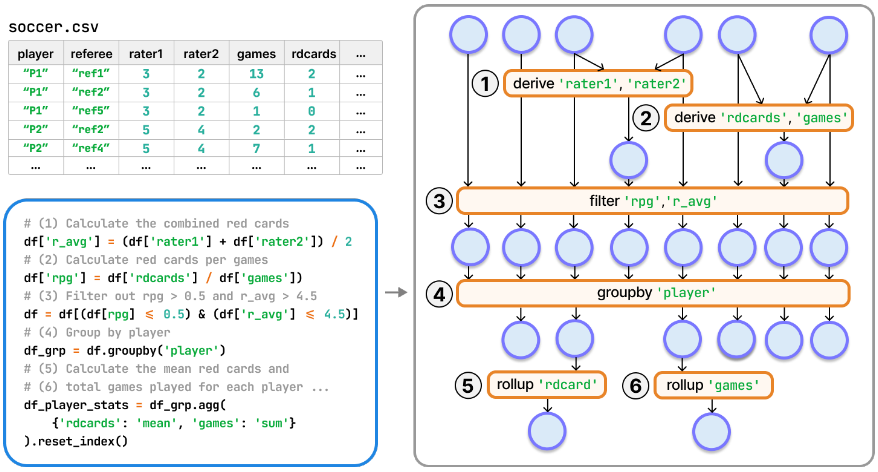
(2) *Executing Data Transformations.* Agents should select relevant columns and apply transformations to operationalize conceptual variables, e.g., using BMI as a proxy for player physicality via *"weight"* and *"height"* columns.

(3) *Implementing Statistical Models.* Agents should choose the appropriate statistical model based on conceptual variables and transformed data to address the research question, requiring in-depth knowledge of statistical methods and the underlying hypothesis (Jun et al., 2019, 2021).

4.1 Task 1: Discern Justifiable Decisions

To evaluate how well agents can discern justifiable decisions (requirement 4), BLADE includes the following **M**ultiple **C**hoice **Q**uestions. (MCQ1) Given the research question and the dataset, which conceptual variable is the *most/least* justifiable for the analysis? (MCQ2) Given the research question, dataset, and conceptual variable of interest, which transformation is the *most/least* justifiable to operationalize the variable?

Each multiple choice question includes one correct and one or more incorrect answers. Justifiable and unjustifiable decisions were gathered during expert reviews of each other's and LM-generated decisions. A decision was deemed justifiable if all experts agreed and unjustifiable if the majority considered it unjustifiable. In addition, for MCQ2, additional negative samples were gathered from transformations that were used to derive conceptual variable that differed from the one in the question (i.e.,



soccer.csv

player	referee	rater1	rater2	games	rdcards	...
“P1”	“ref1”	3	2	13	2	...
“P1”	“ref2”	3	2	6	1	...
“P1”	“ref5”	3	2	1	0	...
“P2”	“ref2”	5	4	2	2	...
“P2”	“ref4”	5	4	7	1	...
...	...	...	...	...	...	...

```
# (1) Calculate the combined red cards
df['r_avg'] = (df['rater1'] + df['rater2']) / 2
# (2) Calculate red cards per games
df['rpg'] = df['rdcards'] / df['games'])
# (3) Filter out rpg > 0.5 and r_avg > 4.5
df = df[(df[rpg] ⩽ 0.5) & (df['r_avg'] ⩽ 4.5)]
# (4) Group by player
df_grp = df.groupby('player')
# (5) Calculate the mean red cards and
# (6) total games played for each player ...
df_player_stats = df_grp.agg(
    {'rdcards': 'mean', 'games': 'sum'}
).reset_index()
```

Figure 2: To allow flexible and fine-grained matching, we represent transforms in code (left) as a column data flow graph G (right). The nodes in blue are column indicator nodes P, and the nodes in orange are transform nodes T. Details of the data flow graph formalization are in Appendix A.3

.

easier negative examples). In total, BLADE contains 188 multiple choice questions.

4.2 Task 2: Generate an End-to-end Analysis

For this significantly more complex task, agents need to generate a complete end-to-end analysis given a research question and a dataset. Specifically, to test agent performance on key analysis decisions, agents are to submit the following artifacts (e.g., in Fig. 1.5 and 7), each mapping to one type of decision.

1. *A list of conceptual variables*, each with a natural language description (e.g, player physicality), the variable type (i.e., an independent, dependent, or control variable), and the name of the column in the final transformed data table used in the statistical model.

2. *An executable transformation function*, which is given a data table as input and returns a data table after performing the transformations to operationalize the conceptual variables.

3. *A statistical model function*, which takes as input the transformed data table and returns the specified statistical model.

5 Flexible Automatic Evaluation

To quantitatively measure the quality of agent-generated analyses (i.e., the agent-generated artifacts) in a way that is *automatic*, *decision-based*, and *flexible to decision input* (requirements 5-7 in Sec. 2), we need concrete representations of analysis decisions and associated matching criteria. We now discuss the representation and matching procedure for each artifact in an agent's submission.

Matching Conceptual Variables. Because conceptual variables capture high-level constructs, two similarly specified constructs (i.e, *player physicality* and *how physically imposing the player is*) should have the same meaning as long as they have the same variable type (i.e., IV, DV, or Control). To match these specifications, we employed an LM (GPT-4o) to determine the semantic equivalence between two conceptual variables. We followed a procedure similar to (Liang et al., 2023) which was validated on semantically matching academic reviews (prompt in Fig. 11). Appendix A.4.2 contains further details.

Matching Data Transformations. Since there are many ways to express data analyses in code, even ways that could be perfectly equivalent, we require a representation that maps equivalent transformations to a single representation (i.e., requirement 7 – *flexible to decision input*). Taking the code in Figure 2 as an example, the ordering of the transforms ①-②-③ or ②-①-③ are functionally equivalent with respect to the final product of the computation (i.e., as long as ① and ② come before ③). In addition, transformations that result in the exact same output column values (with a small margin of error for floating point) should be considered equivalent transformations. Likewise, getting a certain column's values correct should mean that all relevant prior steps were correct and that decisions for each relevant prior transformation were correct. For example, if a submission missed ⑥ but still correctly calculated the "rdcards" after the groupby, then the agent still correctly performed steps ①-⑤, deserving significant partial credit. In complex tasks such as scientific data analysis, such partial credit enables meaningful differentiation of model performance and progress.

To capture the aforementioned nuances, we developed a representation for data transformations using a data flow graph (Kavi et al., 1986) (Fig. 2 right). These graphs are useful because any series of transformations in the order of a topological sort (Manber, 1989) on the graph leads to the same result. In addition, our graph captures data flow at the column-level (i.e., all cell values in a single column) to enable subsequent matching at the granularity of columns. In doing so, we allow for matching on transforms that require and affect only a subset of columns in a data table (e.g., in Fig. 2, getting ① correct is independent of getting ② correct). Appendix A.3 describes our data flow graph formalism in greater detail.

In addition, the transforms (i.e., orange nodes in Fig. 2) in the data flow graph represent a discrete data transformation decision that was made in wrangling the data (requirement 6 – *decision-based* evaluation). Specifically, each transform is defined by a fixed set of transform verbs (Table 6) that are based on existing data wrangling libraries (i.e., Arquero[2] and Vega (Satyanarayan et al., 2016)), expandable, and validated to cover every analysis decision in our benchmark. To match transforms in BLADE, we applied an LM (GPT-4o) to convert the transformation function in an agent's submission to the individual transform units (prompt in Fig. 12 and 13). We then constructed the agent's transformation data flow graph and matched it with the ground truth. We match based on both the *column values* that are the output of any discrete transformation and a *fuzzier graph isomorphism matching* that determines whether approximately the same steps were applied. Appendix A.4.1 describes the matching procedures in detail.

Matching Statistical Models. The implementation of statistical models and relevant parameters could be evaluated in multiple ways (i.e., code, natural language, or mathematical formulas (Jun et al., 2021; McElreath, 2018)). To prioritize the underexplored *planning* aspects of statistical modeling (Gu et al., 2023b), we focus on being able to select the right model and conceptual variables. In principle, this representation could be extended to include code, hyperparameters, and more. We first used an LM (GPT-4o) prompt (Fig. 14) to convert the modeling function into a natural language specification of the model and the columns in the transformed data table that it used. Next, using another LM (GPT-4o) prompt (Fig. 15), we compared this output with the ground truth natural language specifications of the model and associated conceptual variables based on semantic equivalence. See Appendix A.4.3 for additional details.

Evaluation of LM Evaluation Modules. To validate our LM-based evaluation modules, two authors independently reviewed a sample of 615 LM-generated outputs across multiple datasets. After an initial round of review and resolution of any disagreements, the modules achieved the following correctness rates: 93% for matching conceptual variables, 97% for translating transform code into transform units, 97% for converting modeling code into a natural language specification, and 92% for matching statistical models. These results were deemed sufficient for our evaluation purposes.

6 Experiments

To establish a baseline and evaluate the performance of LM-based agents on BLADE, we selected the following models: GPT-3.5 Turbo, GPT-4o (OpenAI, 2024), Gemini 1.5 Pro (Pichai, 2024), and Claude 3.5 Sonnet (Anthropic, 2024) to represent closed-source general-purpose LMs; Llama3 8B, Llama3 70B (Meta, 2024), and Mixtral-8x22B (Mistral, 2024) for open-source LMs; and CodeLlama Instruct 7B (Rozière et al., 2023) and DeepSeek-Coder Instruct 6.7B (Guo et al., 2024a) for coding-specific LMs.

Experiment Settings. For the multiple choice questions (Sec. 4.1, Task 1) we evaluate each LM with a temperature of 0. To generate an end-to-end analysis (Sec. 4.2, Task 2), we evaluate LMs in one turn with a one-shot example (prompt in Fig. 16). In addition, we develop an agent (also with an example demonstration), based on the ReAct framework (Yao et al., 2023), that interacts with a computational notebook environment containing the data, reflects on observations from executing the code, and generates next-step actions. We evaluate the ReAct agent on Mixtral-8x22b, GPT-3.5 Turbo, GPT-4o, Gemini 1.5 Pro, and Claude 3.5 Sonnet. Appendix A.5 contains additional details

[2] `https://idl.uw.edu/arquero/`

on the setup of the agent and the choice of LMs.

For each LM and setting (i.e., one turn vs. ReAct agent), to encourage diversity we set the temperature to 0.8 and record a total of 40 runs for the one-turn setting and 20 runs for the agent setting to consider for computational budget. For all LMs used to facilitate the evaluation (i.e., conversion and semantic matching), we use GPT-4o with a temperature of 0. Appendix A.6 includes all prompts for the baselines and LM-aided evaluation.

Evaluation Metrics. For the multiple choice tasks (Task 1), we measure agents on *accuracy*. For the generation tasks (Task 2), to measure an agent's ability to both generate justifiable analyses and capture the breadth of justifiable approaches, we calculate an adapted *F1-score* for each type of analysis decision (conceptual variables, transformation, and statistical model). The F1-score takes the harmonic mean of *average precision* across runs and *coverage@k*. The former quantifies how well an agent's response matched with the ground truth while the latter evaluates how comprehensive agents are in generating justifiable alternative analyses. In our experiments, we report average precision across *all* runs and coverage for $k = 10$ runs. Appendix A.7 contains the full details of our evaluation metrics.

7 Results

We report the performance of LMs on MCQs (Task 1) in Figure 3 and the results of LMs and our ReAct agent for analysis generation (Task 2) in Table 2 and Figure 4. Here, we summarize our main findings.

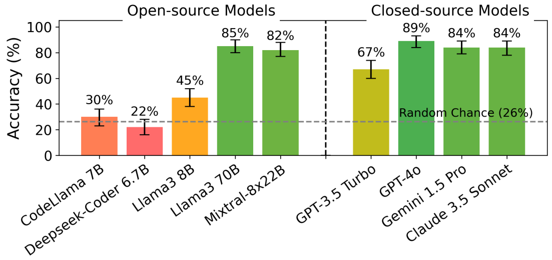


Figure 3: Accuracy scores and 95% confidence intervals for different models on BLADE's 188 MCQs (168 for transformations and 20 for conceptual variables).

Models	F1	(95% CI)
One-turn Setting		
CodeLlama 7B	16.8	(15.2, 18.5)
Deepseek-Coder 6.7B	33.9	(32.2, 35.4)
Llama3 8B	29.6	(27.7, 31.5)
Llama3 70B	36.3	(34.7, 37.8)
Mixtral-8x22B	40.1	(38.0, 42.1)
GPT-3.5 Turbo	30.5	(28.7, 32.2)
GPT-4o	41.7	(40.2, 43.2)
Gemini 1.5 Pro	41.1	(39.6, 42.5)
Claude 3.5 Sonnet	<u>43.9</u>	(42.6, 44.9)
Agent Setting		
Mixtral-8x22B	40.8	(38.2, 42.9)
GPT-3.5 Turbo	37.2	(34.7, 39.7)
GPT-4o	**<u>44.8</u>**	(43.0, 46.3)
Gemini 1.5 Pro	40.1	(38.3, 41.5)
Claude 3.5 Sonnet	43.1	(41.4, 44.8)

Table 2: We report the decision-type weighted F1-score on analysis generation based on average precision and coverage@10. Appendix A.7 has the calculation details.

LMs have acceptable world knowledge. We find that LMs can identify some relevant conceptual variables based on the research question and dataset (i.e., a reasonable precision and coverage for conceptual variables). In BLADE, many relevant conceptual variables are possibly hinted at in the research question and available data columns. Although our setting is realistic and common, future work could explore how LM agents perform in generating hypotheses and identifying relevant data without such context (Majumder et al., 2024b). In addition, we find that the best general LMs (i.e., Gemini-1.5 Pro, Mixtral-8x22b, Claude-3.5-Sonnet and GPT-4o) perform well on the MCQs (Fig. 3). They can discern the obvious transformations for a given conceptual variable. In contrast, code-specific LMs, like CodeLlama and DeepSeek-Coder, struggle to identify the correct decision.

Most LMs can generate non-empty executable analyses. For generating an analysis, we find that most large LMs can generate a non-empty executable analysis over 60% of the time, with GPT-4o being the best at 96% (Fig. 5). Among the open-source models, Mixtral-8x22b performs best, generating an executable analysis 73% of the time and DeepSeek-Coder also does surprisingly well at 65%. In a manual inspection of non-executable analyses, we notice issues with respect to hallucinating data attributes. Taking one of DeepSeek-Coder's submissions to the soccer dataset as an example, we observe plausible looking code, but it hallucinates the "RefCountry" column, which does not actually appear in the data table (Figure 21-7).

LMs struggle to specify statistical models and concretely operationalize conceptual variables. LMs perform relatively poorly in forming statistical models with the right conceptual variables (precision below 35%) and operationalizing the variables

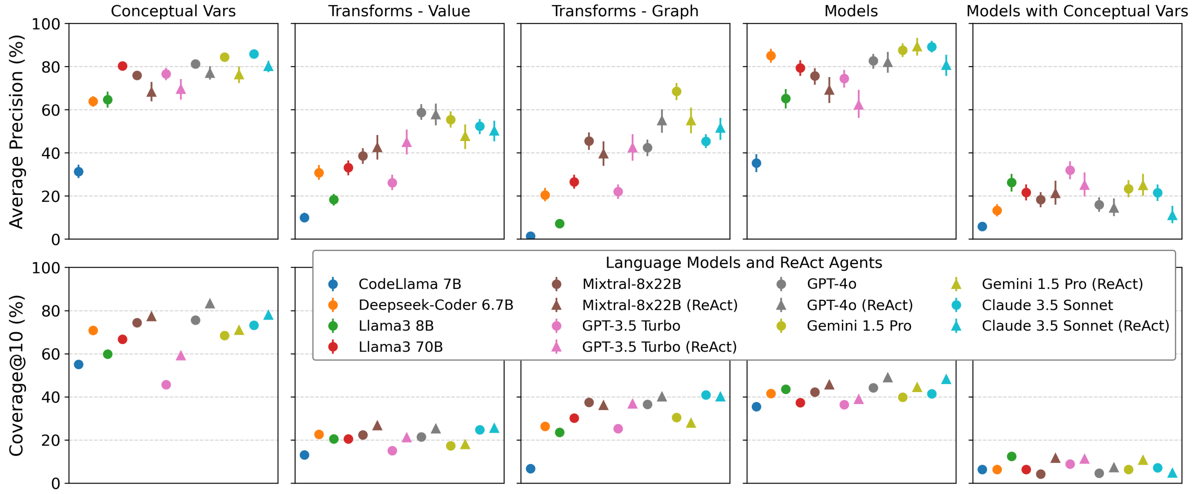


Figure 4: Average precision (top row) and coverage@10 (bottom row) percentages averaged across datasets in BLADE. All runs were included in the results. Run errors default to a hit rate of 0 and are counted in the coverage calculation (i.e., treated as a run that generated nothing). Error bars represent bootstrapped 95% confidence intervals.

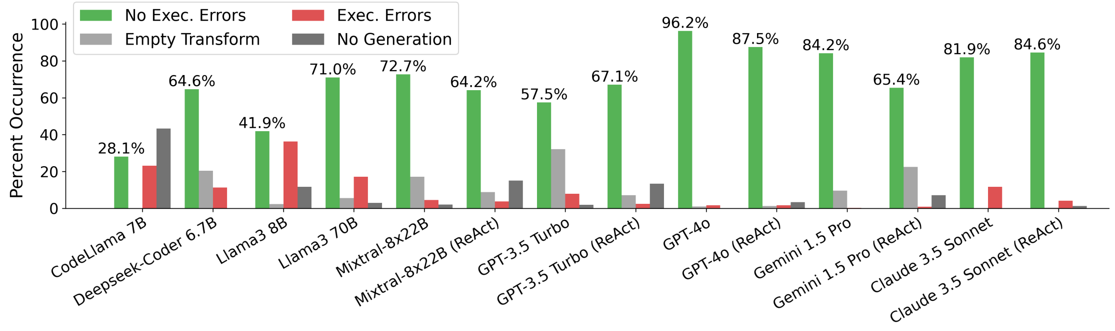


Figure 5: Characterization of run results for analysis generation for each LM and ReAct agent variants. "No execution errors" indicates executable transform code, "Empty transform" means no transformations were provided, "Execution errors" means the code resulted in errors, and "No generation" indicates the result could not be parsed.

(precision below 60%). In addition, LMs perform even worse in terms of coverage for forming statistical models with conceptual variables (coverage@10 below 13% across) and operationalizing the variables (coverage@10 below 27%). This indicates there is room for improvement not only in generating valid analyses, but also generating more complex and diverse analyses that might require additional reasoning beyond the basic steps.

LMs are limited to forming basic analyses. Figure 5 also shows that many LM's submissions contain empty transform code, especially for GPT-3.5 Turbo and Gemini 1.5 Pro. We also observe low coverage of the ground truth examples (Fig. 4 bottom), especially with respect to data transformations and specific model specifications. Through qualitatively reviewing a random sample of LM-generated analyses, we find that LMs often perform basic analysis that can yield decent precision (i.e., matching basic decisions) but poor coverage across runs. See Appendix A.8 for examples.

Agents can improve the diversity of analyses. Comparing the one-turn and agent settings, LMs consistently had higher coverage when allowed to iteratively explore data. Moreover, ReAct agents perform best overall on coverage for data transformations and statistical modeling, which require a more detailed understanding of data semantics (Fig. 4 bottom). Future work can explore how augmenting agents with external knowledge (e.g., from academic papers) can further improve performance.

Stronger performance on code generation does not translate directly to BLADE. When comparing our results in analysis generation with those from the HumanEval coding benchmark (Fig. 6), we found that most metrics showed a positive correlation, indicating that higher HumanEval performance is broadly correlated with higher BLADE performance. However, coverage measures (Fig. 6

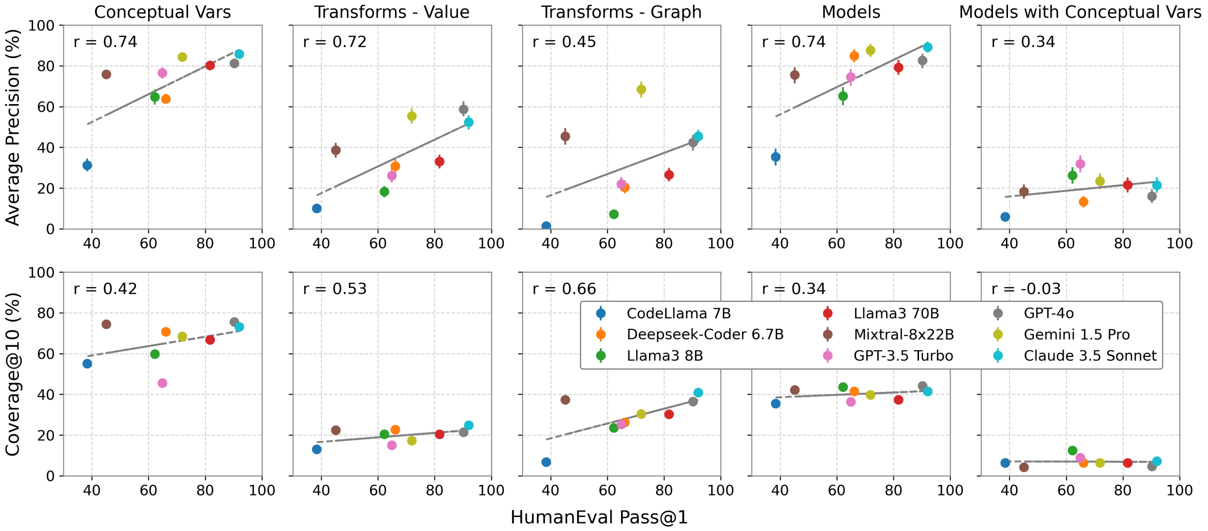


Figure 6: BLADE Performance vs. HumanEval Performance. We compare BLADE evaluation metrics against reported Pass@1 on HumanEval (Chen et al., 2021) for all LMs in our experiments.

bottom) had a weaker correlation compared to precision (Fig. 6 top). This suggests that while current training methods, such as Reinforcement Learning from Human Feedback (RLHF) and instruction tuning, optimize for one solution, they may struggle to generate diverse solutions, a phenomenon observed in other contexts (Li et al., 2024c).

We also highlight that Gemini 1.5 Pro consistently performed better on precision than its HumanEval performance would suggest, while Mixtral-8x22B excelled in both precision and coverage for conceptual variables and data transformations. In contrast, CodeLlama consistently performed worse on BLADE than HumanEval. Given that Gemini 1.5 Pro and Mixtral-8x22B are general-purpose Mixture-of-Experts models, our findings highlight BLADE as a challenging benchmark that assesses more than just code generation. Our results identify specific areas for improvement, such as enhancing the complexity and diversity of analyses or generating justifiable statistical models.

8 Related Work

Our work broadly relates to agent benchmarks for data science and LM agents in science.

Benchmarks for Data Science. Many benchmarks (Li et al., 2024b; Yin et al., 2022; Hu et al., 2024a) assess agents' data science code execution but are limited in measuring complex reasoning and external knowledge integration needed for scientific analyses. Other works focus on improving specific metrics in machine learning tasks (Huang et al., 2023b; Hong et al., 2024a) without evaluating intermediate decisions (examples in Table 4). Some works assess agents' causal (Jin et al., 2023) and quantitative reasoning (Liu et al., 2024a), but often lack data or involve closed-ended solutions, missing the flexibility inherent in open-ended scientific analyses. DiscoveryBench (Majumder et al., 2024a) evaluates agents on generating data-driven hypotheses but does not explicitly measure decisions in the analyses. In contrast, BLADE focuses on evaluating *multiple* valid analysis *approaches* to *open-ended*, *data-driven* research questions.

Agents for Science. Advancements in LMs have ignited research interest in applying agents to automate scientific discovery (Liang et al., 2023; Romera-Paredes et al., 2023; Shojaee et al., 2024; Kramer et al., 2023; Bran et al., 2023; Boiko et al., 2023; Majumder et al., 2024b). Our work seeks to provide a thorough automated evaluation of agents for scientific analyses across domains.

9 Conclusion

We introduce BLADE, a benchmark designed to advance the development of LM agents for data-driven scientific tasks. We collected a dataset of research questions and data tables, along with ground truth analyses from expert annotators. To support an *automatic*, *decision-based*, and *input-flexible* evaluation, we devised representations of core analysis decisions and developed corresponding matching algorithms. Although current generations of LMs can generate *some* analyses matching the ground truth *sometimes*, we find that these analyses are limited in complexity and lack diversity.

10 Limitations

Our work is not without limitations. First, BLADE does not evaluate an agent's ability to interpret the results of data analyses as part of the end-to-end data analysis process. Understanding and interpreting model results is vital but can be difficult to capture cleanly since it may require analysts' subjective interpretation of the problem with respect to model results. We leave this important dimension for future work.

In addition, though our work elucidates the decisions an agent may make, we do not explicitly evaluate the exploratory parts of an analysis. Further, we assume that the dataset is contained in a single, potentially extremely large table. This may not be common of all research datasets, but we believe this factor does not significantly reduce the scope of BLADE since joining tables to enable downstream analyses is a task that LMs already commonly perform (Liu et al., 2023; Li et al., 2024a; Pourreza and Rafiei, 2023).

Finally, some components of our evaluation rely on LMs (e.g., conversion of code to discrete transforms, semantically matching model, and conceptual variable), which are known to hallucinate. Therefore, we made multiple efforts to validate each component and do not think that hallucination significantly impacts our ability to effectively and automatically evaluate agents. We also open source these evaluation modules so that researchers can build upon them to improve our evaluation.

Acknowledgments

We are grateful for the expert annotators who worked with us over multiple rounds to gather the highest quality data. We also thank the UW Behavioral Data Science Group members for their suggestions and feedback. Additionally, we thank Tiffany Zheng for her support and brainstorming on the figures, Josh Gardner and Andrew McNutt for their early feedback on the overall project, and Farhan Samir for his input towards evaluation. This research was supported in part by NSF grant IIS-1901386, NSF CAREER IIS-2142794, Bill & Melinda Gates Foundation (INV-004841), and the Office of Naval Research (#N00014-21-1-2154).

References

2021. Arquero: Query processing and transformation of array-backed data tables. `https://github.com/uwdata/arquero`.

Anthropic. 2024. Claude 3.5 sonnet. Accessed: July 13, 2024.

Katrin Auspurg and Josef Brüderl. 2021. Has the credibility of the social sciences been credibly destroyed? reanalyzing the "many analysts, one data set" project. *Socius*, 7:23780231211024421.

Monya Baker. 2016. 1,500 scientists lift the lid on reproducibility. *Nature*, 533:452–454.

Laura A Bakkensen and William Larson. 2014. Population matters when modeling hurricane fatalities. *Proceedings of the National Academy of Sciences*, 111(50):E5331–E5332.

Christophe Boesch, Daša Bombjaková, Amelia Meier, and Roger Mundry. 2019. Learning curves and teaching when acquiring nut-cracking in humans and chimpanzees. *Scientific Reports*, 9(1):1515.

Daniil A. Boiko, Robert MacKnight, Ben Kline, and Gabe Gomes. 2023. Autonomous chemical research with large language models. *Nature*, 624:570 – 578.

Andrés M Bran, Sam Cox, Oliver Schilter, Carlo Baldassari, Andrew D. White, and Philippe Schwaller. 2023. Augmenting large language models with chemistry tools. *Nature Machine Intelligence*, 6:525 – 535.

Nate Breznau, Eike Mark Rinke, Alexander Wuttke, Muna Adem, Jule Adriaans, Amalia Álvarez-Benjumea, Henrik Kenneth Andersen, Daniel Auer, Flávio Azevedo, Oke Bahnsen, David Balzer, Gerrit Bauer, Paul Cornelius Bauer, Markus Baumann, Sharon Baute, et al. 2022. Observing many researchers using the same data and hypothesis reveals a hidden universe of uncertainty. *Proceedings of the National Academy of Sciences of the United States of America*, 119.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde, Jared Kaplan, Harrison Edwards, Yura Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, et al. 2021. Evaluating large language models trained on code. *ArXiv*, abs/2107.03374.

Open Science Collaboration. 2015. Estimating the reproducibility of psychological science. *Science*, 349(6251):aac4716.

Margaret C Crofoot, Ian C Gilby, Martin C Wikelski, and Roland W Kays. 2008. Interaction location outweighs the competitive advantage of numerical superiority in cebus capucinus intergroup contests. *Proceedings of the National Academy of Sciences*, 105(2):577–581.

Victor Dibia. 2023. Lida: A tool for automatic generation of grammar-agnostic visualizations and infographics using large language models. *arXiv preprint arXiv:2303.02927*.

Ray C Fair. 1978. A theory of extramarital affairs. *Journal of political economy*, 86(1):45–61.

Andrew Gelman and E. Loken. 2014. The statistical crisis in science. *American Scientist*, 102:460.

Andrew Gelman and E. Loken. 2019. The garden of forking paths : Why multiple comparisons can be a problem , even when there is no " fishing expedition " or " p-hacking " and the research hypothesis was posited ahead of time.

Cassandra C Gilmore. 2013. A comparison of antemortem tooth loss in human hunter-gatherers and non-human catarrhines: Implications for the identification of behavioral evolution in the human fossil record. *American journal of physical anthropology*, 151(2):252–264.

Ken Gu, Madeleine Grunde-McLaughlin, Andrew M. McNutt, Jeffrey Heer, and Tim Althoff. 2023a. How do data analysts respond to ai assistance? a wizard-of-oz study. In *International Conference on Human Factors in Computing Systems*.

Ken Gu, Ruoxi Shang, Tim Althoff, Chenglong Wang, and Steven Mark Drucker. 2023b. How do analysts understand and verify ai-assisted data analyses? In *International Conference on Human Factors in Computing Systems*.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Yu Wu, Y. K. Li, Fuli Luo, Yingfei Xiong, and Wenfeng Liang. 2024a. Deepseek-coder: When the large language model meets programming - the rise of code intelligence. *ArXiv*, abs/2401.14196.

Siyuan Guo, Cheng Deng, Ying Wen, Hechang Chen, Yi Chang, and Jun Wang. 2024b. Ds-agent: Automated data science by empowering large language models with case-based reasoning. *ArXiv*, abs/2402.17453.

Daniel S Hamermesh and Amy Parker. 2005. Beauty in the classroom: Instructors' pulchritude and putative pedagogical productivity. *Economics of Education Review*, 24(4):369–376.

Sirui Hong, Yizhang Lin, Bangbang Liu, Binhao Wu, Danyang Li, Jiaqi Chen, Jiayi Zhang, Jinlin Wang, Lingyao Zhang, Mingchen Zhuge, Taicheng Guo, Tuo Zhou, Wei Tao, Wenyi Wang, Xiangru Tang, et al. 2024a. Data interpreter: An llm agent for data science. *ArXiv*, abs/2402.18679.

Sirui Hong, Yizhang Lin, Bangbang Liu, Binhao Wu, Danyang Li, Jiaqi Chen, Jiayi Zhang, Jinlin Wang, Lingyao Zhang, Mingchen Zhuge, et al. 2024b. Data interpreter: An llm agent for data science. *arXiv preprint arXiv:2402.18679*.

Xueyu Hu, Ziyu Zhao, Shuang Wei, Ziwei Chai, Guoyin Wang, Xuwu Wang, Jing Su, Jingjing Xu, Ming Zhu, Yao Cheng, Jianbo Yuan, Kun Kuang, Yang Yang, Hongxia Yang, and Fei Wu. 2024a. Infiagent-dabench: Evaluating agents on data analysis tasks. *ArXiv*, abs/2401.05507.

Xueyu Hu, Ziyu Zhao, Shuang Wei, Ziwei Chai, Guoyin Wang, Xuwu Wang, Jing Su, Jingjing Xu, Ming Zhu, Yao Cheng, et al. 2024b. Infiagent-dabench: Evaluating agents on data analysis tasks. *arXiv preprint arXiv:2401.05507*.

Qian Huang, Jian Vora, Percy Liang, and Jure Leskovec. 2023a. Benchmarking large language models as ai research agents. *arXiv preprint arXiv:2310.03302*.

Qian Huang, Jian Vora, Percy Liang, and Jure Leskovec. 2023b. Mlagentbench: Evaluating language agents on machine learning experimentation.

Zhijing Jin, Yuen Chen, Felix Leeb, Luigi Gresele, Ojasv Kamal, Zhiheng Lyu, Kevin Blin, Fernando Gonzalez Adauto, Max Kleiman-Weiner, Mrinmaya Sachan, and Bernhard Scholkopf. 2023. Cladder: Assessing causal reasoning in language models.

Eunice Jun, M. Keith Birchfield, Nicole de Moura, Jeffrey Heer, and René Just. 2021. Hypothesis formalization: Empirical findings, software limitations, and design implications. *ACM Trans. Comput. Hum. Interact.*, 29:6:1–6:28.

Eunice Jun, Maureen Daum, Jared Roesch, Sarah E. Chasins, E. Berger, René Just, and Katharina Reinecke. 2019. Tea: A high-level language and runtime system for automating statistical analysis. *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology*.

Kiju Jung, Sharon Shavitt, Madhu Viswanathan, and Joseph M Hilbe. 2014. Female hurricanes are deadlier than male hurricanes. *Proceedings of the National Academy of Sciences*, 111(24):8782–8787.

Krishna M. Kavi, Bill P. Buckles, Senior Member, and U. Narayan Bhat. 1986. A formal definition of data flow graph models. *IEEE Transactions on Computers*, C-35:940–948.

Christian Kleiber and Achim Zeileis. 2008. *Applied econometrics with R*. Springer Science & Business Media.

Lyle W Konigsberg and Susan R Frankenberg. 2013. Bayes in biological anthropology. *American Journal of Physical Anthropology*, 152:153–184.

Stefan Kramer, Mattia Cerrato, Sašo Džeroski, and Ross D. King. 2023. Automated scientific discovery: From equation discovery to autonomous discovery systems. *ArXiv*, abs/2305.02251.

Haoyang Li, Jing Zhang, Hanbing Liu, Ju Fan, Xiaokang Zhang, Jun Zhu, Renjie Wei, Hongyan Pan, Cuiping Li, and Hong Chen. 2024a. Codes: Towards building open-source language models for text-to-sql. *ArXiv*, abs/2402.16347.

Jinyang Li, Nan Huo, Yan Gao, Jiayi Shi, Yingxiu Zhao, Ge Qu, Yurong Wu, Chenhao Ma, Jian-Guang Lou, and Reynold Cheng. 2024b. Tapilot-crossing: Benchmarking and evolving llms towards interactive data analysis agents. *ArXiv*, abs/2403.05307.

Margaret Li, Weijia Shi, Artidoro Pagnoni, Peter West, and Ari Holtzman. 2024c. Predicting vs. acting: A trade-off between world modeling & agent modeling. *ArXiv*, abs/2407.02446.

Qisheng Li, Meredith Ringel Morris, Adam Fourney, Kevin Larson, and Katharina Reinecke. 2019. The impact of web browser reader views on reading speed and user experience. In *Proceedings of the 2019 CHI conference on human factors in computing systems*, pages 1–12.

Weixin Liang, Yuhui Zhang, Hancheng Cao, Binglu Wang, Daisy Ding, Xinyu Yang, Kailas Vodrahalli, Siyu He, Daniel Smith, Yian Yin, Daniel A. McFarland, and James Zou. 2023. Can large language models provide useful feedback on research papers? a large-scale empirical analysis. *ArXiv*, abs/2310.01783.

Aiwei Liu, Xuming Hu, Lijie Wen, and Philip S. Yu. 2023. A comprehensive evaluation of chatgpt's zero-shot text-to-sql capability. *ArXiv*, abs/2303.13547.

Jiali Liu, Nadia Boukhelifa, and James R. Eagan. 2020a. Understanding the role of alternatives in data analysis practices. *IEEE Transactions on Visualization and Computer Graphics*, 26:66–76.

Xiao Liu, Zirui Wu, Xueqing Wu, Pan Lu, Kai-Wei Chang, and Yansong Feng. 2024a. Are llms capable of data-based statistical and causal reasoning? benchmarking advanced quantitative reasoning with data. *ArXiv*, abs/2402.17644.

Xiao Liu, Zirui Wu, Xueqing Wu, Pan Lu, Kai-Wei Chang, and Yansong Feng. 2024b. Are llms capable of data-based statistical and causal reasoning? benchmarking advanced quantitative reasoning with data. *arXiv preprint arXiv:2402.17644*.

Yang Liu, Tim Althoff, and Jeffrey Heer. 2019. Paths explored, paths omitted, paths obscured: Decision points & selective reporting in end-to-end data analysis. *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*.

Yang Liu, Alex Kale, Tim Althoff, and Jeffrey Heer. 2020b. Boba: Authoring and visualizing multiverse analyses. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):1753–1763.

J Scott Long and Jeremy Freese. 2006. *Regression models for categorical dependent variables using Stata*, volume 7. Stata press.

Bodhisattwa Prasad Majumder, Harshit Surana, Dhruv Agarwal, Bhavana Dalvi, Abhijeetsingh Meena, Aryan Prakhar, Tirth Vora, Tushar Khot, Ashish Sabharwal, and Peter Clark. 2024a. Discoverybench: Towards data-driven discovery with large language models.

Bodhisattwa Prasad Majumder, Harshit Surana, Dhruv Agarwal, Sanchaita Hazra, Ashish Sabharwal, and Peter Clark. 2024b. Data-driven discovery with large generative models. *ArXiv*, abs/2402.13610.

Steve Maley. 2014. Statistics show no evidence of gender bias in the public's hurricane preparedness. *Proceedings of the National Academy of Sciences*, 111(37):E3834–E3834.

Daniel Malter. 2014. Female hurricanes are not deadlier than male hurricanes. *Proceedings of the National Academy of Sciences*, 111(34):E3496–E3496.

Udi Manber. 1989. *Introduction to Algorithms: A Creative Approach*. Addison-Wesley Longman Publishing Co., Inc., USA.

Richard McElreath. 2018. *Statistical rethinking: A Bayesian course with examples in R and Stan*. Chapman and Hall/CRC.

Richard Mcelreath. 2020. Statistical rethinking.

Meta. 2024. Introducing meta llama 3: The most capable openly available llm to date. Accessed: July 13, 2024.

Mistral. 2024. Cheaper, better, faster, stronger. Accessed: July 13, 2024.

Alicia H Munnell, Geoffrey MB Tootell, Lynn E Browne, and James McEneaney. 1996. Mortgage lending in boston: Interpreting hmda data. *The American Economic Review*, pages 25–53.

OpenAI. 2024. Hello, GPT-4.0! `https://openai.com/index/hello-gpt-4o/`. Accessed: June 13, 2024.

Sundar Pichai. 2024. Next-generation ai model: Google gemini. Accessed: June 13, 2024.

Mohammad Reza Pourreza and Davood Rafiei. 2023. Din-sql: Decomposed in-context learning of text-to-sql with self-correction. *ArXiv*, abs/2304.11015.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, Alhussein Fawzi, Josh Grochow, Andrea Lodi, Jean-Baptiste Mouret, et al. 2023. Mathematical discoveries from program search with large language models. *Nature*, 625:468 – 475.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, I. Evtimov, Joanna Bitton, Manish P Bhatt, Cristian Cantón Ferrer, et al. 2023. Code llama: Open foundation models for code. *ArXiv*, abs/2308.12950.

Arvind Satyanarayan, Dominik Moritz, Kanit Wongsuphasawat, and Jeffrey Heer. 2016. Vega-lite: A grammar of interactive graphics. *IEEE transactions on visualization and computer graphics*, 23(1):341–350.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *ArXiv*, abs/2302.04761.

Martin Schweinsberg, Michael Feldman, Nicola Staub, Olmo van den Akker, Robbie C. M. Aert, Marcel A.L.M. van Assen, Yang Liu, Tim Althoff, Jeffrey Heer, Alex Kale, Zainab Mohamed, Hashem Amireh, Vaishali Venkatesh Prasad, Abraham Bernstein, Emily Spears Robinson, et al. 2021. Same data, different conclusions: Radical dispersion in empirical results when independent analysts operationalize and test the same hypothesis. *Organizational Behavior and Human Decision Processes*.

Parshin Shojaee, Kazem Meidani, Shashank Gupta, Amir Barati Farimani, and Chandan K. Reddy. 2024. Llm-sr: Scientific equation discovery via programming with large language models. *ArXiv*, abs/2404.18400.

Raphael Silberzahn, Eric Luis Uhlmann, Daniel P. Martin, Pasquale Anselmi, Frederik Aust, Eli Awtrey, Štěpán Bahník, Fangxing Bai, Colin Bannard, Evelina Bonnier, Rickard Carlsson, Felix Cheung, G. Christensen, Russ Clay, Maureen A. Craig, et al. 2018. Many analysts, one data set: Making transparent how variations in analytic choices affect results. *Advances in Methods and Practices in Psychological Science*, 1:337 – 356.

Joseph P. Simmons, Leif D. Nelson, and Uri Simonsohn. 2011. False-positive psychology. *Psychological Science*, 22:1359 – 1366.

Uri Simonsohn, Joseph P Simmons, and Leif D Nelson. 2020. Specification curve analysis. *Nature Human Behaviour*, 4(11):1208–1214.

James H Stock and Mark W Watson. 2020. *Introduction to econometrics*. Pearson.

Theodore R. Sumers, Shunyu Yao, Karthik Narasimhan, and Thomas L. Griffiths. 2023. Cognitive architectures for language agents. *ArXiv*, abs/2309.02427.

Edwin JC Van Leeuwen, Emma Cohen, Emma Collier-Baker, Christian J Rapold, Marie Schäfer, Sebastian Schütte, and Daniel BM Haun. 2018. The development of human social learning across seven societies. *Nature Communications*, 9(1):2076.

Lei Wang, Chengbang Ma, Xueyang Feng, Zeyu Zhang, Hao ran Yang, Jingsen Zhang, Zhi-Yang Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Ji rong Wen. 2023. A survey on large language model based autonomous agents. *ArXiv*, abs/2308.11432.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Shaokun Zhang, Erkang Zhu, Beibin Li, Li Jiang, Xiaoyun Zhang, and Chi Wang. 2023. Autogen: Enabling next-gen llm applications via multi-agent conversation framework. *ArXiv*, abs/2308.08155.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*.

Pengcheng Yin, Wen-Ding Li, Kefan Xiao, A. Eashaan Rao, Yeming Wen, Kensen Shi, Joshua Howland, Paige Bailey, Michele Catasta, Henryk Michalewski, Oleksandr Polozov, and Charles Sutton. 2022. Natural language to code generation in interactive data science notebooks. *ArXiv*, abs/2212.09248.

Cristobal Young and Katherine Holsteen. 2017. Model uncertainty and robustness: A computational framework for multimodel analysis. *Sociological Methods & Research*, 46(1):3–40.

A Appendix

A.1 Data Collection Recruitment

Data analysts were recruited through open calls on social media platforms and personal connections. Of the analysts interested, a subset was selected based on their CVs reflecting education, training, and practices in statistical foundations and data analysis. The selected analysts provided sufficiently detailed analysis reports in a screening task and proceeded to the formal annotation phase. A total of 11 analysts participated in the final annotation (see Table 5 for annotator information).

The participating analysts self-reported an average of 6 years of experience in data analysis (range: 4-8 years), with 4 analysts performing data analysis on a daily basis and 7 engaging in it a few times a week. The team included 6 people holding or pursuing Ph.D. degree in Statistics or a related field (Ph. D. in Biostatistics, Ph.D. in Biomedical and Health Informatics, Ph.D. in Measurement, Evaluation, & Research Methodology), the rest held at least 1 Master's degree in a related field. The analysts' occupations varied, 7 were graduate students, 3 held data scientist positions in the finance and technology industries, and 1 was a quantitative researcher in finance.

By assembling a team of analysts with diverse backgrounds and a broad range of expertise in statistical analysis methods, we ensure that the ground truth dataset is constructed using a comprehensive set of methods. At least half (n=5) of the analysts self-reported being familiar "to a high extent" or "to a very high extent" with common classes of analysis methods including descriptive statistics, inferential statistics, hypothesis testing, estimation, correlation, and regression.

A.2 Data Collection Procedure

While the analysts were free to conduct the analysis in their preferred computational environment, we took several additional steps to ensure the quality of our ground truth.

To ensure consistency of annotations, we built a pipeline with structured training and annotation procedure aimed at ensuring well-prepared analysts, consistent and reliable analysis decision specifications, and a diverse range of justifiable models and analysis approaches. These decisions cover conceptual variable formulation, executing data transformations to operationalize the variables, and implementing statistical models (Sec. 4).

To streamline the annotation process and reduce some of the cognitive load in specification, we developed a customized annotation interface that supports structured inputs and sanity checks.

We started with a training and familiarization procedure for the analysts. The process involved on-boarding and training to establish a clear mutual understanding of the expected level of analysis and the format of decision inputs to be recorded in the ground truth. We provided analysts with video and text tutorials, accompanied by a toy example implemented within the system. Multiple ad hoc meetings and Q&A sessions were also held to further clarify the process and address any issues. Analysts were introduced to example crowd-sourced analyses (Schweinsberg et al., 2021; Silberzahn et al., 2018) to align their mental models with justifiable alternative decisions and the model quality level.

Collaborative efforts were encouraged in curating and shaping the datasets, research questions, and meta-information. In the review and revision phase, we shared input from other annotators and presented LM-generated examples ($n \approx 40$ per annotator, per dataset) for analysts to label as correct or incorrect. This process helped identify gaps, promote diversity, and encourage the incorporation of additional justifiable decisions. Analysts labeled the generated examples as justifiable or not justifiable, drawing inspiration from their peers and LM-generated outputs. The diversity in familiarity with various analysis methods among the analysts complemented each other, resulting in a more robust set of annotations.

At the end of the annotation, we collected 118 conceptual variable decisions, 246 discrete transform decisions, and 172 modeling decisions (i.e., choice of statistical model and model formula).

A.3 Analysis Decision Representations

In this section, we formally describe the representation of different analysis decisions as described in Section 5. These representations capture all alternative approaches in our ground truth and are matched with an agent's generated analysis artifacts (Sec. 4.2).

Data Transformations. Formally, a transform data flow graph is a bipartite graph with two types of nodes: *transform* nodes and *column pointer* nodes.

$$G = (T \cup P,\ E) \tag{1}$$

where $T = \{t_1, t_2, \ldots\}$ is the set of transforms. $P = \{p_1, p_2, \ldots\}$ is the set of column pointers, and the set of edges is denoted by:

$$E \subseteq (T \times P) \cup (P \times T) \tag{2}$$

Each transform t represents one unit of transformation and is defined by a fixed set of transform verbs V (Table 6). This set of transform verbs is based on existing data wrangling libraries (arq, 2021; Satyanarayan et al., 2016), were validated to cover every analysis decision in our benchmark, and represent a discrete data transformation decision that was made in wrangling the data.

Given our graph, we ultimately want to match based on the column values as a result of any series of transformations. Thus, each column pointer holds the column values and facilitates the flow of column values from transform to transform. We denote the column vector value at a column pointer node p as $\mathbf{v}_p$ and S as the set of all column values associated with G.

$$S = \{\mathbf{v}_p \mid p \in P\} \tag{3}$$

The set of input column pointers to a transform t and the output column pointers from a transform t are defined by $I(t)$ and $O(t)$:

$$I(t) = \{p \in P \mid (p, t) \in E\}, \tag{4}$$

$$O(t) = \{p \in P \mid (t, p) \in E\}. \tag{5}$$

The exact transform performed dictates $I(t)$ and $O(t)$. Specifically, $O(t)$ reflects only the columns that are changed by t and $I(t)$ are the columns that are necessary to compute the output $O(t)$.

Our transform data flow graph satisfies the following properties:

$$\begin{aligned} &|\ I(t)| > 0 \\ &|O(t)| \geq 1 \\ &|I(p)| = 1 \text{ except for original columns} \end{aligned}$$

So far, a single data flow graph G, represents a unique series of transformations. To account for all alternative transformation choices (e.g., an alternative in which the filter step ③ in Fig. 2 is skipped), we define $\boldsymbol{\mathcal{G}} = \{G_1, G_2, \ldots, G_n\}$ to be the set representing all unique series of transformations for an analysis. Note that any two graphs $G_i = (T_i, E_i)$ and $G_j = (T_j, E_j)$ may contain the same transformation (e.g., two graphs can contain the same derive rater average transform ①) and so $T_i \cap T_j \neq \emptyset$.

Finally, to keep track of all transformations across all justifiable alternatives, we define $\boldsymbol{\mathcal{T}}$ and $\boldsymbol{\mathcal{S}}$ to be the set of all transformations and columns values, respectively, across all data flow graphs. Any agent benchmark submission will be matched against these ground-truth representations (described in Appendix A.4.1).

$$\boldsymbol{\mathcal{T}} = \bigcup_{i=1}^{n} T_i \qquad \boldsymbol{\mathcal{S}} = \bigcup_{i=1}^{n} S_i \tag{6}$$

Conceptual Variables. A conceptual variable $c \in \boldsymbol{C}$ is a triplet $(c_{desc}, c_{type}, C_{cols})$ where c_{desc} is a natural language description of the conceptual variable, $c_{type} \in \{\texttt{IV}, \texttt{DV}, \texttt{Control}\}$ is the variable type, and $C_{cols} \subseteq \boldsymbol{\mathcal{S}}$ is the set of column vectors that operationalize c. Here, $\boldsymbol{C}$ denotes the set of conceptual variables across all alternative approaches.

Statistical Models. A statistical model $m \in \boldsymbol{M}$ is a tuple (m_{desc}, M_{cols}) where m_{desc} is the natural language description of the statistical model and $M_{cols} \subseteq C_{cols}$ is a set of column vectors associated with the model which are also associated with a conceptual variable. In addition, M_{cols} should be associated with only one series of transformations or one data flow graph, that is:

$$\exists S_i \in \{S_1, S_2, \ldots, S_n\} \mid M_{cols} \subseteq S_i \tag{7}$$

From M_{cols}, we can also derive the associated conceptual variables $C_m \subseteq \boldsymbol{C}$ in a model.

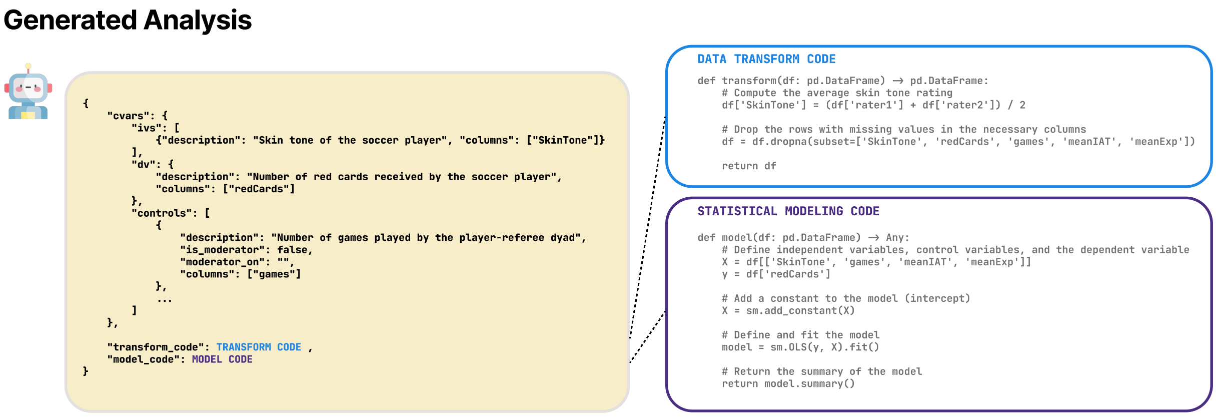
$$C_m = \{c_i \mid C_{cols,i} \cap M_{cols} \neq \emptyset\} \tag{8}$$

In addition, for each statistical model m, there is one associated variable that is a DV, at least one associated variable that is an IV and 0 or more Control variables. $\boldsymbol{M}$ denotes the set of statistical models across all alternative approaches.

A.4 Decision Matching Procedure

With an understanding of the representations of different analysis decisions, we now describe a procedure to match an agent-generated analysis to the ground truth.

Given the agent submission artifacts (Fig. 7), we first apply LMs to handle the conversion of generated artifacts into our ground truth representation format. Specifically, we use GPT-4 to perform two tasks: convert the transform function into individual transform units, and translate the modeling function into a statistical model specification (e.g.,



```
{
    "cvars": {
        "ivs": [
            {"description": "Skin tone of the soccer player", "columns": ["SkinTone"]}
        ],
        "dv": {
            "description": "Number of red cards received by the soccer player",
            "columns": ["redCards"]
        },
        "controls": [
            {
                "description": "Number of games played by the player-referee dyad",
                "is_moderator": false,
                "moderator_on": "",
                "columns": ["games"]
            },
            ...
        ]
    },

    "transform_code": TRANSFORM CODE ,
    "model_code": MODEL CODE
}
```

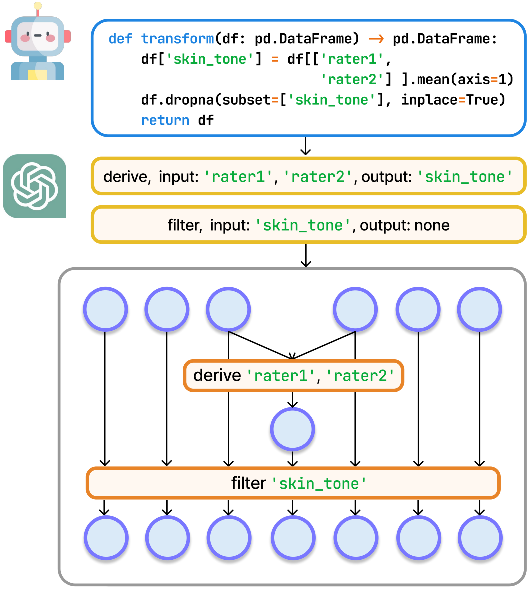
DATA TRANSFORM CODE

```
def transform(df: pd.DataFrame) → pd.DataFrame:
    # Compute the average skin tone rating
    df['SkinTone'] = (df['rater1'] + df['rater2']) / 2

    # Drop the rows with missing values in the necessary columns
    df = df.dropna(subset=['SkinTone', 'redCards', 'games', 'meanIAT', 'meanExp'])

    return df
```

STATISTICAL MODELING CODE

```
def model(df: pd.DataFrame) → Any:
    # Define independent variables, control variables, and the dependent variable
    X = df[['SkinTone', 'games', 'meanIAT', 'meanExp']]
    y = df['redCards']

    # Add a constant to the model (intercept)
    X = sm.add_constant(X)

    # Define and fit the model
    model = sm.OLS(y, X).fit()

    # Return the summary of the model
    return model.summary()
```

Figure 7: Example of the full analysis submission to BLADE.



```
def transform(df: pd.DataFrame) → pd.DataFrame:
    df['skin_tone'] = df[['rater1',
                          'rater2'] ].mean(axis=1)
    df.dropna(subset=['skin_tone'], inplace=True)
    return df
```

derive, input: 'rater1', 'rater2', output: 'skin_tone'
filter, input: 'skin_tone', output: none
derive 'rater1', 'rater2'
filter 'skin_tone'

Figure 8: Given a transform function from a graph (top), we first use an LM (GPT-4o) to convert the transform into individual transform units with verb and column specifications (middle). Using this information, we then derive the column data flow graph G (bottom).

linear regression) along with the columns used in the model (Fig. 8).

Next, we describe the procedure to match a given analysis to the ground truth. Specifically, given the ground truth $\mathcal{G}$, $\mathcal{T}$, $\boldsymbol{C}$, and $\boldsymbol{M}$, we describe matching a single analysis containing G', T', $\boldsymbol{C}'$, and $\boldsymbol{M}'$.

A.4.1 Matching Transforms

Data transformations are inherently open-ended with multiple valid approaches and free-form responses. Our goal is to capture how well agents perform in the underlying data analysis decisions. Therefore, we define multiple approaches to capture different levels of performance (i.e., getting the exact column vector vs. rough same steps) in how well a given analysis matches with the decisions in the ground truth: value matching and graph matching.

In both matching schemes, we determine whether a match occurs (i.e., based on matching column values or the graph structure based on the transform specification) and match all upstream transformations based on the data flow graph G.

Here, in order to evaluate the quality of a series of transform T' in G', we attempt to identify ground truth transforms $t \in \mathcal{T}$ associated with $\mathcal{G}$ that matches with $t' \in T'$, that is, $Match(t) = 1$ and $Match(t') = 1$.

To match all transforms $\mathcal{T}$ in all specified alternatives with T', as the transforms are situated in the graphs, we perform all pairwise matching between $G \in \mathcal{G}$ and G'.

Value Matching. In value matching, we want to match two *series* of transformations if they result in the same column value.

Given S and S' denoting the sets of column vectors associated with G and G', if $\mathbf{v}_p \in S = \mathbf{v}_{p'} \in S'$ (i.e., all cell values are equal when comparing two column vectors at column pointer nodes p and p'), then this means that the series of transformations that resulted in $\mathbf{v}_p$ and $\mathbf{v}_{p'}$ are equivalent. Therefore, all parents transforms of p in G and p' in G' should be matched.

Let $I(p)^+$ denote the set of transforms in the transitive closure of p and its ancestors: $I(p) = \{t \in T \mid T \in I(p) \text{ or } T \in I(I(I(p)))\ldots\}$. If $\mathbf{v}_p = \mathbf{v}_{p'}$, then $I(p) \subseteq T$ is matched and $I(p') \subseteq T'$ are matched.

$$Match(t) = 1 \ \ \forall t \in I(p)^{+} \text{ and}$$
$$Match(t') = 1 \ \ \forall t \in I(p')^{+}$$

While it may be the case that there are other column values involved in $O(t)$ which may differ, we at least know for sure that two series of transformations produced the same column value. In addition, because the definition of each t is set to only include the affected columns, we find that the match of values in two pairs of columns is a sufficient criterion for equivalence.

Fuzzy Graph Isomorphism Matching. Value matching may be considered to be too strict, especially when small changes in the numerical parameters of a transform can lead to different column values (e.g., in Fig. 2, filter on `rpg > 0.5` vs. `rpg > 0.45`). To allow greater flexibility in the matching, we introduce fuzzy graph matching. In graph matching, we match based on the transform verbs and column specifications rather than the exact column values (e.g., choosing to filter on `rpg` and `r_avg` after steps ① and ②). More specifically, if two series transforms shared the same high-level definition in which transforms are used in a similar way defined by the transform verb and parameter columns and dataflow, then they should be equivalent.

To accomplish this, we add a node label mapping $L : T \rightarrow V \times P^{n}$ mapping the transform to its associated transform verb and column pointer parameters (e.g., step ③ in Fig. 2 would have the node label (filter, $\{p_{\texttt{rpg}}, p_{\texttt{r\_avg}}\}$) where $p_{\texttt{rpg}}$ is the column node associated with the `rpg` column and $p_{\texttt{r\_avg}}$ is the column node associated with the `r_avg` column). Given this definition, if a subgraph is equivalent to another subgraph, then this means they represent the same choices of transforms (at a higher-level of abstraction relative to Value Matching).

More formally, let $H(t)$ denote the subgraph induced by the transitive closure of t and its parents. $H(t)$ captures both the transform nodes and the relevant column pointer nodes. If $H(t)$ is isomorphic to $H(t')$, including the node labels added from L and L', then all t in $H(t)$ and t in $H(t')$ are matched.

$$Match(t) = 1 \ \ \forall t \ \text{ in the graph } \ H(t)$$
$$Match(t') = 1 \ \ \forall t \ \text{ in the graph } \ H(t')$$

A.4.2 Matching Conceptual Variables

Given $c \in \boldsymbol{C}$ and $c' \in \boldsymbol{C}'$, c and c' are equivalent if $c_{type} = c'_{type}$ and c_{desc} and c'_{desc} are semantically equivalent. For practical purposes, we use a language model to determine semantic equivalence. Specifically, we use GPT-4o following Liang et al.'s (2023) prompting approach.

We input JSON-formatted conceptual variable specifications for $\{c_{desc} \mid c \in \boldsymbol{C}\}$ and $\{c_{desc} \mid c \in \boldsymbol{C}'\}$. The LM then generates a JSON output where containing the pair of matching point IDs, and an associated similarity value providing the explanation for the match (see Fig. 11 for the prompt).

A.4.3 Matching Statistical Models

Given $m \in \boldsymbol{M}$ and $m' \in \boldsymbol{M}'$, we define two levels of matching: semantic and conceptual model-based matching. First, m and m' are semantically matched if m_{desc} and m'_{desc} are semantically equivalent following the same matching procedure for conceptual variables (see Fig. 15 for the prompt). This represents a coarse level of matching.

As determining the choice of a justifiable model involves including the right conceptual variables in the model, we then perform matching based on the conceptual variables (Appendix. A.4.2) associated with the model.

A.5 Baseline ReAct Agent Details

The baseline framework is an ReAct agent with `[Thought]`, `[Action]`, and `[Observation]` stages before a final `[Finish]` stage. The initial `[Thought]` stage integrates the current context (i.e., latest observation) and the prior outputs (i.e., history of thoughts, actions, and observations) to formulate the next step action. Next, with the `[Action]` tag, the LM calls the underlying notebook and executes a new cell with the new LM-generated code. The `[Observation]` then comes from the notebook environment and is the string representation of the last-line output in the code following Yin et al., 2022. This cycle repeats until the LM decides to output the final analysis with the `[Finish]` tag. The prompt for the agent includes one example of a ReAct trajectory (`[Thought] -> [Action] -> [Observation]`) that iteratively explores the data. See Figure 18 for the prompt template.

The notebook sandbox environment uses Python 3.10 with the following imports:

```
import pandas as pd
import sklearn
import scipy
import statsmodels.api as sm
import statsmodels.formula.api as smf
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

These imports were determined during development such that the code generations do not involve any import errors on the main coding libraries.

Compared to the one-turn setting, the ReAct agent can explore the data more closely. In our experiments, we allow the agent to perform up to 10 steps, interacting with the environment with the full context of prior actions and observations. Based on preliminary experiments, we determined that the ReAct agent needed LMs with at least an 8k context window to handle multiple turns of code execution outputs. Because of this, we performed experiments with the ReAct framework on the following LMs: Mixtral-8x22b, GPT-3.5 Turbo, GPT-4o, Gemini 1.5 Pro, and Claude 3.5 Sonnet.

A.6 Prompt Templates

The following figures (Figures 9-19) show the various prompt templates used in the construction and evaluation of BLADE. The prompts in Figure 9 and Figure 10 are used to elicit alternative conceptual variables and data transformations in the benchmark data collection (Sec. 3).

The next set of prompts are used in the automatic evaluation of BLADE(Figures 11-16). Figure 11 shows the prompt to semantically match conceptual variables. Figure 12 and 13 show the prompt for converting the agent's transformation function submission (e.g., Fig. 7). Figure 14 shows the prompt to convert the statistical modeling function into a natural language specification of the model and the columns in the transformed data table that are used in modeling. Finally, Figure 15 shows the prompt used to semantically match statistical models.

We also include the prompts for our evaluation tasks. Figure 16 shows the instructions to generate the entire analysis, while Figure 18 shows our implementation of the ReAct framework, which guides an AI assistant through reasoning and action steps for data analysis tasks. Figure 19 gives an example of our MCQ prompt.

Most of these prompts utilize a JSON representation of Pydantic objects for standardized formatting, leveraging Langchain's Pydantic parser[3]. Additionally, the schema of the dataset is represented as a JSON object, generated using the data summarizer from Dibia, 2023. Figure 13 provides a detailed description of the transformation API used in the prompt for Figure 12, specifying the available transformation verbs and their corresponding input/output mappings. Figure 17 provides the one-shot example to guide the LM in generating an analysis (i.e,. the prompt in Fig. 16).

A.7 Evaluation Metrics Details

Average Precision. Average precision is calculated as the mean of the precision scores across all individual runs. For a decision type (i.e., conceptual variables, transformations, statistical modeling) and a given set of agent-submitted decisions for runs $\{R_1, R_2, \ldots, R_n\}$ with a corresponding ground truth set G, the precision for each run R_i is calculated as:

$$\text{Precision}(R_i) = \frac{|R_i \cap G|}{|R_i|} \tag{9}$$

The average precision p_{avg} is then computed as:

$$p_{\text{avg}} = \frac{1}{n} \sum_{i=1}^{n} \text{Precision}(R_i) \tag{10}$$

Coverage@k. Coverage@k is defined as the proportion of the ground truth set that is covered by the union of items across a sample of *k* randomly selected runs (assuming the total number of runs $n > k$. Specifically, for a decision type and agent submitted decisions across a sample of k runs $\{R_1, R_2, \ldots, R_k\}$, $coverage@k$ is calculated as:

$$coverage@k = \frac{\left|\bigcup_{i=1}^{k} R_i \cap G\right|}{|G|} \tag{11}$$

For modeling decisions in which each run has one submission, the denominator is $\min(|G|, k)$.

In our experiments, we report coverage@10 for several reasons. First, we manually determined that for all datasets in BLADE, conceptual variable and transformation decisions can be adequately covered in 10 runs. In addition, generating 10 independent analyses represents a reasonable and realistic scenario, mirroring a situation where one might

[3] https://python.langchain.com/v0.1/docs/

leverage crowd-sourced analyses from 10 different analysts.

F1-score. To reflect the overall performance while balancing precision and coverage, we compute F1-score calculated as follows:

$$F1 = \frac{2 \times (p_{\text{avg}} \times coverage@k)}{p_{\text{avg}} + coverage@k} \qquad (12)$$

To capture performance on BLADE in a single metric, for each decision type, we first take p_{avg} and $coverage@10$ averaged across all datasets and calculate the F1-score. Next, we take the *weighted-averaged* F1-score based on the number of ground truth decisions for each decision type. For statistical modeling decisions, the weight is based on $\min(|G_{model}|, 10)$.

Bootstrap Estimates and Confidence Intervals. To account for the variability in selecting subsets of runs (especially for computing coverage@10), we employed a bootstrap procedure to estimate the expected F1-score and its confidence intervals. Specifically, we performed $m = 1000$ iterations of random sampling with replacement from the set of runs for each dataset. In each iteration, we recalculated both average precision and coverage@10, and then computed the corresponding F1-score. The final reported F1-score is the average of these bootstrap iterations, with a 95% confidence interval derived from the distribution of the bootstrap samples.

A.8 Case Studies with Qualitative Insights

To gain additional insight into the performance of LMs, two of the annotators sampled 56 output files from LM-generated results for qualitative case studies. Our findings reveal several limitations in LMs' ability to generate robust and reliable analyses:

1. **Composite Variables:** In the TeachingRatings dataset (Figure 20-1, Figure 20-2), GPT-4 failed to create important composite variables, such as evaluation response rate, despite their interpretability and explanatory power. LLMs often included only one of the component variables.

2. **Interaction Effects:** GPT-3.5 (Figure 20-3) struggled with understanding interaction effects in linear regression models, often including irrational interaction terms without main effects (e.g., `eval ~ beauty * gender`).

3. **Variable Selection:** While GPT-4 provided more comprehensive models with most control variables (see one example in Figure 20-4), it sometimes included redundant variables (e.g., "relative group size" derived from "n_focal" and "n_other") (Figure 21-5). In contrast, GPT-3.5 often used very minimal models (only one IV with no controls) (Figure 21-6).

Dataset	Domain	Keywords	Research Question	Source paper
hurricane	Behavioral Sciences	hurricane names, gender stereotypes, risk perception, natural disasters	Hurricanes with more feminine names are perceived as less threatening and hence lead to fewer precautionary measures by the general public.	(Jung et al., 2014; Malter, 2014; Maley, 2014; Bakkensen and Larson, 2014; Simonsohn et al., 2020)
mortgage	Finance and Economics, Demographics	lending discrimination, redlining, credit risk, fair housing	How does gender affect whether banks approve an individual's mortgage application?	(Liu et al., 2020b; Munnell et al., 1996; Young and Holsteen, 2017)
soccer	Behavioral Sciences	skin tone, racial bias, referee decisions, sports analytics	Are soccer players with a dark skin tone more likely than those with a light skin tone to receive red cards from referees?	(Silberzahn et al., 2018; Auspurg and Brüderl, 2021)
reading	Education	dyslexia, web accessibility, reading comprehension, user experience	Does 'Reader View' – a modified web page layout – improve reading speed for individuals with dyslexia?	(Li et al., 2019; Liu et al., 2020b)
Fish	Health and Well-being	recreational fishing, environmental conservation, visitor demographics, count data analysis	How many fish on average do visitors take per hour, when fishing?	(McElreath, 2018)
AMTL	Evolutionary Biology	antemortem tooth loss, fossil hominins, dental anthropology, comparative anatomy	Do modern humans have higher frequencies of antemortem tooth loss compared to non-human primate genera after accounting for the effects of age, sex, and tooth class?	(Gilmore, 2013; McElreath, 2018; Konigsberg and Frankenberg, 2013)
Boxes	Education, Behavioral Sciences	cultural transmission, social learning biases, cognitive development, cross-cultural research	How do children's reliance on majority preference develop over growth in age across different cultural contexts?	(Van Leeuwen et al., 2018; McElreath, 2018)
Crofoot	Evolutionary Biology	intergroup competition, territorial behavior, spatial analysis, animal tracking	How do relative group size and contest location influence the probability of a capuchin monkey group winning an intergroup contest?	(Crofoot et al., 2008; McElreath, 2018)
Panda_nuts	Evolutionary Biology	tool use, skill acquisition, social learning, primate cognition	How do age, sex, and receiving help from another chimpanzee influence the nut-cracking efficiency of western chimpanzees?	(Boesch et al., 2019; McElreath, 2018)
Affairs	Behavioral Sciences, Demographics	infidelity, marital satisfaction, sexual behavior, limited dependent variables	Does having children decrease (if at all) the engagement in extramarital affairs?	(Fair, 1978; Kleiber and Zeileis, 2008; Long and Freese, 2006)
CASchools	Education	standardized testing, school resources, achievement gap, education policy	Is a lower student-teacher ratio associated with higher academic performance?	(Kleiber and Zeileis, 2008; Stock and Watson, 2020)
TeachingRatings	Education	student evaluations, instructor characteristics, gender bias, higher education	What is the impact of beauty on teaching evaluations received by teachers?	(Simonsohn et al., 2020; Hamermesh and Parker, 2005; Kleiber and Zeileis, 2008; Stock and Watson, 2020)

Table 3: Open-ended scientific research questions in BLADE across different domains.

BLADE			
Question	Are soccer players with a dark skin tone more likely than those with a light skin tone to receive red cards from referees?	How do age, sex, and receiving help from another chimpanzee influence the nut-cracking efficiency of western chimpanzees?	Does 'Reader View' – a modified web page layout – improves reading speed for individuals with dyslexia?
Answer(s)	*16* conceptual variables, *77* transformations, and *41* modeling decisions	*11* conceptual variables, *19* transformations, and *31* modeling decisions	*18* conceptual variables, *24* transformations, and *12* modeling decisions
ARCADE (Yin et al., 2022)			
Question	How many male and female employees are born in 1992?	Which countries host at least two Olympic games?	What is the most expensive phone in each brand?
Answer(s)	Two number counts	A list of country names	Dataframe of brand, model and price
DABench (Hu et al., 2024b)			
Question	Calculate the correlation coefficient between the "High Price" column and the "Low Price" column.	Calculate the mean fare paid by the passengers.	Categorize passengers into age groups and calculate mean fare for each group.
Answer(s)	["relationship_type", "linear"], ["correlation_coefficient", "0.99"]	["mean_fare", "34.65"]	["mean_fare_elderly", "43.47"], ["mean_fare_teenager", "31.98"], ["mean_fare_child", "31.09"], ["mean_fare_adult", "35.17"]
MLAgentBench (Huang et al., 2023b)			
Question	`[CIFAR-10]` Given a training script on a dataset train.py, improve upon the current model performance (trained with current hyperparmeters in train.py). The training epochs should be within 10 to save time.	`[Feedback]` Go through the data_description.txt file to understand the data and all the features. You can summarize it in your research logs to keep track of what all you have to do. Then fill in the provided train.py script to train a model and iterate over different models or feature selections to get a better performance.	`[IMDB]` Fill out train.py to 1) finetune DistilBERT on the IMDb dataset to determine whether a movie review is positive or negative and 2) save per class probabilities for test set examples to submission.csv.
Answer(s)	Predictions for ML classification	Predictions for ML regression	Predictions for ML classification
DS-Agent (Guo et al., 2024b)			
Question	`[Airline Reviews]` You are solving this machine learning tasks of regression: The dataset presented here (Airline reviews) comprises customer feedback for British Airways. Here, we provide the text reviews. Your task is to predict the corresponding rating in the range of 1-10 given the reviews in the test set. The evaluation metric is root mean squared error (RMSE).	`[Bitcoin Price Prediction]` You are solving this machine learning tasks of regression: The dataset presented here (the BTC News to Bitcoin Price dataset) comprises a series of BTC news title. Your task is to predict the bitcoin price based on the given BTC news title in the test set. The evaluation metric is root mean squared error (RMSE).	`[BoolQ]` You are solving this machine learning tasks of classification: The dataset presented here (the BoolQ dataset) comprises a series of passage-question pairs. Given a passage and a question, your task is to identify whether the question can be inferred from the passage, with 0 as False and 1 as True. The evaluation metric is accuracy.
Answer(s)	Predictions for ML regression	Predictions for ML regression	Predictions for ML classification
ML-Benchmark for Data Interpreter (Hong et al., 2024b)			
Question	`[Titanic]` This is a Titanic passenger survival dataset, and your goal is to predict passenger survival outcomes. The target column is Survived. Perform data analysis, data preprocessing, feature engineering, and modeling to predict the target. Report accuracy on the eval data.	`[Santander Customer]` This is a customer's financial dataset. Your goal is to predict which customers will make a specific transaction in the future. The target column is the target. Perform data analysis, data preprocessing, feature engineering, and modeling to predict the target. Report AUC on the eval data.	`[Santander Value]` This is a medical dataset with over fifty anonymized health characteristics linked to three age-related conditions. Your goal is to predict whether a subject has or has not been diagnosed with one of these conditions.The target column is Class. Perform data analysis... the target. Report F1 Score on the eval data.
Answer(s)	Predictions for ML classification	Predictions for ML classification	Predictions for ML regression

Table 4: Comparison of task examples in BLADE and related benchmarks. BLADE prioritizes open-ended scientific research questions rather than ML prediction tasks or data analysis code execution, focusing on the analysis approach and allowing for multiple valid solutions.

Annotator ID	Current Occupation	Stats and Analysis Exp.	Analysis Frequency
A01	PhD student in Statistics	8 Years	A few times a week
A02	PhD student in Statistics	5 Years	A few times a week
A03	PhD student in Statistics	4 Years	A few times a week
A04	PhD student in Biomedical and Health Informatics	5 Years	A few times a week
A05	PhD student in Measurement, Evaluation, and Research Methodology	6 Years	A few times a week
A06	Master’s student in Communications	5 Years	A few times a week
A07	Master’s student in Statistics	6 Years	A few times a week
A08	Data Scientist in the Finance Industry	8 Years	Daily
A09	Data Scientist in the Tech Industry	8 Years	Daily
A10	Data Scientist in the Tech Industry	5 Years	Daily
A11	Quantitative Researcher in Finance	5 Years	Daily

Table 5: Expert level data annotation. All annotators have at least 4 years of experience in statistics and data analysis. In addition, they are either currently pursuing a postgraduate degree in a relevant scientific field or are regularly working with data in industry.

Verb	Description	Input Columns	Affected Output Column(s)	Example Code

Derive	Derive a new column value based on the provided expressions.	Mandatory	One	`# derive a new column ’sumXY’ by adding ’x’ and ’y’`
				`df[’sumXY’] = df[’x’] + df[’y’]`

Filter	Filter a table to a subset of rows based on the input criteria.	Optional	All	`# filter the dataframe to include only rows where ’x’ > 2`
				`df = df[df[’x’] > 2]`

Slice	Extract rows with indices from start to end (end not included).	Optional	All	`# slice the dataframe to include rows 2 to 4`
				`df = df.iloc[2:4]`

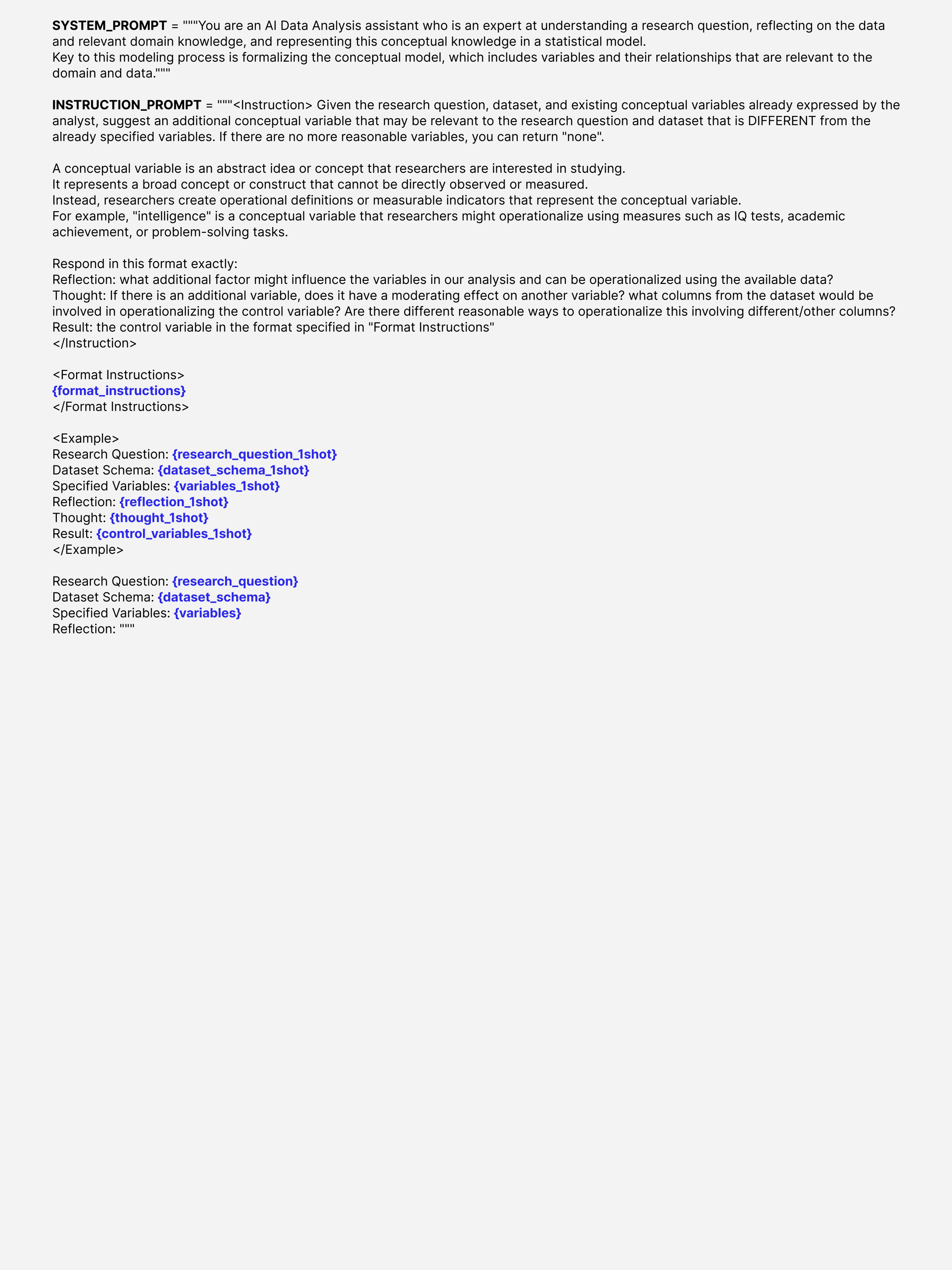
Groupby	Group table rows based on a set of column values. Groupby should return a pandas groupby object for subsequent operations.	Mandatory	All except groupby input columns	`# group the dataframe by ’x’`
				`grouped = df.groupby(’x’)`

De-duplicate	De-duplicate table rows by removing repeated row values.	Optional	All	`# remove duplicate rows in the dataframe`
				`df = df.drop_duplicates()`

Impute	Impute missing values or rows.	Optional	One/All	`# replace NaN values with a specific value`
				`df = df.fillna(0)`

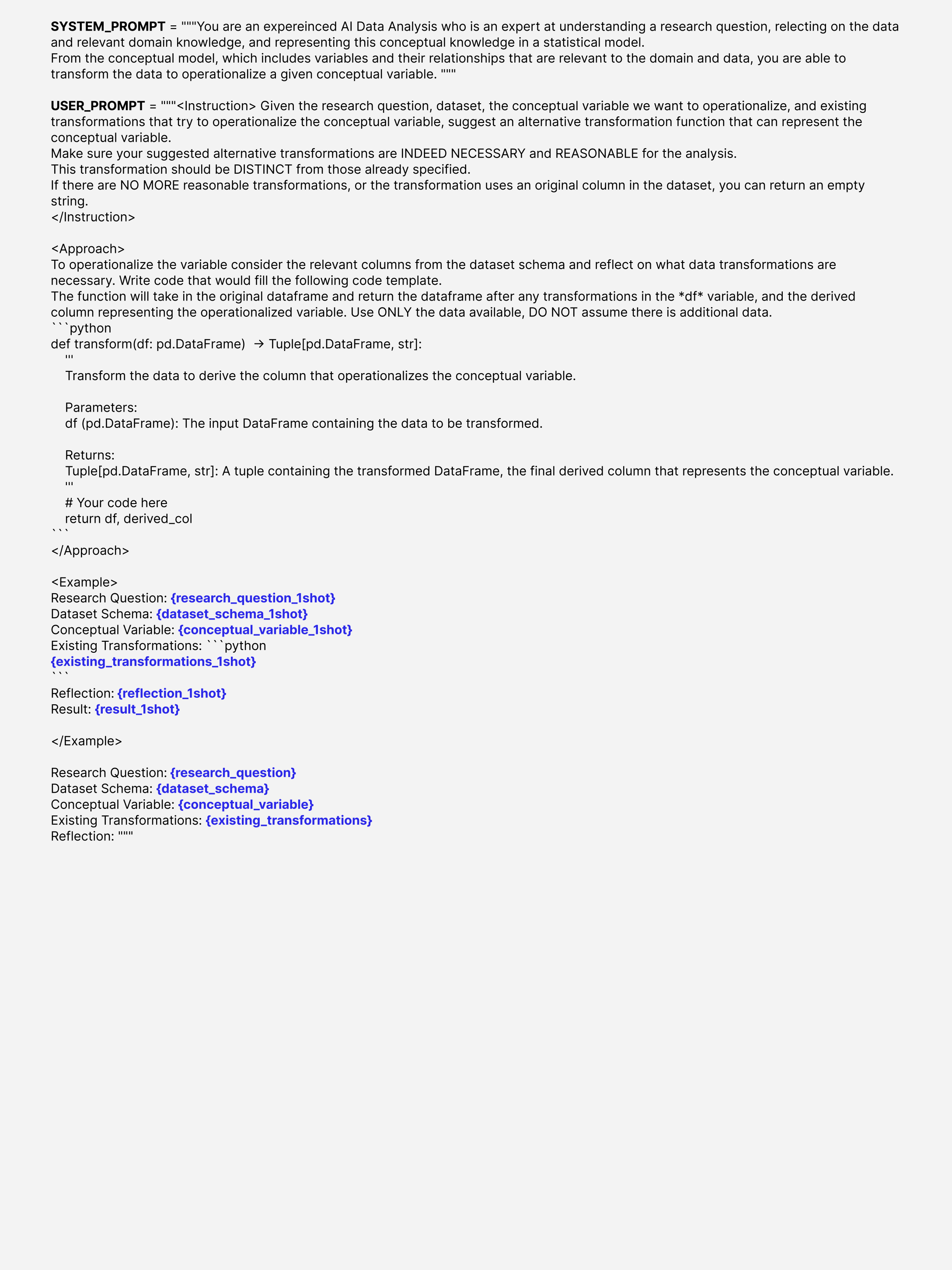
Rollup	Rollup a table to produce an aggregate summary. This is used with groupby when aggregating a group.	Mandatory	One	`df_grp = df.groupby(’x’)`
				`# rollup the grouped dataframe to get the mean of ’y’`
				`df = df_grp.agg(mean_y=(’y’, ’mean’)) .reset_index()`

Table 6: Taxonomy of transformation verbs utilized in the analysis ground truth. BLADE leverages these verbs in its evaluation to measure the nuance and complexity inherent in transformation approaches (Appendix A.4.1 explains our “fuzzy” transformation matching).



```
SYSTEM_PROMPT = """You are an AI Data Analysis assistant who is an expert at understanding a research question, reflecting on the data
and relevant domain knowledge, and representing this conceptual knowledge in a statistical model.
Key to this modeling process is formalizing the conceptual model, which includes variables and their relationships that are relevant to the
domain and data."""

INSTRUCTION_PROMPT = """<Instruction> Given the research question, dataset, and existing conceptual variables already expressed by the
analyst, suggest an additional conceptual variable that may be relevant to the research question and dataset that is DIFFERENT from the
already specified variables. If there are no more reasonable variables, you can return "none".

A conceptual variable is an abstract idea or concept that researchers are interested in studying.
It represents a broad concept or construct that cannot be directly observed or measured.
Instead, researchers create operational definitions or measurable indicators that represent the conceptual variable.
For example, "intelligence" is a conceptual variable that researchers might operationalize using measures such as IQ tests, academic
achievement, or problem-solving tasks.

Respond in this format exactly:
Reflection: what additional factor might influence the variables in our analysis and can be operationalized using the available data?
Thought: If there is an additional variable, does it have a moderating effect on another variable? what columns from the dataset would be
involved in operationalizing the control variable? Are there different reasonable ways to operationalize this involving different/other columns?
Result: the control variable in the format specified in "Format Instructions"
</Instruction>

<Format Instructions>
{format_instructions}
</Format Instructions>

<Example>
Research Question: {research_question_1shot}
Dataset Schema: {dataset_schema_1shot}
Specified Variables: {variables_1shot}
Reflection: {reflection_1shot}
Thought: {thought_1shot}
Result: {control_variables_1shot}
</Example>

Research Question: {research_question}
Dataset Schema: {dataset_schema}
Specified Variables: {variables}
Reflection: """
```

Figure 9: Prompt template A asking the LM to suggest an additional conceptual variable relevant to the research question and dataset. The format instructions asks the LM to generate a JSON representation of a Pydantic Object. Specifically we use Langchain's pydantic parser (https://python.langchain.com/v0.1/docs/modules/model_io/output_parsers/types/pydantic/) for the format instructions. The dataset schema is a, JSON representation of a data table. We use the data summarizer in LIDA (Dibia, 2023)



~~~~
SYSTEM_PROMPT = """You are an expereinced AI Data Analysis who is an expert at understanding a research question, relecting on the data
and relevant domain knowledge, and representing this conceptual knowledge in a statistical model.
From the conceptual model, which includes variables and their relationships that are relevant to the domain and data, you are able to
transform the data to operationalize a given conceptual variable. """

USER_PROMPT = """<Instruction> Given the research question, dataset, the conceptual variable we want to operationalize, and existing
transformations that try to operationalize the conceptual variable, suggest an alternative transformation function that can represent the
conceptual variable.
Make sure your suggested alternative transformations are INDEED NECESSARY and REASONABLE for the analysis.
This transformation should be DISTINCT from those already specified.
If there are NO MORE reasonable transformations, or the transformation uses an original column in the dataset, you can return an empty
string.
</Instruction>

<Approach>
To operationalize the variable consider the relevant columns from the dataset schema and reflect on what data transformations are
necessary. Write code that would fill the following code template.
The function will take in the original dataframe and return the dataframe after any transformations in the *df* variable, and the derived
column representing the operationalized variable. Use ONLY the data available, DO NOT assume there is additional data.
```python
def transform(df: pd.DataFrame) → Tuple[pd.DataFrame, str]:
    '''
    Transform the data to derive the column that operationalizes the conceptual variable.

    Parameters:
    df (pd.DataFrame): The input DataFrame containing the data to be transformed.

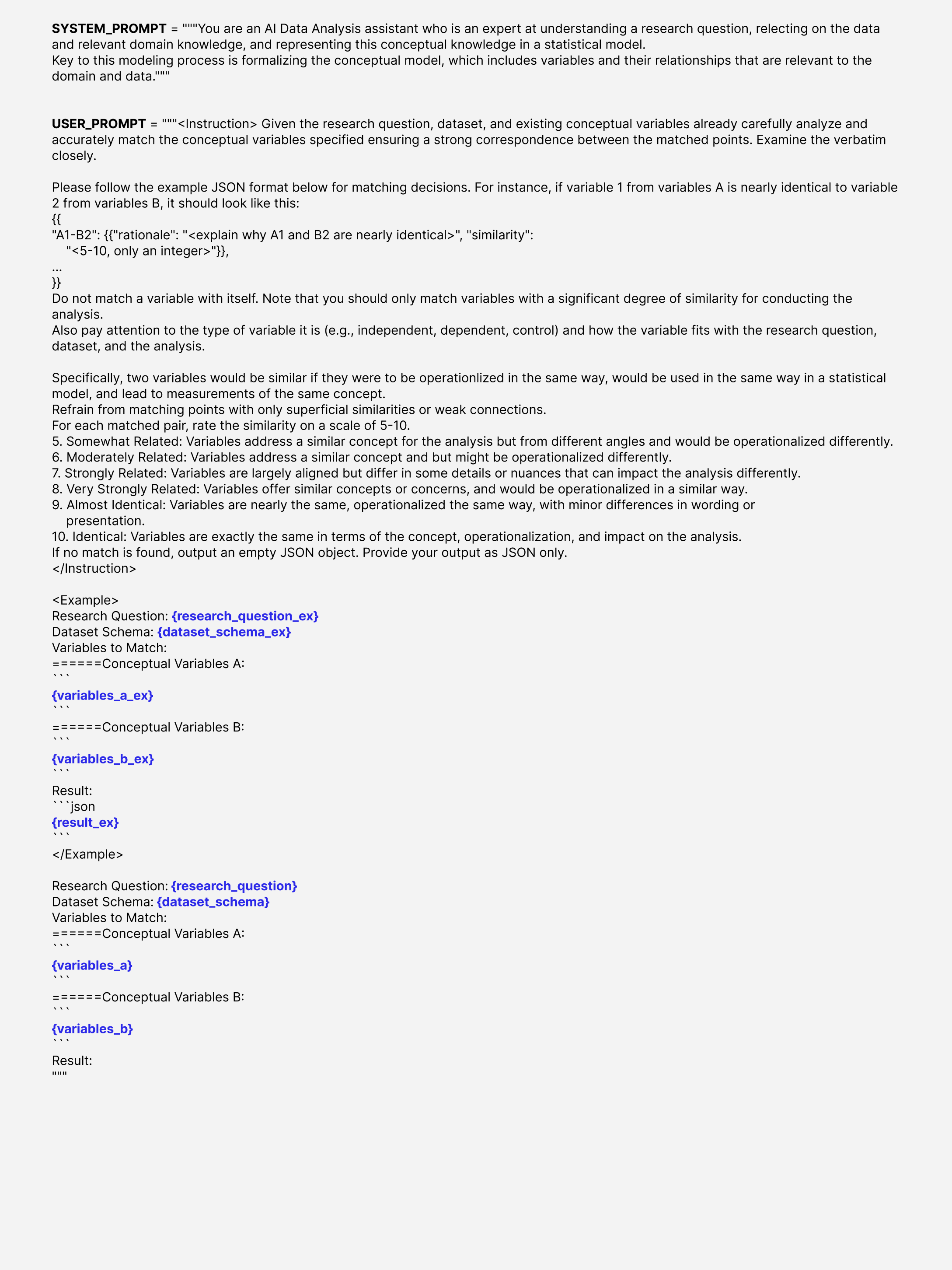
    Returns:
    Tuple[pd.DataFrame, str]: A tuple containing the transformed DataFrame, the final derived column that represents the conceptual variable.
    '''
    # Your code here
    return df, derived_col
```
</Approach>

<Example>
Research Question: {research_question_1shot}
Dataset Schema: {dataset_schema_1shot}
Conceptual Variable: {conceptual_variable_1shot}
Existing Transformations: ```python
{existing_transformations_1shot}
```
Reflection: {reflection_1shot}
Result: {result_1shot}

</Example>

Research Question: {research_question}
Dataset Schema: {dataset_schema}
Conceptual Variable: {conceptual_variable}
Existing Transformations: {existing_transformations}
Reflection: """
~~~~

Figure 10: Prompt template B asking the LM to suggest an alternative transformation in Python that transforms the given data columns to operationalize a conceptual variable.



~~~~
SYSTEM_PROMPT = """You are an AI Data Analysis assistant who is an expert at understanding a research question, relecting on the data
and relevant domain knowledge, and representing this conceptual knowledge in a statistical model.
Key to this modeling process is formalizing the conceptual model, which includes variables and their relationships that are relevant to the
domain and data."""


USER_PROMPT = """<Instruction> Given the research question, dataset, and existing conceptual variables already carefully analyze and
accurately match the conceptual variables specified ensuring a strong correspondence between the matched points. Examine the verbatim
closely.

Please follow the example JSON format below for matching decisions. For instance, if variable 1 from variables A is nearly identical to variable
2 from variables B, it should look like this:
{{
"A1-B2": {{"rationale": "<explain why A1 and B2 are nearly identical>", "similarity":
   "<5-10, only an integer>"}},
...
}}
Do not match a variable with itself. Note that you should only match variables with a significant degree of similarity for conducting the
analysis.
Also pay attention to the type of variable it is (e.g., independent, dependent, control) and how the variable fits with the research question,
dataset, and the analysis.

Specifically, two variables would be similar if they were to be operationlized in the same way, would be used in the same way in a statistical
model, and lead to measurements of the same concept.
Refrain from matching points with only superficial similarities or weak connections.
For each matched pair, rate the similarity on a scale of 5-10.
5. Somewhat Related: Variables address a similar concept for the analysis but from different angles and would be operationalized differently.
6. Moderately Related: Variables address a similar concept and but might be operationalized differently.
7. Strongly Related: Variables are largely aligned but differ in some details or nuances that can impact the analysis differently.
8. Very Strongly Related: Variables offer similar concepts or concerns, and would be operationalized in a similar way.
9. Almost Identical: Variables are nearly the same, operationalized the same way, with minor differences in wording or
   presentation.
10. Identical: Variables are exactly the same in terms of the concept, operationalization, and impact on the analysis.
If no match is found, output an empty JSON object. Provide your output as JSON only.
</Instruction>

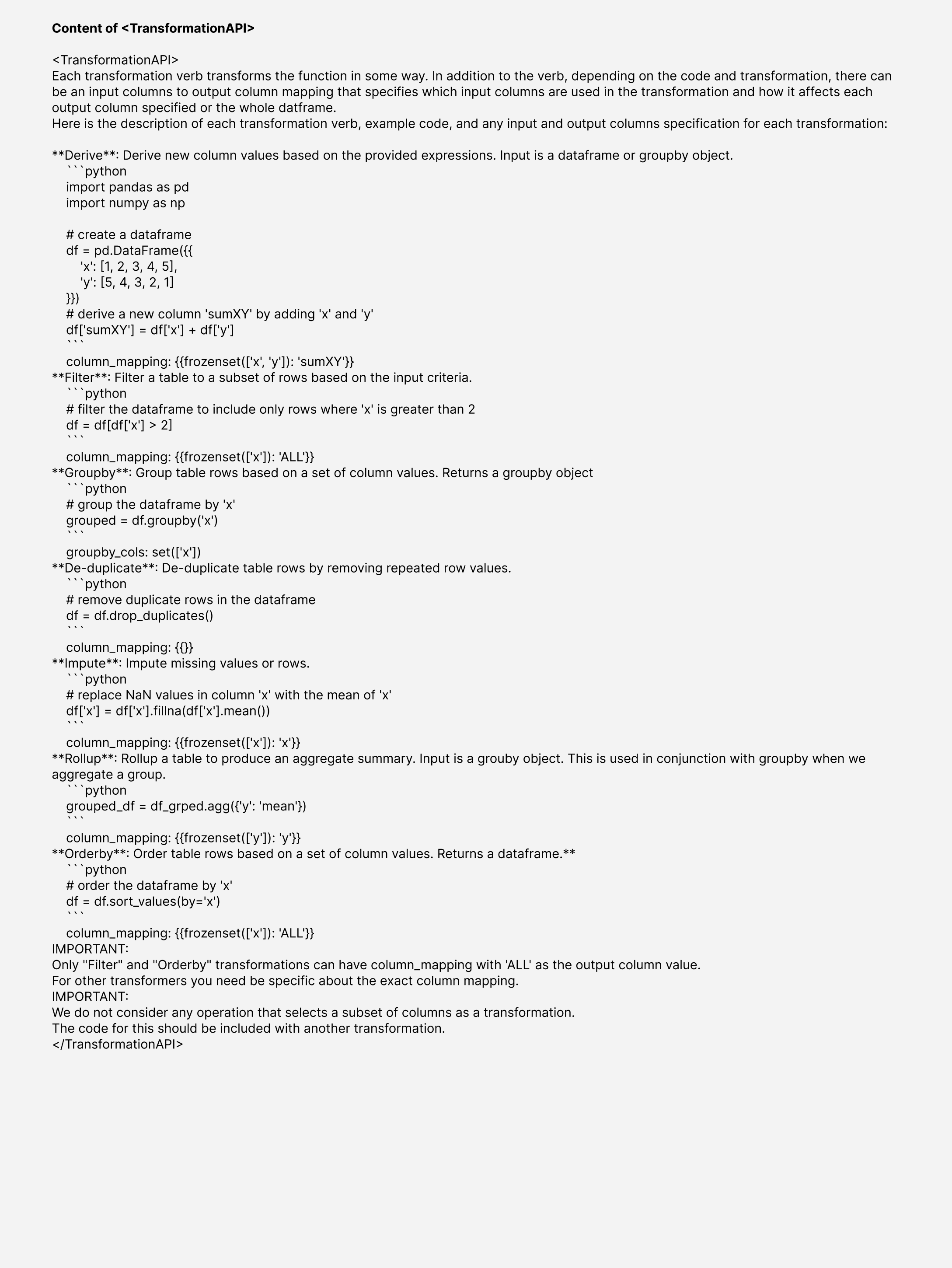
<Example>
Research Question: {research_question_ex}
Dataset Schema: {dataset_schema_ex}
Variables to Match:
======Conceptual Variables A:
```
{variables_a_ex}
```
======Conceptual Variables B:
```
{variables_b_ex}
```
Result:
```json
{result_ex}
```
</Example>

Research Question: {research_question}
Dataset Schema: {dataset_schema}
Variables to Match:
======Conceptual Variables A:
```
{variables_a}
```
======Conceptual Variables B:
```
{variables_b}
```
Result:
"""
~~~~

Figure 11: Prompt template C asking the LM to match conceptual variables from two given sets, considering their similarity in the context of the research question and dataset.

~~~
SYSTEM_PROMPT = """You are an AI Python Data Science assistant who is an expert at understanding data transformation code."""

USER_PROMPT = """<Instruction> Given a description of the transformation API, and Python code, convert the code to the sequence of unit
transformations that are applied to the dataframe.
Each unit function will take in and return a TransformDataReturn objet defined below:
```python
class TransformDataReturn(BaseModel):
   df: pd.DataFrame
   column_mapping: Dict[FrozenSet[str], str] # we can specify multiple input and output column mappings
   groupby_cols: Set[str] # only for groupby verb
   transform_verb: Literal['derive', 'filter', 'groupby', 'deduplicate', 'impute', 'rollup', 'orderby']
```
These transform functions will be called by the main "transform" function that will apply the transformations in sequence.
IMPORTANT: The transform functions (i.e., transform_funcs) cannot call each other or reference any variables outside of the function.
```python
def transform(df: pd.DataFrame,
          transform_funcs: List[Callable[[pd.DataFrame], TransformDataReturn]]):
   td_objs: List[TransformDataReturn] = []
   for func in transform_funcs:
      td_obj = func(df)
      df = td_obj.df
      td_objs.append(td_obj)
   return df
```
Your answer should be a list of transform functions of type Callable[[pd.DataFrame], TransformDataReturn] stored in the variable
"transform_funcs".
DO NOT ADD ANY NEW CODE THAT WAS NOT SPECIFIED IN THE ORIGINAL CODE.
For example, if the original code is empty then return empty.
If there is any statistical modeling code, do not include it in the transformation functions.
We are only interested in the code that transforms the dataframe.
</Instruction>

<TransformationAPI>

\\ Please refer to the next figure for content here. \\

</TransformationAPI>

<Example 1>
original code: ```python
{ex_code}
```
result: ```python
{ex_converted_code}
```
</Example 1>
<Example 2>
original code: ```python
{ex2_code}
```
result: ```python
{ex2_converted_code}
```
</Example 2>

\\ Example 3 - 5 with the same template are omitted here due to space limit. \\

<Example 6>
original code: ```python
{ex6_code}
```
result: ```python
{ex6_converted_code}
```
</Example 6>
<Example 7>
original code: ```python
{ex7_code}
```
result: ```python
{ex7_converted_code}
```
</Example 7>
original code: ```python
{code}
```
result:
"""
~~~

Figure 12: Prompt template D asking the LM to convert a given Python function for data transformation into a sequence of unit transformation functions, each taking a DataFrame as input and returning a TransformDataReturn object. Refer to Figure 13 for the content of TransformationAPI.



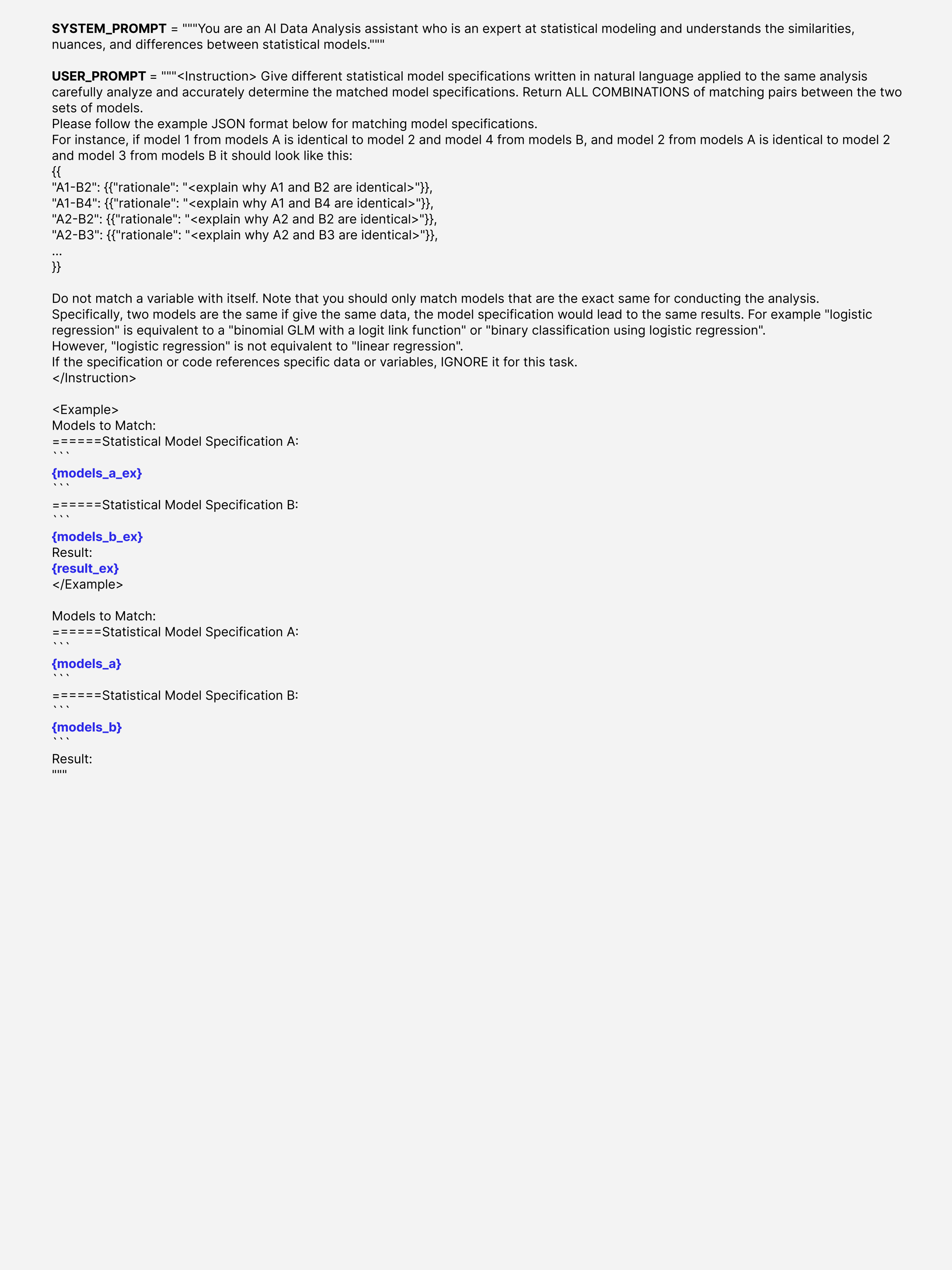
Content of <TransformationAPI>

~~~
<TransformationAPI>
Each transformation verb transforms the function in some way. In addition to the verb, depending on the code and transformation, there can
be an input columns to output column mapping that specifies which input columns are used in the transformation and how it affects each
output column specified or the whole datframe.
Here is the description of each transformation verb, example code, and any input and output columns specification for each transformation:

**Derive**: Derive new column values based on the provided expressions. Input is a dataframe or groupby object.
    ```python
    import pandas as pd
    import numpy as np

    # create a dataframe
    df = pd.DataFrame({{
        'x': [1, 2, 3, 4, 5],
        'y': [5, 4, 3, 2, 1]
    }})
    # derive a new column 'sumXY' by adding 'x' and 'y'
    df['sumXY'] = df['x'] + df['y']
    ```
    column_mapping: {{frozenset(['x', 'y']): 'sumXY'}}
**Filter**: Filter a table to a subset of rows based on the input criteria.
    ```python
    # filter the dataframe to include only rows where 'x' is greater than 2
    df = df[df['x'] > 2]
    ```
    column_mapping: {{frozenset(['x']): 'ALL'}}
**Groupby**: Group table rows based on a set of column values. Returns a groupby object
    ```python
    # group the dataframe by 'x'
    grouped = df.groupby('x')
    ```
    groupby_cols: set(['x'])
**De-duplicate**: De-duplicate table rows by removing repeated row values.
    ```python
    # remove duplicate rows in the dataframe
    df = df.drop_duplicates()
    ```
    column_mapping: {{}}
**Impute**: Impute missing values or rows.
    ```python
    # replace NaN values in column 'x' with the mean of 'x'
    df['x'] = df['x'].fillna(df['x'].mean())
    ```
    column_mapping: {{frozenset(['x']): 'x'}}
**Rollup**: Rollup a table to produce an aggregate summary. Input is a grouby object. This is used in conjunction with groupby when we
aggregate a group.
    ```python
    grouped_df = df_grped.agg({'y': 'mean'})
    ```
    column_mapping: {{frozenset(['y']): 'y'}}
**Orderby**: Order table rows based on a set of column values. Returns a dataframe.**
    ```python
    # order the dataframe by 'x'
    df = df.sort_values(by='x')
    ```
    column_mapping: {{frozenset(['x']): 'ALL'}}
IMPORTANT:
Only "Filter" and "Orderby" transformations can have column_mapping with 'ALL' as the output column value.
For other transformers you need be specific about the exact column mapping.
IMPORTANT:
We do not consider any operation that selects a subset of columns as a transformation.
The code for this should be included with another transformation.
</TransformationAPI>
~~~

Figure 13: Detailed description of the transformation API, specifying the available transformation verbs (derive, filter, groupby, de-duplicate, impute, rollup, and orderby) along with example code and input/output column mappings for each transformation. This is used in part of the prompt in Figure 12.

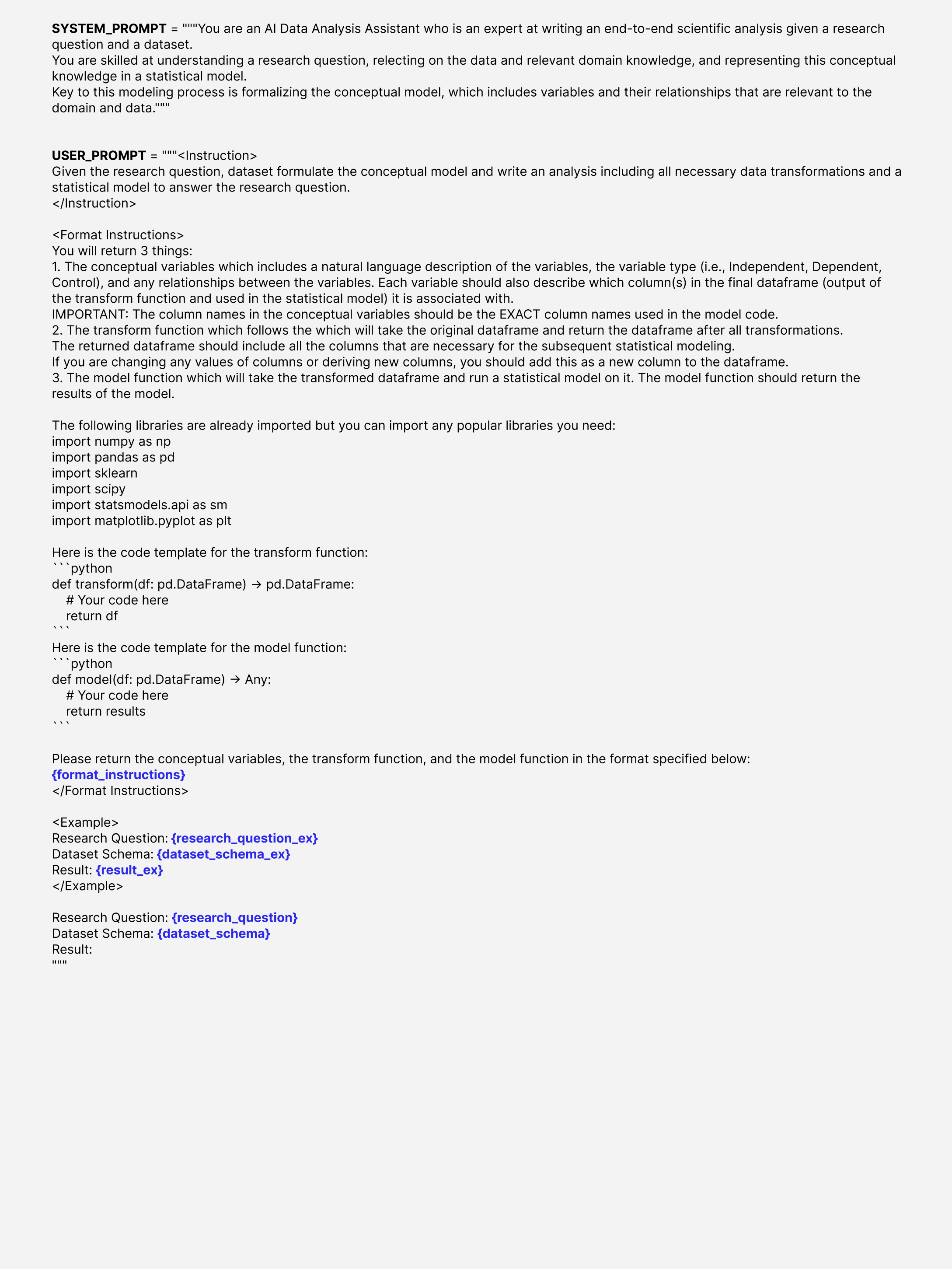
```
SYSTEM_PROMPT = """You are an AI Python Data Science assistant who is an expert at understanding statistical modeling code."""

USER_PROMPT = """<Instruction> Given the code snippet, carefully analyze and accurately determine the statistical model specification.
Return the model specification that best matches the code snippet. Please respond in the format specified by "Format Instructions".

IMPORTANT: the column names in the model specification should be the EXACT column names used in the model code.
</Instruction>

<Format Instructions>
{format_instructions}
</Format Instructions>

<Example 1>
Code Snippet: {code_snippet_ex}
Model Specification: {model_spec_ex}
</Example 1>
<Example 2>
Code Snippet: {code_snippet_2_ex}
Model Specification: {model_spec_2_ex}
</Example 2>
<Example 3>
Code Snippet: {code_snippet_3_ex}
Model Specification: {model_spec_3_ex}
</Example 3>

Code Snippet: {code_snippet}
Model Specification:
"""
```

Figure 14: Prompt template E instructing the LM to analyze code snippets and determine the corresponding statistical model specifications.



~~~
SYSTEM_PROMPT = """You are an AI Data Analysis assistant who is an expert at statistical modeling and understands the similarities,
nuances, and differences between statistical models."""

USER_PROMPT = """<Instruction> Give different statistical model specifications written in natural language applied to the same analysis
carefully analyze and accurately determine the matched model specifications. Return ALL COMBINATIONS of matching pairs between the two
sets of models.
Please follow the example JSON format below for matching model specifications.
For instance, if model 1 from models A is identical to model 2 and model 4 from models B, and model 2 from models A is identical to model 2
and model 3 from models B it should look like this:
{{
"A1-B2": {{"rationale": "<explain why A1 and B2 are identical>"}},
"A1-B4": {{"rationale": "<explain why A1 and B4 are identical>"}},
"A2-B2": {{"rationale": "<explain why A2 and B2 are identical>"}},
"A2-B3": {{"rationale": "<explain why A2 and B3 are identical>"}},
...
}}

Do not match a variable with itself. Note that you should only match models that are the exact same for conducting the analysis.
Specifically, two models are the same if give the same data, the model specification would lead to the same results. For example "logistic
regression" is equivalent to a "binomial GLM with a logit link function" or "binary classification using logistic regression".
However, "logistic regression" is not equivalent to "linear regression".
If the specification or code references specific data or variables, IGNORE it for this task.
</Instruction>

<Example>
Models to Match:
======Statistical Model Specification A:
```
{models_a_ex}
```
======Statistical Model Specification B:
```
{models_b_ex}
Result:
{result_ex}
</Example>

Models to Match:
======Statistical Model Specification A:
```
{models_a}
```
======Statistical Model Specification B:
```
{models_b}
```
Result:
"""
~~~

Figure 15: Prompt template F asking the LM to match statistical model specifications written in natural language, determining which models from two given sets are identical.

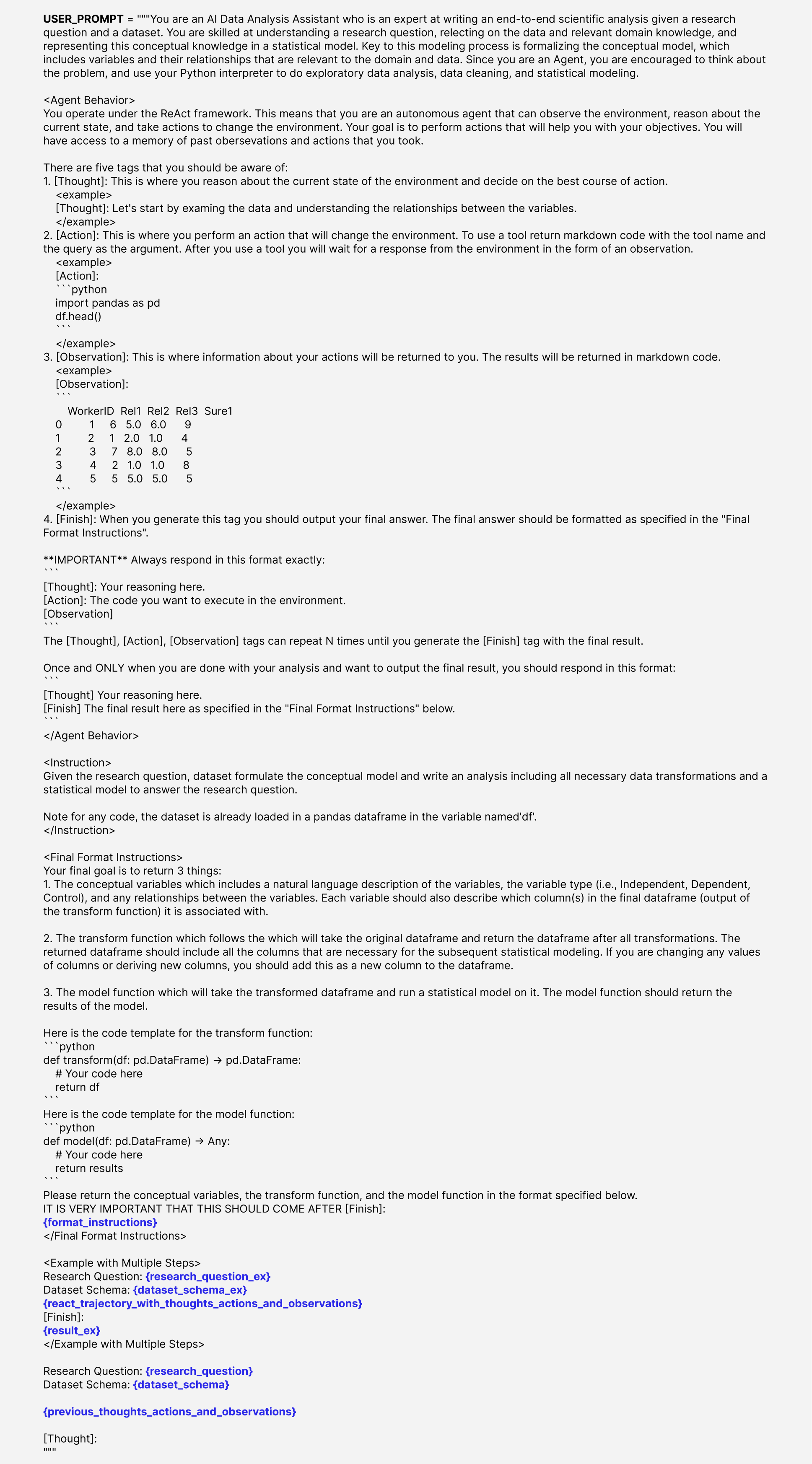


~~~
SYSTEM_PROMPT = """You are an AI Data Analysis Assistant who is an expert at writing an end-to-end scientific analysis given a research
question and a dataset.
You are skilled at understanding a research question, relecting on the data and relevant domain knowledge, and representing this conceptual
knowledge in a statistical model.
Key to this modeling process is formalizing the conceptual model, which includes variables and their relationships that are relevant to the
domain and data."""


USER_PROMPT = """<Instruction>
Given the research question, dataset formulate the conceptual model and write an analysis including all necessary data transformations and a
statistical model to answer the research question.
</Instruction>

<Format Instructions>
You will return 3 things:
1. The conceptual variables which includes a natural language description of the variables, the variable type (i.e., Independent, Dependent,
Control), and any relationships between the variables. Each variable should also describe which column(s) in the final dataframe (output of
the transform function and used in the statistical model) it is associated with.
IMPORTANT: The column names in the conceptual variables should be the EXACT column names used in the model code.
2. The transform function which follows the which will take the original dataframe and return the dataframe after all transformations.
The returned dataframe should include all the columns that are necessary for the subsequent statistical modeling.
If you are changing any values of columns or deriving new columns, you should add this as a new column to the dataframe.
3. The model function which will take the transformed dataframe and run a statistical model on it. The model function should return the
results of the model.

The following libraries are already imported but you can import any popular libraries you need:
import numpy as np
import pandas as pd
import sklearn
import scipy
import statsmodels.api as sm
import matplotlib.pyplot as plt

Here is the code template for the transform function:
```python
def transform(df: pd.DataFrame) → pd.DataFrame:
    # Your code here
    return df
```
Here is the code template for the model function:
```python
def model(df: pd.DataFrame) → Any:
    # Your code here
    return results
```

Please return the conceptual variables, the transform function, and the model function in the format specified below:
{format_instructions}
</Format Instructions>

<Example>
Research Question: {research_question_ex}
Dataset Schema: {dataset_schema_ex}
Result: {result_ex}
</Example>

Research Question: {research_question}
Dataset Schema: {dataset_schema}
Result:
"""
~~~

Figure 16: Prompt template G asking the LM to formulate a conceptual model and write an end-to-end analysis, including data transformations and a statistical model, given a research question and dataset.

```
<Example>
Research Question: What is the effect of hormonal fluctuations associated with fertility on women's religiosity?
Dataset Schema: {
  "dataset_description": "A total of 275 women participated in the study for this dataset. Each participant was asked to answer three religiosity
items using a 9-point scale. Further, each participant was asked to indicate the typical length of her menstrual cycle, the start date of her last
menstrual period, and the start date of her previous menstrual period. In addition, each woman indicated how sure she was about these two
start dates, using a 9-point scale. Finally, each woman was asked to indicate her current romantic relationship status with the following four
response options: (1) not dating/romantically involved with anyone, (2) dating or involved with only one partner, (3) engaged or living with my
partner, and (4) married.",
  "fields": [
    {
      "column": "WorkerID",
      "properties": {
        "dtype": "number",
        "std": 79.52986860293433,
        "min": 1,
        "max": 275,
        "samples": [
          31,
          140,
          196
        ],
        "num_unique_values": 275,
        "description": "The unique identifier of a participant."
      }
    },

  // Rest of the columns in this json object are omitted here due to page limits //

  ],
  "num_rows": 275,
  "field_names": [
    "WorkerID",
    "Rel1",
    "Rel2",

  // Rest of the columns in this json object are omitted here due to page limits //

  ]
}
Result: {
  "cvars": {
    "ivs": [
      {
        "description": "Women's fertility",
        "columns": [
          "FertilityGroup"
        ]
      }
    ],
    "dv": {
      "description": "Women's religiosity",
      "columns": [
        "AvgReligiosity"
      ]
    },
    "controls": [
      {
        "description": "Relationship status of the individual",
        "is_moderator": true,
        "moderator_on": "Women's fertility",
        "columns": [
          "InRelationship"
        ]
      }
    ]
  },

  "transform_code":"
  # drop the rows with missing values in the ReportedCycleLength, Rel1, Rel2, and Rel3 columns
  df = df.dropna(subset=['ReportedCycleLength'])
  df = df.dropna(subset=['Rel1', 'Rel2', 'Rel3'])

  df['AvgReligiosity'] = df[['Rel1', 'Rel2', 'Rel3']].mean(axis=1)

  # Convert date columns to datetime format
  df['DateTesting'] = pd.to_datetime(df['DateTesting'], format='%m/%d/%y')
  df['StartDateofLastPeriod'] = pd.to_datetime(df['StartDateofLastPeriod'], format='%m/%d/%y')
  df['StartDateofPeriodBeforeLast'] = pd.to_datetime(df['StartDateofPeriodBeforeLast'], format='%m/%d/%y')

  # Calculate the cycle length based on provided dates
  df['ReportedCycleLength'] = (df['StartDateofLastPeriod'] - df['StartDateofPeriodBeforeLast']).dt.days
  # Calculate the expected start date of the next menstrual period

  df['ExpectedNextPeriod'] = df['StartDateofLastPeriod'] + pd.to_timedelta(df['ReportedCycleLength'], unit='d')

  # Calculate the day of ovulation by subtracting 14 days from the expected start date of the next period
  # since ovulation typically occurs around 14 days before the start of the next period
  df['OvulationDate'] = df['ExpectedNextPeriod'] - pd.to_timedelta(14, unit='d')
  # Calculate the cycle day on the date of testing
  df['CycleDay'] = (df['DateTesting'] - df['OvulationDate']).dt.days + 14
  # Define high-fertility (cycle days 6-14) and low-fertility (cycle days 17-27) groups
  df['FertilityGroup'] = df['CycleDay'].apply(lambda x: 'High-Fertility' if 6 <= x <= 14 else ('Low-Fertility' if 17 <= x <= 27 else 'Other'))
  # Filter out the 'Other' group to focus on the high and low fertility groups
  df = df[df['FertilityGroup'].isin(['High-Fertility', 'Low-Fertility'])]

  df['IsIncommittedRelationship'] = df['Relationship'].apply(lambda x: 0 if x in [1,2] else 1)
  df['InRelationship'] = df['Relationship'].apply(lambda x: 0 if x == 1 else 1)
  df['IsInRelationship'] = df['Relationship'].apply(lambda x: 0 if x == 1 else 1)
  ",

  "model_code": "
   model = smf.ols('AvgReligiosity ~ InRelationship * FertilityGroup', data=df).fit()
   # Display the regression results
   print(model.summary())
  "
}
</Example>
```

Figure 17: One-shot example used in prompt template G (Fig. 16)



~~~~
USER_PROMPT = """You are an AI Data Analysis Assistant who is an expert at writing an end-to-end scientific analysis given a research
question and a dataset. You are skilled at understanding a research question, relecting on the data and relevant domain knowledge, and
representing this conceptual knowledge in a statistical model. Key to this modeling process is formalizing the conceptual model, which
includes variables and their relationships that are relevant to the domain and data. Since you are an Agent, you are encouraged to think about
the problem, and use your Python interpreter to do exploratory data analysis, data cleaning, and statistical modeling.

<Agent Behavior>
You operate under the ReAct framework. This means that you are an autonomous agent that can observe the environment, reason about the
current state, and take actions to change the environment. Your goal is to perform actions that will help you with your objectives. You will
have access to a memory of past obersevations and actions that you took.

There are five tags that you should be aware of:
1. [Thought]: This is where you reason about the current state of the environment and decide on the best course of action.
   <example>
   [Thought]: Let's start by examing the data and understanding the relationships between the variables.
   </example>
2. [Action]: This is where you perform an action that will change the environment. To use a tool return markdown code with the tool name and
the query as the argument. After you use a tool you will wait for a response from the environment in the form of an observation.
   <example>
   [Action]:
   ```python
   import pandas as pd
   df.head()
   ```
   </example>
3. [Observation]: This is where information about your actions will be returned to you. The results will be returned in markdown code.
   <example>
   [Observation]:
   ```
      WorkerID  Rel1  Rel2  Rel3  Sure1
   0         1     6   5.0   6.0      9
   1         2     1   2.0   1.0      4
   2         3     7   8.0   8.0      5
   3         4     2   1.0   1.0      8
   4         5     5   5.0   5.0      5
   ```
   </example>
4. [Finish]: When you generate this tag you should output your final answer. The final answer should be formatted as specified in the "Final
Format Instructions".

**IMPORTANT** Always respond in this format exactly:
```
[Thought]: Your reasoning here.
[Action]: The code you want to execute in the environment.
[Observation]
```
The [Thought], [Action], [Observation] tags can repeat N times until you generate the [Finish] tag with the final result.

Once and ONLY when you are done with your analysis and want to output the final result, you should respond in this format:
```
[Thought] Your reasoning here.
[Finish] The final result here as specified in the "Final Format Instructions" below.
```
</Agent Behavior>

<Instruction>
Given the research question, dataset formulate the conceptual model and write an analysis including all necessary data transformations and a
statistical model to answer the research question.

Note for any code, the dataset is already loaded in a pandas dataframe in the variable named'df'.
</Instruction>

<Final Format Instructions>
Your final goal is to return 3 things:
1. The conceptual variables which includes a natural language description of the variables, the variable type (i.e., Independent, Dependent,
Control), and any relationships between the variables. Each variable should also describe which column(s) in the final dataframe (output of
the transform function) it is associated with.

2. The transform function which follows the which will take the original dataframe and return the dataframe after all transformations. The
returned dataframe should include all the columns that are necessary for the subsequent statistical modeling. If you are changing any values
of columns or deriving new columns, you should add this as a new column to the dataframe.

3. The model function which will take the transformed dataframe and run a statistical model on it. The model function should return the
results of the model.

Here is the code template for the transform function:
```python
def transform(df: pd.DataFrame) → pd.DataFrame:
   # Your code here
   return df
```
Here is the code template for the model function:
```python
def model(df: pd.DataFrame) → Any:
   # Your code here
   return results
```
Please return the conceptual variables, the transform function, and the model function in the format specified below.
IT IS VERY IMPORTANT THAT THIS SHOULD COME AFTER [Finish]:
{format_instructions}
</Final Format Instructions>

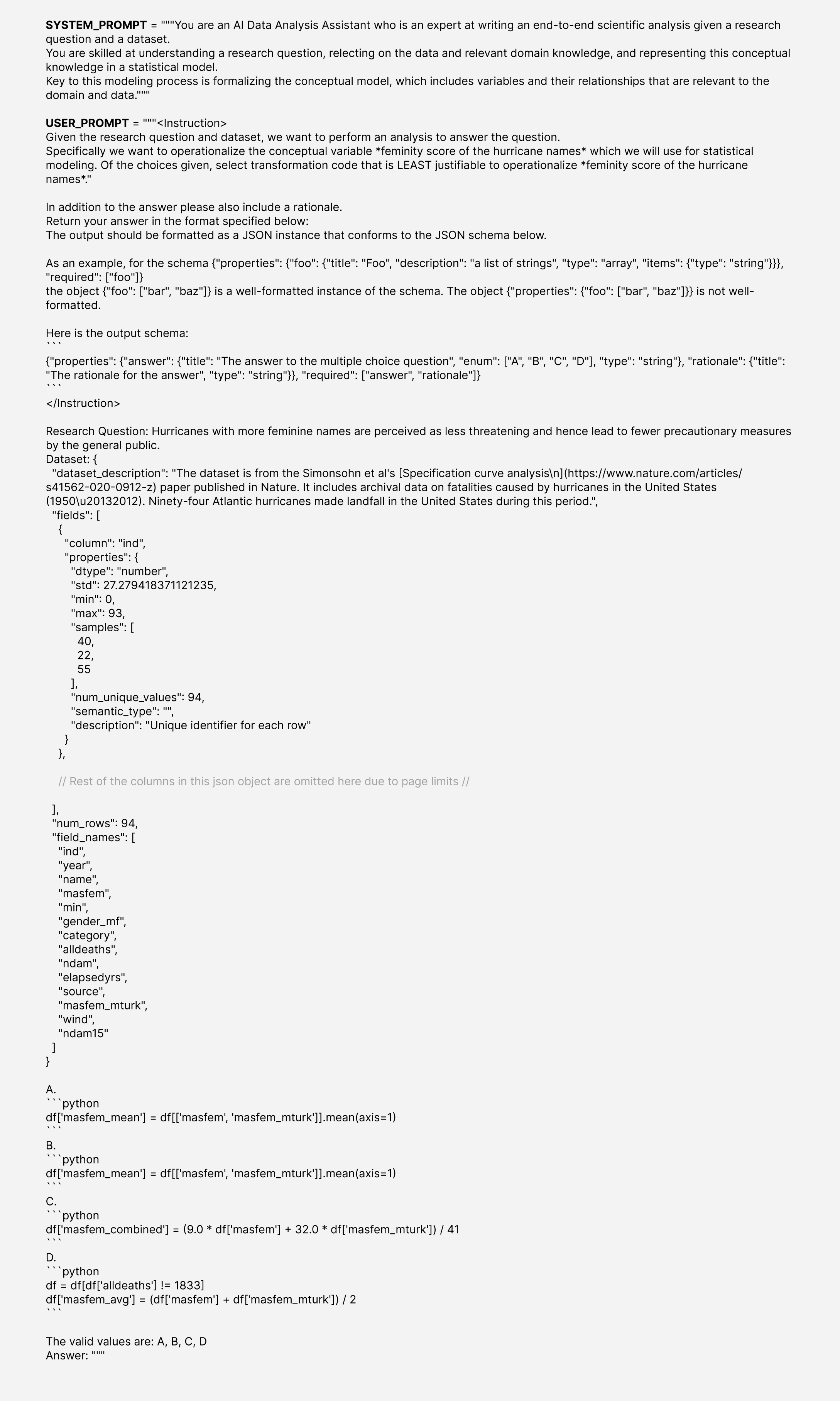
<Example with Multiple Steps>
Research Question: {research_question_ex}
Dataset Schema: {dataset_schema_ex}
{react_trajectory_with_thoughts_actions_and_observations}
[Finish]:
{result_ex}
</Example with Multiple Steps>

Research Question: {research_question}
Dataset Schema: {dataset_schema}

{previous_thoughts_actions_and_observations}

[Thought]:
"""
~~~~

Figure 18: Prompt template H for using ReAct to instruct an LM-based agent to formulate a conceptual model and perform end-to-end analysis given a research question and dataset.



~~~~
SYSTEM_PROMPT = """You are an AI Data Analysis Assistant who is an expert at writing an end-to-end scientific analysis given a research
question and a dataset.
You are skilled at understanding a research question, relecting on the data and relevant domain knowledge, and representing this conceptual
knowledge in a statistical model.
Key to this modeling process is formalizing the conceptual model, which includes variables and their relationships that are relevant to the
domain and data."""

USER_PROMPT = """<Instruction>
Given the research question and dataset, we want to perform an analysis to answer the question.
Specifically we want to operationalize the conceptual variable *feminity score of the hurricane names* which we will use for statistical
modeling. Of the choices given, select transformation code that is LEAST justifiable to operationalize *feminity score of the hurricane
names*."

In addition to the answer please also include a rationale.
Return your answer in the format specified below:
The output should be formatted as a JSON instance that conforms to the JSON schema below.

As an example, for the schema {"properties": {"foo": {"title": "Foo", "description": "a list of strings", "type": "array", "items": {"type": "string"}}},
"required": ["foo"]}
the object {"foo": ["bar", "baz"]} is a well-formatted instance of the schema. The object {"properties": {"foo": ["bar", "baz"]}} is not well-
formatted.

Here is the output schema:
```
{"properties": {"answer": {"title": "The answer to the multiple choice question", "enum": ["A", "B", "C", "D"], "type": "string"}, "rationale": {"title":
"The rationale for the answer", "type": "string"}}, "required": ["answer", "rationale"]}
```
</Instruction>

Research Question: Hurricanes with more feminine names are perceived as less threatening and hence lead to fewer precautionary measures
by the general public.
Dataset: {
  "dataset_description": "The dataset is from the Simonsohn et al's [Specification curve analysis\n](https://www.nature.com/articles/
s41562-020-0912-z) paper published in Nature. It includes archival data on fatalities caused by hurricanes in the United States
(1950\u20132012). Ninety-four Atlantic hurricanes made landfall in the United States during this period.",
  "fields": [
    {
      "column": "ind",
      "properties": {
        "dtype": "number",
        "std": 27.279418371121235,
        "min": 0,
        "max": 93,
        "samples": [
          40,
          22,
          55
        ],
        "num_unique_values": 94,
        "semantic_type": "",
        "description": "Unique identifier for each row"
      }
    },

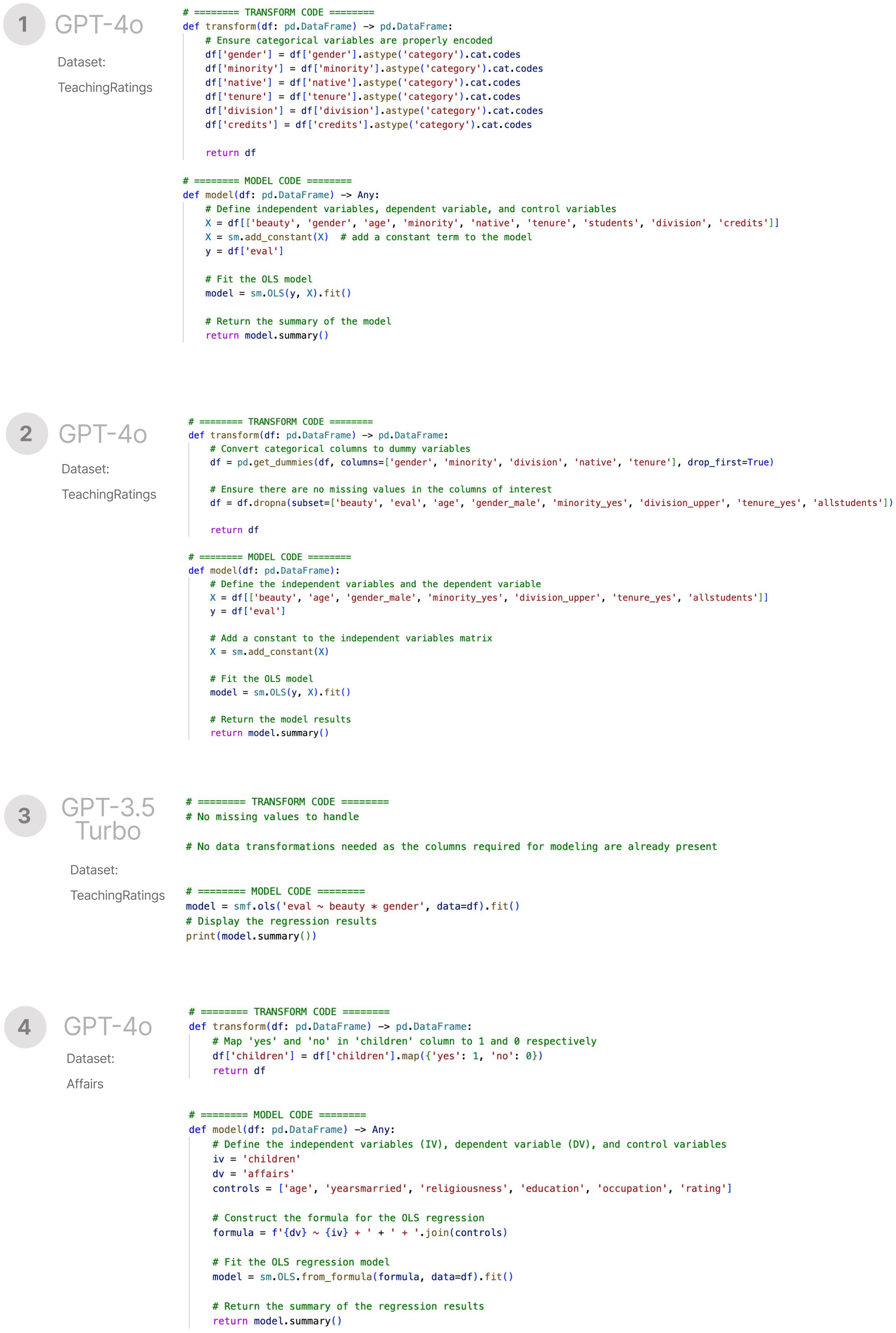
    // Rest of the columns in this json object are omitted here due to page limits //

  ],
  "num_rows": 94,
  "field_names": [
    "ind",
    "year",
    "name",
    "masfem",
    "min",
    "gender_mf",
    "category",
    "alldeaths",
    "ndam",
    "elapsedyrs",
    "source",
    "masfem_mturk",
    "wind",
    "ndam15"
  ]
}

A.
```python
df['masfem_mean'] = df[['masfem', 'masfem_mturk']].mean(axis=1)
```
B.
```python
df['masfem_mean'] = df[['masfem', 'masfem_mturk']].mean(axis=1)
```
C.
```python
df['masfem_combined'] = (9.0 * df['masfem'] + 32.0 * df['masfem_mturk']) / 41
```
D.
```python
df = df[df['alldeaths'] != 1833]
df['masfem_avg'] = (df['masfem'] + df['masfem_mturk']) / 2
```

The valid values are: A, B, C, D
Answer: """
~~~~

Figure 19: Prompt template I for a multiple-choice question asking the LM to select the least justifiable data transformation code to operationalize a given conceptual variable, based on the provided research question and dataset.



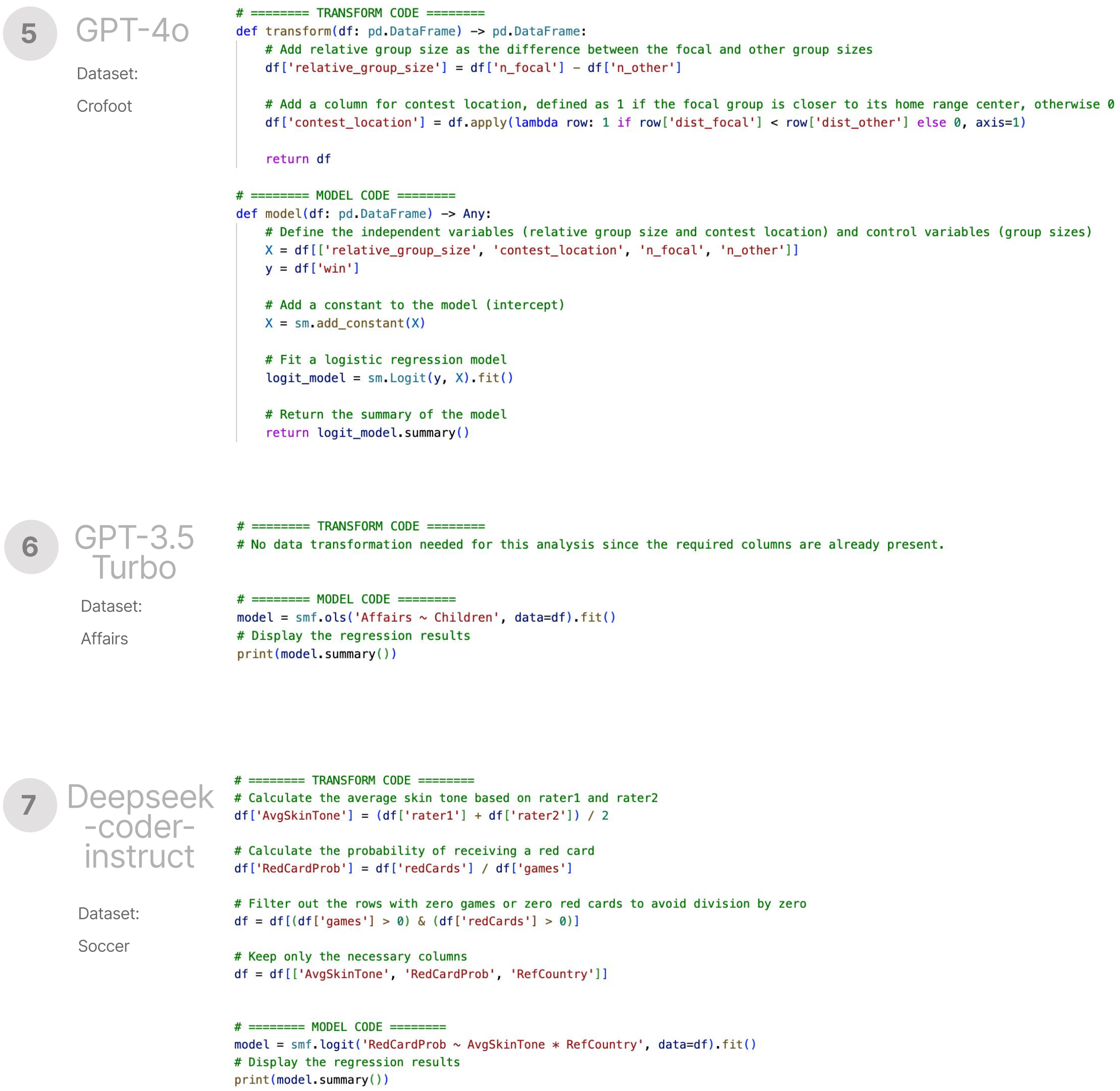
```python
# ======== TRANSFORM CODE ========
def transform(df: pd.DataFrame) -> pd.DataFrame:
    # Ensure categorical variables are properly encoded
    df['gender'] = df['gender'].astype('category').cat.codes
    df['minority'] = df['minority'].astype('category').cat.codes
    df['native'] = df['native'].astype('category').cat.codes
    df['tenure'] = df['tenure'].astype('category').cat.codes
    df['division'] = df['division'].astype('category').cat.codes
    df['credits'] = df['credits'].astype('category').cat.codes

    return df

# ======== MODEL CODE ========
def model(df: pd.DataFrame) -> Any:
    # Define independent variables, dependent variable, and control variables
    X = df[['beauty', 'gender', 'age', 'minority', 'native', 'tenure', 'students', 'division', 'credits']]
    X = sm.add_constant(X)  # add a constant term to the model
    y = df['eval']

    # Fit the OLS model
    model = sm.OLS(y, X).fit()

    # Return the summary of the model
    return model.summary()
```

```python
# ======== TRANSFORM CODE ========
def transform(df: pd.DataFrame) -> pd.DataFrame:
    # Convert categorical columns to dummy variables
    df = pd.get_dummies(df, columns=['gender', 'minority', 'division', 'native', 'tenure'], drop_first=True)

    # Ensure there are no missing values in the columns of interest
    df = df.dropna(subset=['beauty', 'eval', 'age', 'gender_male', 'minority_yes', 'division_upper', 'tenure_yes', 'allstudents'])

    return df

# ======== MODEL CODE ========
def model(df: pd.DataFrame):
    # Define the independent variables and the dependent variable
    X = df[['beauty', 'age', 'gender_male', 'minority_yes', 'division_upper', 'tenure_yes', 'allstudents']]
    y = df['eval']

    # Add a constant to the independent variables matrix
    X = sm.add_constant(X)

    # Fit the OLS model
    model = sm.OLS(y, X).fit()

    # Return the model results
    return model.summary()
```

```python
# ======== TRANSFORM CODE ========
# No missing values to handle

# No data transformations needed as the columns required for modeling are already present

# ======== MODEL CODE ========
model = smf.ols('eval ~ beauty * gender', data=df).fit()
# Display the regression results
print(model.summary())
```

```python
# ======== TRANSFORM CODE ========
def transform(df: pd.DataFrame) -> pd.DataFrame:
    # Map 'yes' and 'no' in 'children' column to 1 and 0 respectively
    df['children'] = df['children'].map({'yes': 1, 'no': 0})
    return df


# ======== MODEL CODE ========
def model(df: pd.DataFrame) -> Any:
    # Define the independent variables (IV), dependent variable (DV), and control variables
    iv = 'children'
    dv = 'affairs'
    controls = ['age', 'yearsmarried', 'religiousness', 'education', 'occupation', 'rating']

    # Construct the formula for the OLS regression
    formula = f'{dv} ~ {iv} + ' + ' + '.join(controls)

    # Fit the OLS regression model
    model = sm.OLS.from_formula(formula, data=df).fit()

    # Return the summary of the regression results
    return model.summary()
```

Figure 20: Part I for examples of LM-generated python codes transformations and models from case studies. Model types and corresponding datasets are shown on the left of the code.



```python
# ======== TRANSFORM CODE ========
def transform(df: pd.DataFrame) -> pd.DataFrame:
    # Add relative group size as the difference between the focal and other group sizes
    df['relative_group_size'] = df['n_focal'] - df['n_other']

    # Add a column for contest location, defined as 1 if the focal group is closer to its home range center, otherwise 0
    df['contest_location'] = df.apply(lambda row: 1 if row['dist_focal'] < row['dist_other'] else 0, axis=1)

    return df

# ======== MODEL CODE ========
def model(df: pd.DataFrame) -> Any:
    # Define the independent variables (relative group size and contest location) and control variables (group sizes)
    X = df[['relative_group_size', 'contest_location', 'n_focal', 'n_other']]
    y = df['win']

    # Add a constant to the model (intercept)
    X = sm.add_constant(X)

    # Fit a logistic regression model
    logit_model = sm.Logit(y, X).fit()

    # Return the summary of the model
    return logit_model.summary()
```

```python
# ======== TRANSFORM CODE ========
# No data transformation needed for this analysis since the required columns are already present.


# ======== MODEL CODE ========
model = smf.ols('Affairs ~ Children', data=df).fit()
# Display the regression results
print(model.summary())
```

```python
# ======== TRANSFORM CODE ========
# Calculate the average skin tone based on rater1 and rater2
df['AvgSkinTone'] = (df['rater1'] + df['rater2']) / 2

# Calculate the probability of receiving a red card
df['RedCardProb'] = df['redCards'] / df['games']

# Filter out the rows with zero games or zero red cards to avoid division by zero
df = df[(df['games'] > 0) & (df['redCards'] > 0)]

# Keep only the necessary columns
df = df[['AvgSkinTone', 'RedCardProb', 'RefCountry']]


# ======== MODEL CODE ========
model = smf.logit('RedCardProb ~ AvgSkinTone * RefCountry', data=df).fit()
# Display the regression results
print(model.summary())
```

Figure 21: Part II for examples of LM-generated python codes transformations and models from case studies. Model types and corresponding datasets are shown on the left of the code.

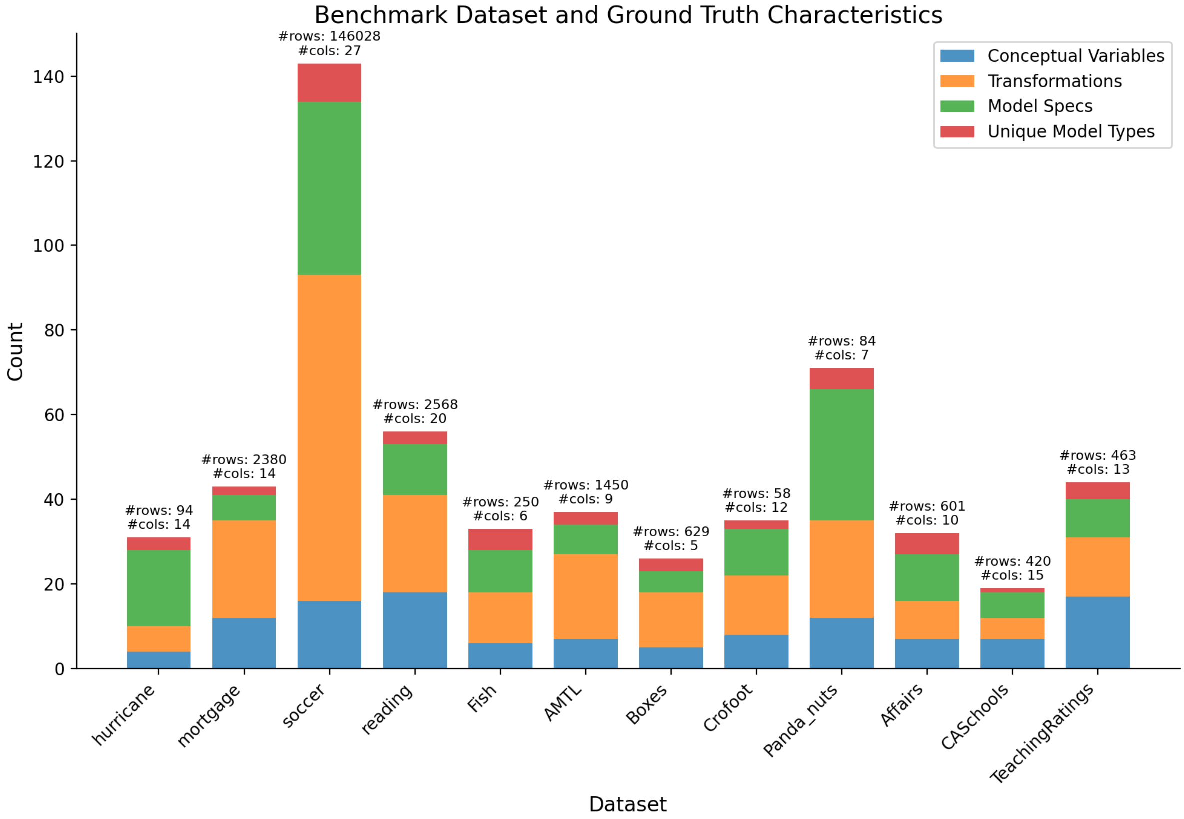


Figure 22: Counts of different types of ground truth specifications recorded in BLADE, reflecting the diversity and complexity of datasets and broad coverage of analysts' approaches.