

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/380523618>

Leveraging Large Language Models to Enhance Domain Expert Inclusion in Data Science Workflows

Conference Paper · May 2024

DOI: 10.1145/3613905.3651115

CITATION

1

READS

12

4 authors, including:



[Hari Subramonyam](#)

Stanford University

56 PUBLICATIONS 764 CITATIONS

SEE PROFILE



Leveraging Large Language Models to Enhance Domain Expert Inclusion in Data Science Workflows

Jasmine Y. Shih
Stanford University
USA
jyshih@stanford.edu

Yannis Katsis
IBM Research
USA
yannis.katsis@ibm.com

Vishal Mohanty
Stanford University
USA
vmohanty@stanford.edu

Hari Subramonyam
Stanford University
USA
harihars@stanford.edu

ABSTRACT

Domain experts can play a crucial role in guiding data scientists to optimize machine learning models while ensuring contextual relevance for downstream use. However, in current workflows, such collaboration is challenging due to differing expertise, abstract documentation practices, and lack of access and visibility into low-level implementation artifacts. To address these challenges and enable domain expert participation, we introduce CellSync, a collaboration framework comprising (1) a Jupyter Notebook extension that continuously tracks changes to dataframes and model metrics and (2) a Large Language Model powered visualization dashboard that makes those changes interpretable to domain experts. Through CellSync's cell-level dataset visualization with code summaries, domain experts can interactively examine how individual data and modeling operations impact different data segments. The chat features enable data-centric conversations and targeted feedback to data scientists. Our preliminary evaluation shows that CellSync provides transparency and promotes critical discussions about the intents and implications of data operations.

CCS CONCEPTS

• **Human-centered computing** → **User interface programming**; *Collaborative and social computing systems and tools*; *Information visualization*.

KEYWORDS

Collaborative ML, prompt engineering, tabular datasets, computational notebooks, data subset visualization

ACM Reference Format:

Jasmine Y. Shih, Vishal Mohanty, Yannis Katsis, and Hari Subramonyam. 2024. Leveraging Large Language Models to Enhance Domain Expert Inclusion in Data Science Workflows. In *Extended Abstracts of the CHI Conference on Human Factors in Computing Systems (CHI EA '24)*, May 11–16, 2024, Honolulu, HI, USA. ACM, New York, NY, USA, 11 pages. <https://doi.org/10.1145/3613905.3651115>

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
CHI EA '24, May 11–16, 2024, Honolulu, HI, USA
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0331-7/24/05
<https://doi.org/10.1145/3613905.3651115>

1 INTRODUCTION

When building machine learning (ML) models, collaborating with domain experts such as educators, public health experts, agricultural specialists, etc. is essential to ensure the accuracy, fairness, and contextual applicability of ML models [46]. For instance, the domain expert might provide knowledge about specific data patterns useful for missing value imputations; based on current model performance, they may suggest creating new features such as calculating blood sugar levels over a 3-month period. Domain experts play a critical role in ensuring ML solutions are tailored to the broader domain goals, including fairness, transparency, and accountability [33, 53].

Given the dynamic and iterative nature of ML modeling processes, cross-discipline collaboration should be *continuous* through constant evaluation and feedback [20, 44]. However, current support for domain expert involvement is largely limited to abstract summarizations of modeling decisions in the form of documentation. Artifacts such as Datasheets [14], Factsheets [3], and Model Cards [28] are only created towards the end of the development process [16], making it difficult for domain experts to actively participate in upstream modeling tasks. Furthermore, domain experts not proficient in data-driven work may face challenges in accessing and interpreting computational artifacts such as code [55]. This knowledge gap necessitates extra work from engineering teams to translate complex decisions into formats accessible to domain experts. However, a lack of shared understanding of each other's tasks and terminology can lead to misunderstandings, inconsistent assumptions, and even conflicts [31].

To address current challenges in collaborative ML, we explore the potential of large language models (LLMs) to enhance transparency and facilitate the involvement of domain experts in data science workflows. We developed CellSync, which consists of a visualization dashboard powered by LLMs that provides domain experts with an 'alternate' view of computational notebooks commonly utilized by data science practitioners. Complementing the visualization interface is a Jupyter Notebook [22] extension that parses and relays computational operations to the dashboard. As the data scientist performs each data operation in Jupyter Notebook, a data version card containing a text summary and a grid-based visualization (*SnapGrid*) is added to the dashboard to explain the data operation. CellSync leverages LLM one-shot, few-shot, and chain-of-thought prompting [1, 27, 52] to generate text summaries of Python code and extract information for visualization purposes. Additionally,

CellSync supports a chat feature between the dashboard and the notebook for domain experts to directly share feedback with data scientists. In the current instantiation of our work, we focused on tabular data given their widespread prevalence in predictive modeling tasks. To evaluate how CellSync supports domain experts' understanding of ML data operations, we conducted a controlled study with 10 pairs of domain experts and data scientists. Our work contributes 1) an LLM-powered visualization dashboard that makes it possible for domain experts to follow along with data scientists' work in Jupyter Notebook and 2) a preliminary evaluation of the dashboard features.

2 RELATED WORK






In collaborative ML, researchers have investigated how data science and ML practitioners may work with domain experts in different stages of the ML pipeline [5, 26, 30, 31, 38, 40, 43, 49]. Current collaboration approaches are largely through live meetings, messaging channels, and, in a few instances, specially designed graphical interfaces [5, 8, 10, 25, 37, 40]. Collaborative ML teams engage in a mix of synchronous and asynchronous communication tools depending on whether the current discussion requires content-based (e.g. Jupyter Notebook and Git version control systems) or process-based communication (e.g. emails and Slack) [26, 42]. To ensure effective collaboration, data scientists and domain experts must continue to establish content common ground (shared understanding of work subject) and process common ground (shared understanding of procedures) throughout the lifecycle of ML tasks [26]. However, the lack of centralized tools and protocols makes it difficult for multidisciplinary teams to sustain collaboration across ML stages. As a result, the interactions between data scientists and domain experts are often limited to one stage of the ML pipeline [30, 43].

Differences in expertise between domain experts and data scientists pose additional challenges. Data scientists often spend large amounts of time preparing concrete data examples to communicate with domain experts [37], and meaningful ML stakeholder participation requires scaffolds to aid the exchange of technical and non-technical considerations [45]. To bridge the expertise gap, systems should be designed to aid domain experts in understanding data science techniques in a manner that better matches their mental model of data [21]. Visualizations of data changes [17, 24, 32] have proved to be an effective means of supporting understanding beyond serving the function of documentation. Recent work in Explainable AI [7, 9, 11, 18, 50] has also demonstrated ways to make ML components more interpretable to non-experts through visualization and direct manipulation interactions [2, 29, 39].


Informed by the challenges and potential solutions identified in prior research, we explore the use of LLMs to supplement human effort in translating code into natural language and explainable visualizations. In CellSync, we build on the Smallset Timelines visualization technique [24] to select and visualize a digestible subset of rows and columns for domain experts. Further, we utilize LLMs' code summarization capabilities [1] and refine these summaries through targeted prompt engineering to tailor them towards domain experts. To better support data scientists in their collaboration with domain experts, we looked to prior works to design the data-scientist-facing component of CellSync. Inspired by the Callisto


system [48] that enables data scientists to work together in computational notebooks with context-rich chat features, as well as studies that reveal collaboration patterns among data scientists [41, 47], we developed a Jupyter Notebook extension that is situated within data scientists' ML work environment and provides a chat interface for data-centric communication with domain experts. In our work, we focus specifically on tabular data due to its high availability in health and education domains. We selected predictive model development as the ML task to design for because predictive models have high implications in our chosen domains [4, 12, 15, 19, 34].

3 USER EXPERIENCE

CellSync consists of a Jupyter Notebook extension and a web visualization dashboard. The extension tracks data scientists' data operations in each notebook cell. The visualization dashboard, designed for domain experts, communicates each change to a dataframe with a text summary and a grid-based visualization (*SnapGrid*) on a data version card (Figure 1a). Both the extension and the dashboard provide a chat feature for domain experts (Figure 1d) and data scientists (Figure 2) to exchange comments. Each SnapGrid, inspired by Smallset Timelines [24] which visualizes data changes in grid-based snapshots, shows a static 9-by-9 subset of the data; the subset is selected to maximize the coverage of the changes tied to the data operation. On each square  that represents a cell in the dataframe, the cell's previous and new values are displayed with an arrow between them. Cells, rows, or columns that have been modified, added, or removed are highlighted with different colors – blue  for modification of cells, purple  for new columns and boxes  around related columns, and red  for removed columns or rows. A navigator on the left of the page provides a compact view of the data operation history with clickable circles for navigation (Figure 1b).

To illustrate CellSync's features and user experience, we present a fictional journey of Sandy, a data scientist, and Ellis, an education domain expert, who work together on an ML model to predict students' writing scores.

Set-up: Sandy obtains a dataset with 600 rows of student background information and exam scores. She loads the dataset into a *pandas dataframe* variable named *df* by executing a Jupyter Notebook cell. In the web dashboard, Ellis sees a text summary about the operation along with a SnapGrid showing a subset of the loaded data in the corresponding data version card. He clicks on the thumbnail histogram  under each column name on the card to open the histogram and column statistics in the side panel (Figure 1c).

Missing Value Analysis: Sandy fills in the missing values in the "EthnicGroup" column, which contains categorical values, with the most frequent category. In the web dashboard, Ellis attempts to understand the newly executed operation by reading the code summary and viewing the SnapGrid. As more than 9 values have been modified, all 9 cells in the column are colored in blue , and a number above the first cell in the column indicates the total number of values changed. Concerned about the implications of assuming missing data, Ellis sends a comment to Sandy to suggest she replace the missing values with "Unknown" instead.

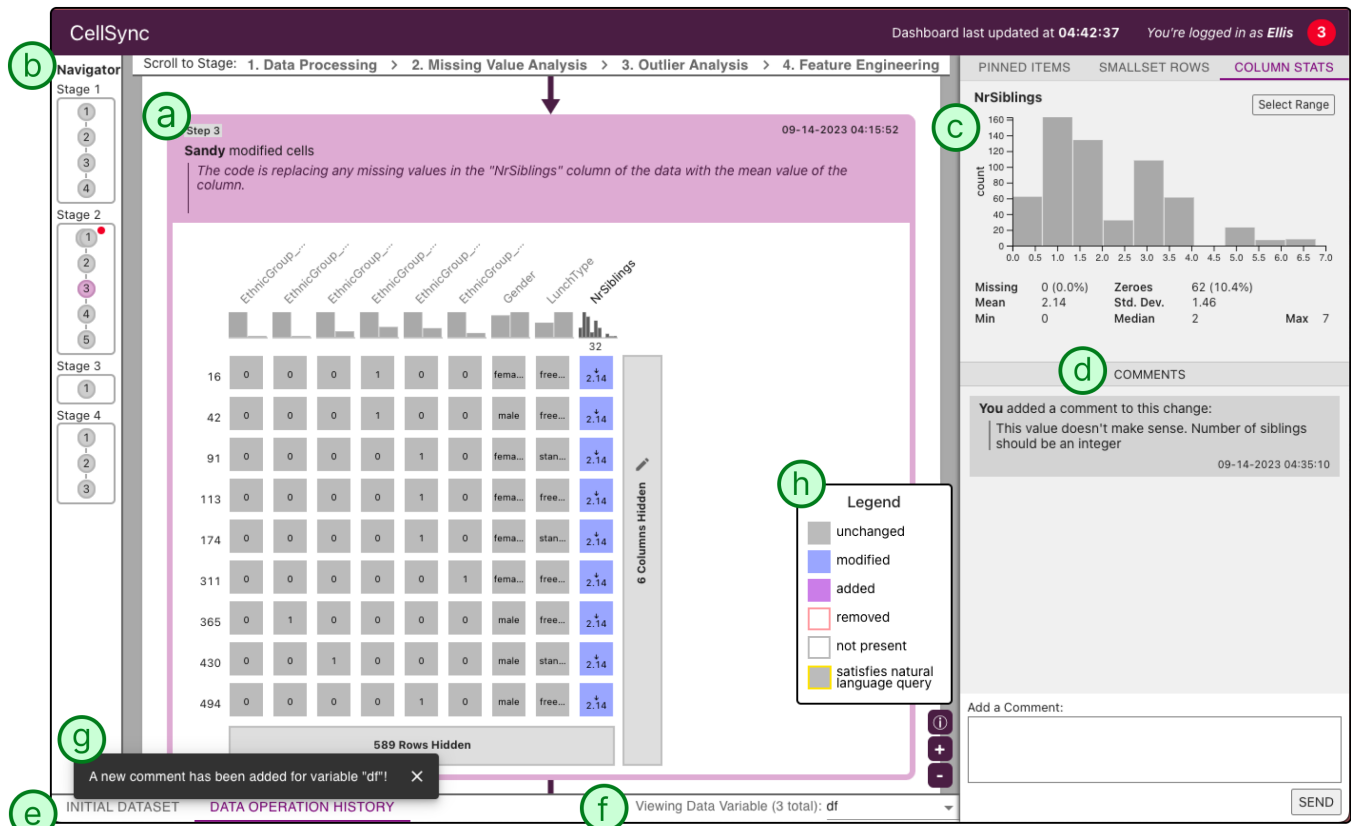


Figure 1: The CellSync visualization dashboard interface: (a) data version card containing code summary and SnapGrid along with column histograms, (b) clickable card navigator displaying a red dot indicating new comments, (c) detailed view for the selected column, (d) comments section for the current card, (e) bottom bar for switching between initial dataset table and data operation history, (f) dropdown menu for variable selection, (g) notification for new comments, and (h) collapsible legend for SnapGrid.

Feature Encoding: Sandy performs one-hot encoding on a categorical column, thereby creating a new column for each category and removing the original column from the dataframe. In the corresponding SnapGrid, the cells in the new columns are colored in purple , and the cells in the removed column are colored in white with a red border . In addition, a light purple box is placed around the removed column to indicate that the new columns have been created based on values in this original column. Ellis views the data version card for the one-hot encoding operation and understands that the encoding operation transforms the original categorical values into binary values.

Feature Selection: Sandy observes that the “SportsPracticeFrequency” column does not seem to be correlated with “WritingScore” and decides to remove it. Ellis inquires Sandy about the deletion of data. Sandy explains that the column is not correlated with “WritingScore” and therefore is not useful for the predictive model. To filter the raw data for his own examination, Ellis uses the natural language query feature in the side panel (Figure 3a). He clicks on the previous data version card where the “SportsPracticeFrequency” column is still present and submits the query “WritingScore is below

75 and SportsPracticeFrequency is less than 2”. The updated SnapGrid highlights values that meet the criteria with a yellow border (Figure 3b). This feature enables Ellis to examine the data in a flexible and natural manner.

Model Training: Sandy splits the dataset into a training set stored as variable X_{train} and a testing set stored as X_{test} . The new dataframe variables lead to the addition of two new data version cards in the visualization dashboard. To access a new card, Ellis clicks on the variables drop-down menu in the bottom bar and selects the X_{train} variable to switch the data operation history view to the new variable. Sandy then trains a “LinearRegression” model with the X_{train} variable and evaluates the model by calculating model performance metrics such as mean squared error, mean absolute error, and R2 score. In the visualization dashboard, Ellis is able to view the model name and metrics in the X_{train} data version card (Figure 3c).

4 SYSTEM IMPLEMENTATION

CellSync consists of a Jupyter Notebook extension and web visualization dashboard. Both components of the system are synced

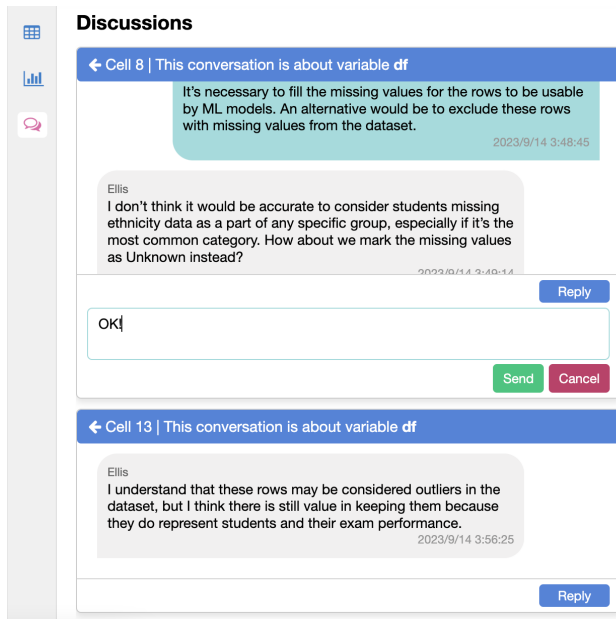


Figure 2: The CellSync data-scientist-facing chat interface rendered by the Jupyter Notebook extension.

with the shared Amazon DynamoDB database every 15 seconds using the serverless Amazon Lambda functions as the backend. The extension is also linked to OpenAI’s *text-davinci-003* LLM model [35] to support several features in the backend.

4.1 Jupyter Notebook Extension

Our extension is built on the Jupyter Nbextensions framework¹ and provides the core functionality to automatically share the data operation and model training history with the visualization dashboard. To capture the data operations in the ML model development process, we utilize a Python script that is invoked each time a code cell is executed in the notebook to obtain variable contents. Our script extends the script used in the open-sourced Variable Inspector extension², which lists all the variables in the context of the notebook using NamespaceMagics³, to compare the current and prior states of dataframe variables. If a dataframe variable has been modified, we obtain an LLM-generated summary of the code (described in 4.1.1) and compute a subset of the dataframe to be displayed in the visualization dashboard (described in 4.1.2). This information is packaged in a JSON object along with contextual data about the notebook cell. The JSON object is then sent to the Amazon DynamoDB database to allow information to be shared between the notebook and the visualization dashboard. In addition, each version of the full dataset is stored in our Amazon S3 database as a CSV file.

¹https://github.com/ipython-contrib/jupyter_contrib_nbextensions

²<https://jupyter-contrib-nbextensions.readthedocs.io/en/latest/nbextensions/varInspector/README.html#variable-inspector>

³<https://ipython.org/ipython-doc/2/api/generated/IPython.core.magics.namespace.html>

4.1.1 Natural Language Code Description. To automatically create text summaries of dataframe changes, we apply prompt engineering [54] to generate descriptions of the executed Python code. The prompt we engineered is given below, with `<dataframe_var>` and `<code>` as placeholders for the variable name and code in the given notebook cell.

I want to explain a data transformation to a domain expert who may not be familiar with technical terms like 'dataframe' or 'transformation' as a very short summary. The goal is to make it easily understandable. Please explain what is happening to the data in the `<dataframe_var>` variable. If any specific rows or columns are being modified, kindly mention them. It doesn't have to be too verbose. Avoid using terms like 'dataframe'. If the code loads a dataset, do not infer what type of information is included in the dataset unless the information is exposed in the code.
`<code>`

This prompt makes use of zero-shot prompting [51] with specific instructions aimed to make the LLM output more friendly to non-technical readers. In our prompt engineering process, we observed that the LLM tended to provide incorrect information about a dataset based solely on the dataset name present in the code (e.g. falsely stating what columns the dataset includes). To counter these LLM hallucinations [56], we explicitly instruct the LLM to not infer the type of information included in a dataset referenced in the code.

4.1.2 Computing Subset for SnapGrid. Building upon the Smallest Timelines subset computation [24], we devise an algorithm to select a subset of 9 rows by 9 columns that maximizes the coverage of value changes for SnapGrid. The algorithm selects the dataset rows with the highest number of changed values, and the columns that have been directly affected or are implicitly involved in column changes (e.g. used to compute new columns). To determine the involved columns, we use the *text-davinci-003* LLM to infer and extract the relationships between columns in the given code. In the LLM prompt, we first provide the executed code, followed by lists of existing columns and newly added columns in the dataframe, and then ask the LLM to return a JSON object indicating the relationships between the columns involved. We demonstrate the prompt with an illustrative example in the appendix (A.1.1). By leveraging the ability of the LLM to make inferences on column dependency using pattern recognition and data science knowledge, we obtain column relationships in a robust manner.

4.1.3 Extracting Model Metrics. When a model is trained on a dataframe variable and evaluated with performance metrics calculations, we extract the following model information using the *text-davinci-003* LLM and the variable contents provided by the Python script: (1) Model Name: The name of the model, such as Linear Regression, (2) Train variables: The variables used in training, e.g. `X_train`, `Y_train`, (3) Test variables: The variables used for testing, e.g. `X_test`, `Y_test`, and (4) Metrics: A list of metric names (such as RMS error, Precision, Recall, Accuracy, etc.), their corresponding Python variable names, and the metric values. To get structured model metrics in the above format, we utilize few-shot chain-of-thought prompting [52] to *teach* the LLM the kind of

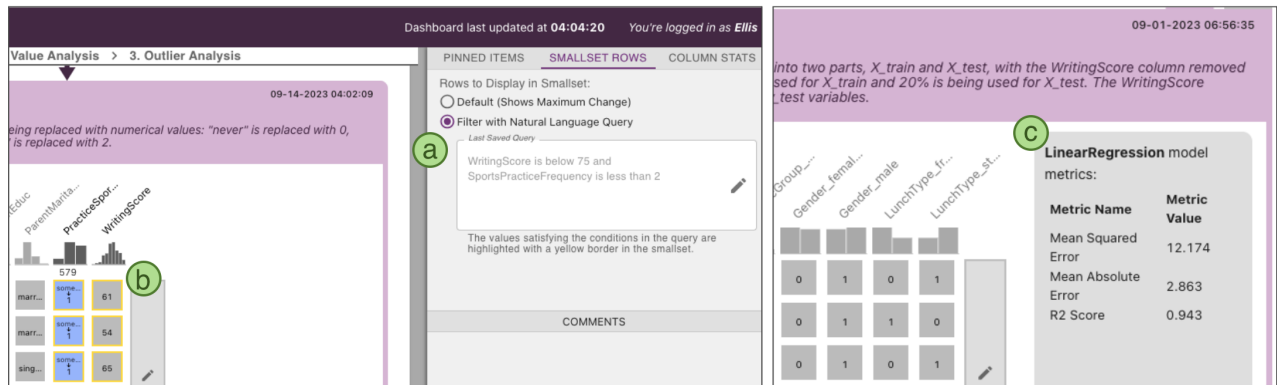


Figure 3: (a) Text field for entering a natural language query to update the selected SnapGrid. (b) Cells with values that meet the query criteria are highlighted. (c) Table display of model metrics information.

output we expect given the input. In the prompt, we first specify the desired output format and then provide several examples of input and expected output, followed by the code input that we need the LLM to generate output for. We found that, to increase the robustness of the LLM, it is necessary to provide a diverse set of code and output examples covering different models and performance metrics, as well as counterexamples that teach the model what types of input should result in empty output. The full prompt can be found in the appendix (A.1.2).

4.2 Visualization Dashboard

The visualization dashboard is built with React.js for the frontend and serverless Amazon Lambda functions as the backend. The SnapGrid and the interactive column histograms are developed with the D3 library. We provide the domain expert with a natural and flexible mechanism to update the rows in the SnapGrid by submitting a natural language query. To achieve the desired outcome, we again apply few-shot chain-of-thought prompting to train the *text-davinci-003* LLM model to transform the given natural language query to filter conditions (see full prompt in A.1.3 in the appendix). The LLM provides room for error in the natural language query as it understands the *intention* of the domain expert in the context of the dataset. Even if the domain expert fails to make exact references to column names in their queries (incorrect case, added spacing, etc.), the LLM still correctly transforms the query to conditions that can be used to filter the rows in the backend.

5 PRELIMINARY EVALUATION

In evaluating our system, our goal was to understand whether CellSync's features allow domain experts to understand the underlying data science code, and whether that understanding leads to communication and feedback to data scientists. We conducted a virtual user study on Zoom with pairs of domain experts and data scientists (a total of ten pairs). Five of the domain experts had an education background and the other five had public health expertise (see Table 1 in the appendix for participants' demographic information). The participants were recruited through our professional network and specialized communities on social platforms. All participants

had at least 1.5 years of experience in their respective fields and received a \$60 gift card for two hours of their time.

In each session, we introduced the domain expert and the data scientist to a dataset relevant to the domain expert's field and an ML prediction task for them to collaborate on (see Table 2 in the appendix for study details). To maximize the domain experts' use of the dashboard features, we placed the domain expert and the data scientist in two different Zoom rooms. After the main task, we brought the participants back to the same room and concluded the session with a 15-minute semi-structured interview. During the session, both participants' screens and the interview were recorded.

5.1 Findings

Overall, the domain expert participants found the code summary (see Table 3 in the appendix for examples) and the visualization features in the CellSync dashboard to be effective in aiding their understanding of data operations. Some participants preferred to read the text summaries to gain a high-level understanding of the data operation before deciding whether they needed to closely examine the SnapGrid. Other participants relied on the SnapGrid to understand the type and extent of data change. As the domain expert E7 commented, SnapGrids helped her understand why it may be necessary for data scientists to perform certain operations: "I really liked seeing the differences in the data highlighted... to someone who isn't a data scientist, seeing the changes visualized this way helps bridge that gap of why an operation is important for a data scientist to do to make the data easier to work with." The participants' differing preferences for the code summary and the SnapGrid suggest that displaying both components can support a wide range of domain experts in understanding data operations.

Most participants expressed that the visualization dashboard had a high learning curve. This sentiment stemmed from both the shortness of the evaluation session and the large number of features that the dashboard supported. Nonetheless, participants expressed appreciation for several features besides the code summary and the SnapGrid. In particular, the column histograms were extensively used. According to E3, "I can use them to make recommendations to the data scientist on what to pay attention to instead of relying on him to provide these basic statistics." In addition, domain experts

found the data version card navigator to be helpful, as it clearly indicated the position of the current card and eliminated the need to scroll excessively in the history view.

Through CellSync, the participant pairs actively exchanged a variety of messages as they collaborated on the prediction task. For instance, many domain expert participants inquired about the purpose of one-hot encoding operations, and their collaborators would explain that it was a data science technique. The domain experts also raised concerns around missing value imputation, providing domain-specific reasons for why it may be problematic. For example, public health experts argued that patients' missing "Region" data should not be imputed given the healthcare disparities between U.S. regions. In these cases, participant pairs engaged in discussions about the trade-offs between keeping the data true and making it usable by ML models, as they attempted to reach common ground. Our LLM-supported features helped domain experts to understand *what changed* in the data, enabling them to initiate conversations with data scientists about *why* certain data operations needed to be performed.

6 DISCUSSION AND CONCLUSION

To bridge the gap in expertise between domain experts and data scientists on ML collaboration tasks, we leveraged visualization and code summaries to help domain experts understand the impact of data science operations in the ML modeling pipeline. Further, the use of LLMs served multiple purposes in our system. First, the LLM-generated code summary reduces the communication burden on data scientists in the collaboration process, allowing them to spend more of their effort on data-specific communication rather than code-specific communication. Second, by using an LLM to extract dataframe- and model-related information from code, we utilize its capacity to understand the context and make inferences, increasing the system's robustness against a wide range of input.

Based on the preliminary evaluation findings, we aim to investigate how to apply different LLM tuning and prompting techniques to better support domain experts in our future work. One direction may be to contextualize LLM models in the given application domain, providing the LLMs with information about the collaboration task, objectives, and input data. As our education domain expert participant E1 explained that education experts typically approach education data with a standard set of considerations (e.g. correlation between socioeconomic factors and student performance), LLMs can be pre-instructed with domain-specific information to guide domain experts to more efficiently and thoroughly examine data transformations and model outputs. In addition, as research on using LLMs for code generation [23, 36] begins to mature, we aim to increase our system's support for data scientists by providing automatic code suggestions based on domain experts' natural language feedback.

Our preliminary evaluation study was conducted over short periods of synchronous collaboration between the participants, resulting in limited iteration in the modeling process and may not reflect most collaboration and communication scenarios in reality. To evaluate our system in more realistic settings, we plan to deploy CellSync to industry teams that perform collaborative ML work and collect usage data over longer periods of time in our future research

efforts. In addition, because CellSync can serve as a learning tool for data science education, we plan to integrate CellSync into university data science courses to examine its educational utility.

In conclusion, our system presents an approach to making computational notebooks accessible to domain experts, which can enhance their participation in upstream ML tasks. The use of LLMs for code summarization and data change visualizations can alleviate the engineering workload required to clarify decision-making processes and reduce friction in collaborative ML practices.

ACKNOWLEDGMENTS

This work was partially supported by IBM as a founding member of Stanford Institute for Human-centered Artificial Intelligence (HAI).

REFERENCES

- [1] Toufique Ahmed and Premkumar Devanbu. 2023. Few-Shot Training LLMs for Project-Specific Code-Summarization. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering (Rochester, MI, USA) (ASE '22)*. Association for Computing Machinery, New York, NY, USA, Article 177, 5 pages. <https://doi.org/10.1145/3551349.3559555>
- [2] Gulsum Alicioglu and Bo Sun. 2022. A survey of visual analytics for Explainable Artificial Intelligence methods. *Computers & Graphics* 102 (2022), 502–520. <https://doi.org/10.1016/j.cag.2021.09.002>
- [3] Matthew Arnold, Rachel KE Bellamy, Michael Hind, Stephanie Houde, Sameep Mehta, Aleksandra Mojsilović, Ravi Nair, K Natesan Ramamurthy, Alexandra Olteanu, David Piorkowski, et al. 2019. FactSheets: Increasing trust in AI services through supplier's declarations of conformity. *IBM Journal of Research and Development* 63, 4/5 (2019), 6–1.
- [4] Ryan S. Baker and Aaron Hawn. 2021. Algorithmic bias in Education. *International Journal of Artificial Intelligence in Education* 32, 4 (2021), 1052–1092. <https://doi.org/10.1007/s40593-021-00285-9>
- [5] Wesley Hanwen Deng, Nur Yildirim, Monica Chang, Motahhareh Eslami, Kenneth Holstein, and Michael Madaio. 2023. Investigating Practices and Opportunities for Cross-Functional Collaboration around AI Fairness in Industry Practice. In *Proceedings of the 2023 ACM Conference on Fairness, Accountability, and Transparency (Chicago, IL, USA) (FAccT '23)*. Association for Computing Machinery, New York, NY, USA, 705–716. <https://doi.org/10.1145/3593013.3594037>
- [6] Des. and Royce Kimmons. 2023. Students exam scores: Extended dataset. <https://www.kaggle.com/datasets/deslegnggeb/students-exam-scores>
- [7] Derek Doran, Sarah Schulz, and Tarek R. Besold. 2017. What Does Explainable AI Really Mean? A New Conceptualization of Perspectives. *CoRR* abs/1710.00794 (2017). arXiv:1710.00794 <http://arxiv.org/abs/1710.00794>
- [8] Jun Du and Charles X. Ling. 2010. Asking Generalized Queries to Domain Experts to Improve Learning. *IEEE Transactions on Knowledge and Data Engineering* 22, 6 (2010), 812–825. <https://doi.org/10.1109/TKDE.2010.33>
- [9] Rudresh Dwivedi, Devam Dave, Het Naik, Smriti Singhal, Rana Omer, Pankesh Patel, Bin Qian, Zhenyu Wen, Tejal Shah, Graham Morgan, and Rajiv Ranjan. 2023. Explainable AI (XAI): Core Ideas, Techniques, and Solutions. *ACM Comput. Surv.* 55, 9, Article 194 (jan 2023), 33 pages. <https://doi.org/10.1145/3561048>
- [10] Bahar Farahani, Mojtaba Barzegari, and Fereidoon Shams Aliee. 2019. Towards Collaborative Machine Learning Driven Healthcare Internet of Things. In *Proceedings of the International Conference on Omni-Layer Intelligent Systems (Crete, Greece) (COINS '19)*. Association for Computing Machinery, New York, NY, USA, 134–140. <https://doi.org/10.1145/3312614.3312644>
- [11] Krishna Gade, Sahin Cem Geyik, Krishnaram Kenthapadi, Varun Mithal, and Ankur Taly. 2019. Explainable AI in Industry. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (Anchorage, AK, USA) (KDD '19)*. Association for Computing Machinery, New York, NY, USA, 3203–3204. <https://doi.org/10.1145/3292500.3332281>
- [12] Josh Gardner, Christopher Brooks, and Ryan Baker. 2019. Evaluating the Fairness of Predictive Student Models Through Slicing Analysis. In *Proceedings of the 9th International Conference on Learning Analytics & Knowledge (Tempe, AZ, USA) (LAK19)*. Association for Computing Machinery, New York, NY, USA, 225–234. <https://doi.org/10.1145/3303772.3303791>
- [13] Ayush Garg and Vaccine Adverse Event Reporting System (VAERS). 2021. Covid-19 world vaccine adverse reactions. <https://www.kaggle.com/datasets/ayushgarg/covid19-vaccine-adverse-reactions>
- [14] Timnit Gebru, Jamie Morgenstern, Briana Vecchione, Jennifer Wortman Vaughan, Hanna Wallach, Hal Daumé III, and Kate Crawford. 2021. Datasheets for datasets. *Commun. ACM* 64, 12 (2021), 86–92.
- [15] Milena A. Gianfrancesco, Suzanne Tamang, Jinoos Yazdany, and Gabriela Schmajuk. 2018. Potential Biases in Machine Learning Algorithms Using Electronic

- Health Record Data. *JAMA Internal Medicine* 178, 11 (11 2018), 1544–1547. <https://doi.org/10.1001/jamainternmed.2018.3763>
- [16] Amy K. Heger, Liz B. Marquis, Mihaela Vorvoreanu, Hanna Wallach, and Jennifer Wortman Vaughan. 2022. Understanding Machine Learning Practitioners' Data Documentation Perceptions, Needs, Challenges, and Desiderata. *Proc. ACM Hum.-Comput. Interact.* 6, CSCW2, Article 340 (nov 2022), 29 pages. <https://doi.org/10.1145/3555760>
- [17] Fred Hohman, Kanit Wongsuphasawat, Mary Beth Kery, and Kayur Patel. 2020. Understanding and Visualizing Data Iteration in Machine Learning. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) (CHI '20). Association for Computing Machinery, New York, NY, USA, 1–13. <https://doi.org/10.1145/3313831.3376177>
- [18] Andreas Holzinger, Chris Biemann, Constantinos S. Pattichis, and Douglas B. Kell. 2017. What do we need to build explainable AI systems for the medical domain? *CoRR* abs/1712.09923 (2017). <http://arxiv.org/abs/1712.09923>
- [19] Jonathan Huang, Galal Galal, Mozziyar Etemadi, and Mahesh Vaidyanathan. 2022. Evaluation and mitigation of racial bias in clinical machine learning models: Scoping review. *JMIR Medical Informatics* 10, 5 (2022). <https://doi.org/10.2196/36388>
- [20] Anna Jobin, Marcello Ienca, and Effy Vayena. 2019. The global landscape of AI ethics guidelines. <https://www.nature.com/articles/s42256-019-0088-2>
- [21] Ju Yeon Jung, Tom Steinberger, John L. King, and Mark S. Ackerman. 2022. How Domain Experts Work with Data: Situating Data Science in the Practices and Settings of Craftwork. 6, CSCW1, Article 58 (apr 2022), 29 pages. <https://doi.org/10.1145/3512905>
- [22] Jupyter. 2023. *Project Jupyter*. <https://jupyter.org/>
- [23] Peng Li, Tianxiang Sun, Qiong Tang, Hang Yan, Yuanbin Wu, Xuanjing Huang, and Xipeng Qiu. 2023. CodeIE: Large Code Generation Models are Better Few-Shot Information Extractors. [arXiv:2305.05711](https://arxiv.org/abs/2305.05711) [cs.CL]
- [24] Lydia R. Lucchesi, Petra M. Kuhnert, Jenny L. Davis, and Lexing Xie. 2022. Small-set Timelines: A Visual Representation of Data Preprocessing Decisions. In *Proceedings of the 2022 ACM Conference on Fairness, Accountability, and Transparency* (Seoul, Republic of Korea) (FAccT '22). Association for Computing Machinery, New York, NY, USA, 1136–1153. <https://doi.org/10.1145/3531146.3533175>
- [25] Joseph MacInnes, Stephanie Santosa, and William Wright. 2010. Visual Classification: Expert Knowledge Guides Machine Learning. *IEEE Computer Graphics and Applications* 30, 1 (2010), 8–14. <https://doi.org/10.1109/MCG.2010.18>
- [26] Yaoli Mao, Dakuo Wang, Michael Muller, Kush R. Varshney, Ioana Baldini, Casey Dugan, and Aleksandra Mojsilović. 2019. How Data Scientists Work Together With Domain Experts in Scientific Collaborations: To Find The Right Answer Or To Ask The Right Question? *Proc. ACM Hum.-Comput. Interact.* 3, GROUP, Article 237 (dec 2019), 23 pages. <https://doi.org/10.1145/3361118>
- [27] Sewon Min, Xinxi Lyu, Ari Holtzman, Mikel Artetxe, Mike Lewis, Hannaneh Hajishirzi, and Luke Zettlemoyer. 2022. Rethinking the role of demonstrations: What makes in-context learning work? (2022). <https://doi.org/10.18653/v1/2022.emnlp-main.759>
- [28] Margaret Mitchell, Simone Wu, Andrew Zaldivar, Parker Barnes, Lucy Vasserman, Ben Hutchinson, Elena Spitzer, Inioluwa Deborah Raji, and Timnit Gebru. 2019. Model cards for model reporting. In *Proceedings of the conference on fairness, accountability, and transparency*. 220–229.
- [29] Sina Mohseni, Niloofar Zarei, and Eric D. Ragan. 2021. A Multidisciplinary Survey and Framework for Design and Evaluation of Explainable AI Systems. *ACM Trans. Interact. Intell. Syst.* 11, 3–4, Article 24 (sep 2021), 45 pages. <https://doi.org/10.1145/3387166>
- [30] Lyngre Asbjørn Møller. 2023. Bridging the Tech-Editorial Gap: Lessons from Two Case Studies of the Development and Integration of Algorithmic Curation in Journalism. *Journalism Studies* 24, 11 (2023), 1458–1475. <https://doi.org/10.1080/1461670X.2023.2227283>
- [31] Nadia Nahar, Shurui Zhou, Grace Lewis, and Christian Kästner. 2022. Collaboration Challenges in Building ML-Enabled Systems: Communication, Documentation, Engineering, and Process. In *Proceedings of the 44th International Conference on Software Engineering* (Pittsburgh, Pennsylvania) (ICSE '22). Association for Computing Machinery, New York, NY, USA, 413–425. <https://doi.org/10.1145/3510003.3510209>
- [32] Christina Niederer, Holger Stitz, Reem Hourieh, Florian Grassinger, Wolfgang Aigner, and Marc Streit. 2018. TACO: Visualizing Changes in Tables Over Time. *IEEE Transactions on Visualization and Computer Graphics* 24, 1 (2018), 677–686. <https://doi.org/10.1109/TVCG.2017.2745298>
- [33] Mahsan Nourani, Joanie T. King, and Eric D. Ragan. 2020. The Role of Domain Expertise in User Trust and the Impact of First Impressions with Intelligent Systems. [arXiv:2008.09100](https://arxiv.org/abs/2008.09100) [cs.HC]
- [34] Ziad Obermeyer, Brian Powers, Christine Vogeli, and Sendhil Mullainathan. 2019. Dissecting racial bias in an algorithm used to manage the health of populations. *Science* 366, 6464 (2019), 447–453. <https://doi.org/10.1126/science.aax2342>
- [35] OpenAI. 2023. *OpenAI*. <https://openai.com/>
- [36] Shuyin Ouyang, Jie M. Zhang, Mark Harman, and Meng Wang. 2023. LLM is Like a Box of Chocolates: the Non-determinism of ChatGPT in Code Generation. [arXiv:2308.02828](https://arxiv.org/abs/2308.02828) [cs.SE]
- [37] Soya Park, April Yi Wang, Ban Kawas, Q. Vera Liao, David Piorkowski, and Marina Danilevsky. 2021. Facilitating Knowledge Sharing from Domain Experts to Data Scientists for Building NLP Models. In *26th International Conference on Intelligent User Interfaces* (College Station, TX, USA) (IUI '21). Association for Computing Machinery, New York, NY, USA, 585–596. <https://doi.org/10.1145/3397481.3450637>
- [38] Samir Passi and Steven J. Jackson. 2018. Trust in Data Science: Collaboration, Translation, and Accountability in Corporate Data Science Projects. 2, CSCW, Article 136 (nov 2018), 28 pages. <https://doi.org/10.1145/3274405>
- [39] Mateus M. Pereira and Fernando V. Paulovich. 2020. RankViz: A visualization framework to assist interpretation of Learning to Rank algorithms. *Computers & Graphics* 93 (2020), 25–38. <https://doi.org/10.1016/j.cag.2020.09.017>
- [40] David Piorkowski, Soya Park, April Yi Wang, Dakuo Wang, Michael Muller, and Felix Portnoy. 2021. How AI Developers Overcome Communication Challenges in a Multidisciplinary Team: A Case Study. *Proc. ACM Hum.-Comput. Interact.* 5, CSCW1, Article 131 (apr 2021), 25 pages. <https://doi.org/10.1145/3449205>
- [41] Deepthi Raghunandan, Aayushi Roy, Shenzi Shi, Niklas Elmqvist, and Leilani Battle. 2023. Code Code Evolution: Understanding How People Change Data Science Notebooks Over Time. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems* (Hamburg, Germany) (CHI '23). Association for Computing Machinery, New York, NY, USA, Article 863, 12 pages. <https://doi.org/10.1145/3544548.3580997>
- [42] Aayushi Roy, Deepthi Raghunandan, Niklas Elmqvist, and Leilani Battle. 2023. How I Met Your Data Science Team: A Tale of Effective Communication. In *2023 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. 199–208. <https://doi.org/10.1109/VL-HCC57772.2023.00032>
- [43] Nithya Sambasivan, Shivani Kapania, Hannah Highfill, Diana Akrong, Praveen Paritosh, and Lora M Aroyo. 2021. “Everyone Wants to Do the Model Work, Not the Data Work”: Data Cascades in High-Stakes AI. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems* (Yokohama, Japan) (CHI '21). Association for Computing Machinery, New York, NY, USA, Article 39, 15 pages. <https://doi.org/10.1145/3411764.3445518>
- [44] Hariharan Subramonyam, Jane Im, Colleen Seifert, and Eytan Adar. 2022. Solving separation-of-concerns problems in collaborative design of human-AI systems through leaky abstractions. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*. 1–21.
- [45] Mei Tan, Hansol Lee, Dakuo Wang, and Hariharan Subramonyam. 2023. Is a Seat at the Table Enough? Engaging Teachers and Students in Dataset Specification for ML in Education. [arXiv:2311.05792](https://arxiv.org/abs/2311.05792) [cs.CY]
- [46] Stijn Viaene. 2013. Data scientists aren't domain experts. *IT Professional* 15, 6 (2013), 12–17.
- [47] April Yi Wang, Anant Mittal, Christopher Brooks, and Steve Oney. 2019. How Data Scientists Use Computational Notebooks for Real-Time Collaboration. *Proc. ACM Hum.-Comput. Interact.* 3, CSCW, Article 39 (nov 2019), 30 pages. <https://doi.org/10.1145/3359141>
- [48] April Yi Wang, Zihan Wu, Christopher Brooks, and Steve Oney. 2020. Callisto: Capturing the “Why” by Connecting Conversations with Computational Narratives (CHI '20). Association for Computing Machinery, New York, NY, USA, 1–13. <https://doi.org/10.1145/3313831.3376740>
- [49] Junbo Wang, Amitangshu Pal, Qinglin Yang, Krishna Kant, Kaiming Zhu, and Song Guo. 2022. Collaborative Machine Learning: Schemes, Robustness, and Privacy. *IEEE Transactions on Neural Networks and Learning Systems* (2022), 1–18. <https://doi.org/10.1109/TNNLS.2022.3169347>
- [50] Zijie J. Wang, Alex Kale, Harsha Nori, Peter Stella, Mark E. Nunnally, Duen Horng Chau, Mihaela Vorvoreanu, Jennifer Wortman Vaughan, and Rich Caruana. 2022. Interpretability, Then What? Editing Machine Learning Models to Reflect Human Knowledge and Values. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining* (Washington DC, USA) (KDD '22). Association for Computing Machinery, New York, NY, USA, 4132–4142. <https://doi.org/10.1145/3534678.3539074>
- [51] Jason Wei, Maarten Bosma, Vincent Y. Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V. Le. 2022. Finetuned Language Models Are Zero-Shot Learners. [arXiv:2109.01652](https://arxiv.org/abs/2109.01652) [cs.CL]
- [52] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems* 35 (2022), 24824–24837.
- [53] Steven Euijong Whang, Yuji Roh, Hwanjun Song, and Jae-Gil Lee. 2023. Data Collection and quality challenges in Deep learning: A data-centric AI perspective. *The VLDB Journal* 32, 4 (2023), 791–813. <https://doi.org/10.1007/s00778-022-00775-9>
- [54] Jules White, Quchen Fu, Sam Hays, Michael Sandborn, Carlos Olea, Henry Gilbert, Ashraf Elnashar, Jesse Spencer-Smith, and Douglas C. Schmidt. 2023. A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT. [arXiv:2302.11382](https://arxiv.org/abs/2302.11382) [cs.SE]
- [55] Yuet Ling Wong, Krishna Madhavan, and Niklas Elmqvist. 2018. Towards Characterizing Domain Experts as a User Group. In *2018 IEEE Evaluation and Beyond - Methodological Approaches for Visualization (BELIV)*. 1–10. <https://doi.org/10.1145/3274405>

//doi.org/10.1109/BELIV.2018.8634026

[56] Yue Zhang, Yafu Li, Leyang Cui, Deng Cai, Lema Liu, Tingchen Fu, Xinting Huang, Enbo Zhao, Yu Zhang, Yulong Chen, Longyue Wang, Anh Tuan Luu, Wei Bi, Freda Shi, and Shuming Shi. 2023. Siren's Song in the AI Ocean: A Survey on Hallucination in Large Language Models. arXiv:2309.01219 [cs.CL]

A APPENDIX

A.1 LLM Prompts

A.1.1 Column Relationship Extraction. Below is an illustrative example of the prompt filled with sample code and lists of existing and newly added columns (in parts that follow a #):

```
#Code
df = pd.get_dummies(df, columns=["Gender"])
#List of Existing Columns = ["Gender", "Age"]
#List of Newly Added Columns = ["Gender_Female",
"Gender_Male"]

Are there new column(s) computed based on the exist-
ing columns in the code? If yes, provide the answer in the
following format: <new_column>:<existing columns used
to compute this specific new column>. Remember to put
double quotes (NOT single quotes) around the column names
in the output format to make it a valid JSON string. If there
are multiple detections, append the dictionary. If no, then just
return an empty dictionary.

It is possible that new columns were generated by one-hot
encoding techniques. In that case, make your best guess about
which existing column was used to generate the new ones.
Answer:
```

In this example, the LLM correctly returns [{"Gender_Female": ["Gender"]}, {"Gender_Male": ["Gender"]}], even though the provided code does not explicitly show that the two new columns were computed based on the "Gender" column.

A.1.2 Model Metrics Extraction. The following is the full prompt we give to the LLM, with <input_code> as placeholder for the code in the notebook cell we need the LLM to extract model information from.

```
I'll give you a piece of code used in an ML model, and you
need to identify some metadata about the model in JSON
format. The metadata includes 'Model Name' (a string), 'Train
Variables' (a list), 'Test Variables' (a list), 'Metrics' (a list of
objects each consisting of keys 'Metric' and 'Metric Variable'
whose values are strings). For example, given
Input:
# imports
import numpy as np
from sklearn.linear_model import
LinearRegression
from sklearn.metrics import mean_squared_error
```

```
from sklearn.metrics import
mean_absolute_error
# Generate some sample data
X_train = np.array([[1], [2], [3], [4], [5]])
Y_train = np.array([3, 5, 7, 9, 11])
# Train a linear regression model
reg = LinearRegression().fit(X_train, Y_train
)
# Test variables
X_test = np.array([6], [7])
y_test = ([13, 15])
y_test_pred = reg.predict(X_test)
# Calculate the mean squared error for the
test data
mse_test = mean_squared_error(y_test,
y_test_pred)
print("Mean squared error for test data:",
mse_test)
# Calculate the mean absolute error for the
test data
mae_test = mean_squared_error(y_test,
y_test_pred)
print("Mean absolute error for test data:",
mse_test)
```

Output:

```
{
"Model Name": "LinearRegression",
"Train variables": ["X_train", "Y_train"],
"Test variables": ["X_test", "y_test"],
"Metrics": [
{"Metric": "Mean Squared Error", "Metric
Variable": "mse_test"},
{"Metric": "Mean Absolute Error", "Metric
Variable": "mae_test"}
]
}
```

Sometimes, the test variables might not be present. In that case, you can return an empty "Test variables". For example given

Input:

```
import numpy as np
from sklearn.linear_model import
LinearRegression
from sklearn.metrics import mean_squared_error
# Generate some sample data
X_train1 = np.array([[1], [2], [3], [4], [5]])
Y_train1 = np.array([3, 5, 7, 9, 11])
# Train a linear regression model
reg1 = LinearRegression().fit(X_train1,
Y_train1)
# Make predictions for the testing data
y_pred1 = reg1.predict(X_train1)
# Calculate the mean squared error
```

```
mse1 = mean_squared_error(Y_train1, y_pred1)
```

Output:

```
{
  "Model Name": "LinearRegression",
  "Train Variables": ["X_train1", "Y_train1"],
  "Test Variables": [],
  "Metrics": [
    {"Metric": "Mean Squared Error", "Metric
      Variable": "mse1"}
  ]
}
```

Now, I'll give you another piece of code. Identify the metadata in it.

Input:

```
from sklearn.linear_model import
  LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.model_selection import
  train_test_split
from sklearn.datasets import load_iris
# Load the iris dataset
iris = load_iris()
# Split the data into training and test sets
X_train, X_test, y_train, y_test =
  train_test_split(iris.data, iris.target,
    test_size=0.2, random_state=42)
# Train a logistic regression classifier
clf = LogisticRegression(random_state=42).fit(
  X_train, y_train)
# Make predictions for the test set
y_pred = clf.predict(X_test)
# Calculate the accuracy of the classifier
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
# Calculate the precision of the classifier
precision = precision_score(y_test, y_pred)
print("Precision:", precision)
# Calculate the recall of the classifier
recall = recall_score(y_test, y_pred)
print("Recall:", recall)
```

Output:

```
{
  "Model Name": "LogisticRegression",
  "Train Variables": ["X_train", "y_train"],
  "Test Variables": ["X_test", "y_test"],
  "Metrics": [
    {"Metric": "Accuracy", "Metric Variable":
      "accuracy"},
    {"Metric": "Precision", "Metric Variable":
      "precision"},
  ]
}
```

```
    {"Metric": "Recall", "Metric Variable": "
      recall"}
  ]
}
```

Very good! Here's another piece of code. Identify the metadata in it.

Input:

```
import tensorflow as tf
from tensorflow import keras
import numpy as np
# Load the CIFAR-10 dataset
(x_train, y_train), (x_test, y_test) = keras.
  datasets.cifar10.load_data()
# Select a subset of the data
n_classes = 3
class_names = ['airplane', 'automobile', 'bird
  ']
idx_train = np.isin(y_train, range(n_classes))
idx_test = np.isin(y_test, range(n_classes))
x_train, y_train = x_train[idx_train], y_train
  [idx_train]
x_test, y_test = x_test[idx_test], y_test[
  idx_test]
# Preprocess the data
x_train = x_train /
  255.0
x_test = x_test / 255.0
# Define the model architecture
model = keras.Sequential([
  keras.layers.Conv2D(32, (3, 3), activation='
    relu', input_shape=x_train.shape[1:]),
  keras.layers.MaxPooling2D((2, 2)),
  keras.layers.Conv2D(64, (3, 3), activation='
    relu'),
  keras.layers.MaxPooling2D((2, 2)),
  keras.layers.Flatten(),
  keras.layers.Dense(64, activation='relu'),
  keras.layers.Dense(n_classes)
])
# Compile the model
model.compile(optimizer='adam',
  loss=tf.keras.losses.
    SparseCategoricalCrossentropy(
      from_logits=True),
  metrics=['accuracy'])
# Train the model
model.fit(x_train, y_train, epochs=10,
  validation_data=(x_test, y_test))
# Evaluate the model on the test set
test_loss, test_acc = model.evaluate(x_test,
  y_test)
print('Test accuracy:', test_acc)
```

Output:

```
{
```

```

"Model Name": "Keras Sequential",
"Train Variables": ["x_train", "y_train"],
"Test Variables": ["x_test", "y_test"],
"Metrics": [{"Metric": "Accuracy", "Metric
Variable": "test_acc"}]
}

```

Note that you should only extract model meta data when an ML model is present. Do not treat methods in sklearn preprocessing modules used for scaling, normalization, or binarization, such as StandardScaler and RobustScaler (just to name a few), as ML models. Also do not treat transformers for missing value imputation, such as SimpleImputer and MissingIndicator as ML models. If no ML model is present in the code, simply return an empty object. For example, given

Input:

```

from sklearn.impute import SimpleImputer
imp_mf = SimpleImputer(strategy='most_frequent',
missing_values=np.nan)
for col in scores.drop(['Gender', 'MathScore'],
axis=1).columns:
scores[col] = imp_mf.fit_transform(scores[[col]])

```

Output:

```
{}
```

Now, I'll give you another piece of code. Identify the metadata in it.

Input:

```
<input_code>
```

Output:

The part before the short horizontal dashed line containing multiple pairs of input and output examples *train* the LLM to return certain outputs, given certain inputs. By training the LLM through multiple steps, we are able to cover different types of inputs and corner cases before we ask it to extract model metrics on new input. The part after the dashed line contains the actual notebook cell code as input and the cue for the LLM to return the output.

A.1.3 Transforming Natural Language Query to Filter Conditions. Similar to extracting model metrics in A.1.2, we show the *fine-tuning* part before the short horizontal dashed line and the *evaluation* part after the dashed line in the prompt to the LLM, with placeholders for the column list and the natural language query denoted using <columns> and <natural_language_query>.

I have a JSON object in Javascript, which represents a python dataframe. We need to get filtering conditions from it based on a input text. I'm going to teach you how to return a JSON object

which is an array, each element containing '(columnName, condition, value)'. For example:

Columns: [Glucose, Age, Gender, Outcome]

Input: "Show me rows/patients having glucose value > 90 and between the age of 25 to 35"

Output:

```

[
{"column": "Glucose", "operator": ">", "value": "90"},
{"column": "Age", "operator": ">=", "value": "25"},
{"column": "Age", "operator": "<=", "value": "35"}
]

```

Sometimes the 'value' might be a string as well. For example: Columns: [Melatonin, Sickness level, Gender, Race, Predicted age]

Input: "Show me Female subjects whose melatonin is greater than 3.5"

Output:

```

[
{"column": "Gender", "operator": "==", "value": "Female"},
{"column": "Melatonin", "operator": ">", "value": "3.5"}
]

```

Now lets do some practice, give me the output for:

Columns: [Students Name, Education level, Parents education level, Dropped out]

Input: "Show me students whose parents' education level is High School and whose Dropped out is 1"

Output:

```

[
{"column": "Parents education level", "operator": "=", "value": "High School"},
{"column": "Dropped out", "operator": "=", "value": "1"}
]

```

Nice, now give me the output for:

Columns: <columns>

Input: <natural_language_query>

Output:

A.2 Evaluation Study Tables

Table 1: Evaluation Study Participants Demographics

	Domain Experts			Data Scientists		
	Domain	Gender	Experience (Years)	Gender	Experience (Years)	
E1	Education	F	10	S1	M	2
E2	Education	F	15	S2	M	4
E3	Education	F	5	S3	M	3
E4	Education	F	15	S4	M	10
E5	Education	M	10	S5	M	2
E6	Public Health	F	5	S6	M	1.5
E7	Public Health	F	5	S7	M	2
E8	Public Health	F	3	S8	M	5
E9	Public Health	F	2	S9	M	5
E10	Public Health	F	4	S10	M	4

Table 2: Descriptions of Evaluation Study Prediction Tasks and Datasets By Domain

	Public Health Domain	Education Domain
Prediction Task	Predict whether hospitalized patients will pass away due to adverse covid vaccine reaction	Predict students' writing exam scores
Dataset Source & Modification	Vaccine Adverse Event Reporting System (VAERS) dataset [13], filtered down to entries corresponding to covid vaccine reactions and hospitalized patients for the year 2021, followed by random sampling for subset extraction	Fictional high school student dataset incorporating missing values and realistic socio-economic factors [6], randomly sampled to extract a subset
Dataset Dimension	600 rows and 12 columns	600 rows and 12 columns
Feature Columns	Patient background information such as patient age, U.S. region, sex, disability, concurrent medications, etc.; Vaccine reaction data such as days of hospitalization, number of days to onset, vaccine manufacturer, ER visit, etc.	Student background information such as gender, ethnic group (as category A to E), parent's education level, number of siblings, free lunch eligibility, etc.; Academic performance data such as weekly studying hours, exam scores on reading and math
Outcome Column	Binary value indicating whether patient has died following hospitalization for vaccine reaction	Integer between 0 and 100 indicating score on writing exam

Table 3: Examples of LLM-generated Code Summaries

Data Operation	LLM-generated Summary
Dataset Loading	The code is loading data from the student_exam_scores.csv file.
Missing Value Imputation	The code is replacing any missing values in the "ParentEducation" column of the dataset with the most frequent value in that column.
Replacing Missing Values with Label	This code is replacing any empty values in the 'EthnicGroup' column with the value 'unknown'.
Outlier Removal	The data transformation is removing two rows (413 and 470) from the dataset that contain outlier values in the ReadingScore, WritingScore, and MathScore columns.
One-hot Encoding	This transformation is creating new columns for each unique value of the "Gender" column in the dataset. For each row, the new columns will have a value of 0 or 1 depending on whether the row matches the value of the original column.
Transformation from Categorical to Numeric	The values in the "TestPrepCourse" column of the dataset are being replaced with 0 or 1 depending on whether the value is "none" or "completed".
Feature Filtering	This code is removing the column named "PracticeSport" from the dataset.
Dataset Splitting	The data from the dataset is being split into two parts: X_train and X_test. X_train contains the data from the dataset, excluding the column "WritingScore". The data in X_train is then used to train a model. X_test contains the same data as X_train, but also includes the column "WritingScore". The data in X_test is then used to test the model.