



# Documentation Matters: Human-Centered AI System to Assist Data Science Code Documentation in Computational Notebooks

APRIL YI WANG, University of Michigan  
DAKUO WANG, IBM Research  
JAIMIE DROZDAL, Rensselaer Polytechnic Institute  
MICHAEL MULLER, IBM Research  
SOYA PARK, MIT CSAIL  
JUSTIN D. WEISZ, IBM Research  
XUYE LIU, Rensselaer Polytechnic Institute  
LINGFEI WU and CASEY DUGAN, IBM Research

---

Computational notebooks allow data scientists to express their ideas through a combination of code and documentation. However, data scientists often pay attention only to the code, and neglect creating or updating their documentation during quick iterations. Inspired by human documentation practices learned from 80 highly-voted Kaggle notebooks, we design and implement Themisto, an automated documentation generation system to explore how human-centered AI systems can support human data scientists in the machine learning code documentation scenario. Themisto facilitates the creation of documentation via three approaches: a deep-learning-based approach to generate documentation for source code, a query-based approach to retrieve online API documentation for source code, and a user prompt approach to nudge users to write documentation. We evaluated Themisto in a within-subjects experiment with 24 data science practitioners, and found that automated documentation generation techniques reduced the time for writing documentation, reminded participants to document code they would have ignored, and improved participants' satisfaction with their computational notebook.

CCS Concepts: • **Human-centered computing** → **Interactive systems and tools**; **Empirical studies in HCI**; • **Computing methodologies** → *Natural language generation*; • **Software and its engineering** → *Documentation*;

Additional Key Words and Phrases: Code summarization, computational notebooks, code documentation

---

A. Y. Wang and D. Wang contributed equally to this research.

Authors' addresses: A. Y. Wang (corresponding author), University of Michigan, 105 South State Street, Ann Arbor, MI, 48109, USA; email: aprilww@umich.edu; D. Wang (corresponding author), M. Muller, J. D. Weisz, L. Wu, and C. Dugan, IBM Research, 75 Binney Street, Cambridge, MA, 02142, USA; emails: dakuo.wang@ibm.com, {michael\_muller, jweisz, wuli, cadugan}@us.ibm.com; J. Drozdal and X. Liu, Rensselaer Polytechnic Institute, 110 Eighth St, Troy, NY, 12180, USA; emails: {drozjd3, liux27}@rpi.edu; S. Park, MIT CSAIL, 32 Vassar St., Cambridge, MA, 02139, USA; email: soya@mit.edu.

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 components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org).

© 2022 Association for Computing Machinery.

1073-0516/2022/01-ART17 \$15.00

<https://doi.org/10.1145/3489465>

**ACM Reference format:**

April Yi Wang, Dakuo Wang, Jaimie Drozdal, Michael Muller, Soya Park, Justin D. Weisz, Xuye Liu, Lingfei Wu, and Casey Dugan. 2022. Documentation Matters: Human-Centered AI System to Assist Data Science Code Documentation in Computational Notebooks. *ACM Trans. Comput.-Hum. Interact.* 29, 2, Article 17 (January 2022), 33 pages.

<https://doi.org/10.1145/3489465>

---

## 1 INTRODUCTION

Documenting the story behind code and results is critical for data scientists to collaborate effectively with others, as well as their future selves [26, 30, 42, 51]. The story, code, and computational results together construct a computational narrative. Unfortunately, data scientists often write messy and drafty analysis code in computational notebooks as they need to quickly test hypotheses and experiment with alternatives. It is a tedious process for data scientists to then manually document and refactor the raw notebook into a more readable computational narrative, thus many people neglect to do so [57].

Many efforts have sought to address the tension between *exploration* and *explanation* in computational notebooks. For example, researchers have explored the use of code gathering techniques to help data scientists organize cluttered and inconsistent notebooks [15], as well as algorithmic and visualization approaches to help data scientists forage past analysis choices [25]. But these efforts focus on the cleaning and organizing of existing notebook content, instead of creating the new content. Another work developed a chat feature that enables data scientists to have simultaneous discussions while coding in a notebook [65], and linked their chat messages as documentations to relevant notebook elements as in Google Docs [66]. However, these chat messages are too fragmented and colloquial to be used for documentation; besides, in real practice data scientists and business analysts rarely work on notebooks at the same time and actively message each other.

We began our project by asking, “What makes a well-documented notebook?” To answer this question, we first conducted an in-depth analysis of how human data scientists document notebooks. Publicly shared user notebooks on Github are often not well documented [57], thus we look up to a special set of notebooks—the highly-voted notebooks users submitted to Kaggle competitions. We conducted a formative study with a sample of 80 of these notebooks, and our interactive in-depth coding analysis suggested these 80 notebooks have much better documentation in comparison to the corpus reported in previous literature [57]. Thus, we refer to them as “well-documented” notebooks.

Our coding process of these 80 notebooks also revealed a taxonomy of nine categories (e.g., Reason, Process, Result) for the documentation content, which reflects the thought processes and decisions made by the notebook owner. These findings together with the insights from related work motivate us to consider **Artificial Intelligence AI** automation as a potential solution to support the human process of crafting documentation.

We propose Themisto, an automated code documentation generation system that integrates into the Jupyter Notebook environment. To support the diverse types of documentation content and to complement the AI limitations, Themisto incorporate three distinct approaches: a deep-learning-based approach to automatically generate new documentation for source code (fully automated); a query-based approach to retrieve existing documentation from online **Application Programming Interface (API)** websites for third party packages and libraries (fully automated); and a prompt-based approach to give users a start of the sentence and encourage them to complete the sentence that serves as documentation (semi-automated).

We evaluated Themisto in a within-subjects experiment with 24 data science practitioners. We found that Themisto reduced the time for data scientists to create documentation, reminded

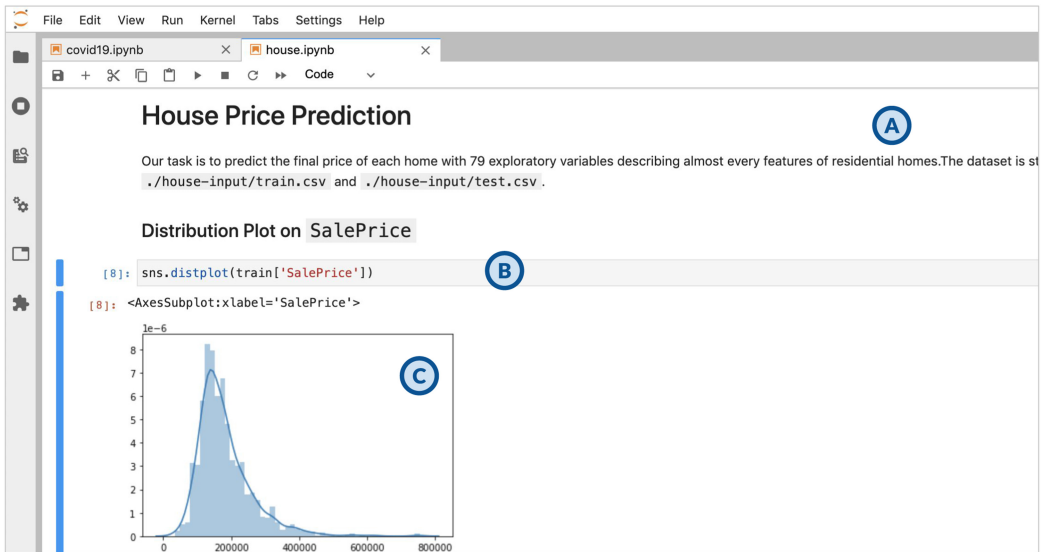


Fig. 1. Computational notebooks allow data scientists to create (A) markdown cells and (B) code cells, and view (C) code output in the same environment. Together, the variety of media including, text explanations, graphs, forms, interactive visualizations, code segments and their outputs—weaves into computational narratives.

them to document code they would have ignored, and improved their satisfaction with their computational notebooks. Meanwhile, the quality of the documentation produced with Themisto is about the same as what data scientists produced on their own. Based on these findings, we re-imagine that the code documentation task can be conducted in a Human-AI Collaboration fashion in the future, where this joint effort may have unique advantages in comparison to the solo effort of a human alone.

Our article provides a three-fold contribution to the **Human-Computer Interaction (HCI)** and data science practitioner communities:

- providing an empirical understanding of best practices of how human documenting a notebook through an analysis of highly-rated Kaggle notebooks,
- demonstrating the design of a human-centered AI system that can collaborate with human data scientists to create high-quality computational narratives,
- reporting empirical evidence that Themisto can collaborate with data scientists to generate high quality and highly-satisfied computational notebooks in much less time.

## 2 RELATED WORK

Our work builds on top of both HCI and **Machine Learning (ML)** fields. Thus, our literature review briefly summarizes the work of both, with a focus on the following three topics: computational notebook management, code documentation supporting systems, and neural-network-based code summarization.

### 2.1 Computational Notebooks as Computational Narrative

Computational notebooks allow data scientists to weave together a variety of media, including text explanations, graphs, forms, interactive visualizations, code segments, and their outputs, into computational narratives (as shown in Figure 1). These computational narratives enable literate

programming [28] and allow data scientists to effectively create, communicate, and collaborate on their analysis work. The data scientist community has widely adopted notebook systems (e.g., Jupyter Notebook [23] and Jupyter Lab [22]) as their main working environment [49].

However, it is not easy for data scientists to create a computational narrative while they are coding for rapid exploration. Data scientists often need to explore diverse sets of hypotheses and theories [33, 52]. Active exploration of alternatives increases the workload for data scientists to track the history of their experimentation [27]. Thus, documenting for those alternatives will pose more workload to data scientists, sometimes interference with their cognitive process of coding, and is hardly rewarding because many of those alternatives will be discarded in later versions.

Because creating and maintaining a clean computational narrative is often an expensive and tedious process, many computational notebooks shared within open communities are not appropriately documented. For example, Rule et al. examined 1 million open-source computational notebooks from Github and found that one in four lacked any sort of written documentation [57]. In addition, they analyzed a sample of 221 academic computational notebooks, which they considered are higher quality notebooks, and found that academic computational notebooks contained text cells for introduction, describing analytical steps, explaining the reasoning, and discussing results.

Poor documentation hinders the readability and reusability of the notebooks that are shared with other collaborators or even with one's future self [6]. Recently, various groups of researchers have developed a wide range of tools to help data scientists to manage their "messy" computational notebooks. Notably, Lau et al. summarized the design space of computational notebooks which covered an overview for improving explanations in computational notebooks [31]. Many strategies interact with markdown comments. For example, facilitating cell folding could help surface important markdown cells [56]; Kery et al. designed Verdant [25], a lightweight local versioning plugin for Jupyter Lab, that uses algorithmic and visualization techniques for data science workers to better forage their past analysis choices; Woods et al. argued [74] for simpler and richer narratives; Head et al. used code gathering tools to help data scientists trace back to the computational code from an end result [15]; Wenskovitch et al. designed an interactive tool that produced a visual summary of the structure of a computational notebook [73]; Wang et al. proposed capturing the contextual connections between notebook content and discussion messages to help data science teams reflect on their decision-making process [65].

However, despite the wide variety of approaches to helping data scientists manage their notebooks, none of these tools directly aids data scientists in creating new, rich, descriptive content to document their computational notebooks, and to improve the quality of the computational narrative. Recent research works have proposed to use AI solutions to automate the various tasks along with a data science project, such as the model training, model selection, and feature selection, and these technologies are commonly referred as AutoML [34, 67]. The research gap and the AutoML techniques motivate us to design and build an AI system to support data scientists to better document their code and to produce higher qualitative computational narratives.

But what makes up a good computational narrative? Despite the portrait of not-so-good notebooks on Github [57], we need further understanding and role models for well-documented computational narratives. Thus, we decided to first conduct an in-depth analysis of some highly-voted notebooks on Kaggle competition.<sup>1</sup> Kaggle competition provides a platform where organizations post datasets as challenges, and many data scientists submit their notebooks as solutions to a challenge. If a solution has the highest accuracy, it wins the competition. But those winning solutions are often not the most voted ones, as community members voted on the readability, and completeness of the computational narrative.

---

<sup>1</sup><https://www.kaggle.com/>.

## 2.2 Code Documentation in Software Engineering

Documentation plays an important role in software programming. Programmers write comments in their source code to make the code easier for both themselves and others to understand [45]. Writing clear and comprehensive documentation is critical to software development and maintenance [7, 24, 39, 54, 62]. However, writing documentation itself is a time-consuming task. And that is why documentation practices in open source communities are widely perceived to be of low quality, due in part to low levels of intrinsic enjoyment for doing documentation work [14].

To save time in creating documentation, template-based approaches are often used to help developers annotate their source code. For example, tools like JavaDoc<sup>2</sup> and JSDoc<sup>3</sup> allow programmers to annotate their code with tags (e.g., @param, @return) and then automatically generate documentation using these tags. This approach helps programmers create documentation for others and works especially well for documenting API, where method signatures and variable types are important pieces of information and can easily be documented from tags. Although, such methods may not work in the rapid, experimental nature of data science work, because data scientists may be particularly reluctant to create and maintain high-quality documentation of their work. Furthermore, these methods cannot capture other aspects of documentation important in data science, such as how a dataset was constructed, the intent behind an analysis, or a description of why an experiment was successful or not.

Recently, some researchers have put forth proposals for better documenting the specific artifacts involved in a data science workflow, i.e., the dataset and the ML model [3, 13, 16, 41]. Notably, Gebu et al. [13] and Holland et al. [16] proposed both qualitative and quantitative guidelines for documenting a dataset, so that the dataset creators and maintainers can follow these guidelines to document useful information for the data. Similarly, Mitchell et al. [41] and Arnold et al. [3] explored using formulas to document the ML model artifacts, and sharing such formulas with others. These approaches are inline with what is called *provenance*, which refers to tracking what has been done with code and data over time—typically to aid reproducibility of results—using applications such as noWorkflow and YesWorkflow [50], ReproduceMeGit [60], and Provbook [59]. However, this approach focus more on the dataset and the model artifact in the final product of a data science project, and supporting data scientists to create a “factsheet” for these artifacts for the non-technical consumers. Instead, we want to support data scientists to better create the documentation during the process of creating models and data science products, and such documentation, together with the code as a computational narrative, is primarily for other technical users to understand and to reuse.

In addition to the various ways of generating new documentation for code, there is another research line that focuses on improving the usability of documentation, as novice programmers may find it difficult to read and use API documentation [17]. Oney et al. proposed linking interactive documentation and example code in an editor to help novice programmers better understand the external documentation and write code [44]. We believe this approach of linking code with external documentation is a promising way to help data scientists to create more usable documentations, and we will also implement this retrieval-based approach in our system.

## 2.3 GNN-Based Automatic Code Summarization System

Automatic code summarization is a rapidly-expanding research topic in the **Natural Language Processing (NLP)** and ML communities [2, 10, 19, 20, 32]. The automatic code summarization

---

<sup>2</sup>JavaDoc: <https://docs.oracle.com/javase/8/docs/technotes/tools/windows/javadoc.html>.

<sup>3</sup>JSDoc: <https://devdocs.io/jsdoc/>.

task can be considered as a translation task, which takes a code snippet as the input sequence, and “translating” it into a natural language description of the code as an output sequence.

Early work primarily used predefined templates and heuristics to produce code summaries (e.g., [10, 64]). Recent studies have taken the advantage of modern deep neural network architectures to generate the summary for the code (e.g., [2, 19, 21]). Motivated by the language-translation task (e.g., English to French), most of these learning-based approaches are based on the **Neural Machine Translation (NMT)** model architecture [38]. This architecture breaks code into a sequence of input tokens and produces the summarization text as a sequence of output tokens.

However, this sequence-to-sequence approach does not work well in practice because source code is not just a stream of tokens. There is additional semantic information that is lost when processing source code in this way. LeClair et al. proposed improving code summarization through the use of **Graph Neural Networks GNNs** [32]. The GNN model can take in both the code sequence and its **Abstract Syntax Tree AST** structure (refer to Figure 5 for an example) as input to generate summary sentences as output. Their approach achieved better accuracy than the baseline algorithms.

Our work explores neural-network-based automatic code summarization techniques to support document writing in computational notebooks. To our knowledge, there has been little discussion in the HCI community on leveraging automatic code summarization techniques to improve documentation. Furthermore, we suspect that the automation approach alone may not work well in the documentation creation task, as data science is a highly interdisciplinary field that requires various human expertises to explain and interpret. Inspired by prior studies that implement AI systems to work together with human [36, 69], we believe the system will work better if it has both the automated documentation capability and the capability that allows users to directly manipulate the documentation. However, what types of documentation may be better suited for AI to do, and what works should the system leave to human data scientists? This is a design question that requires further exploration of the best practices for creating notebooks. Thus, we start this project with a formative study to fill this research gap.

### 3 FORMATIVE STUDY

In order to build a useful system that can support data scientists to create documentations and to improve their computational narrative’s quality, we first need to explore and understand the characteristics of good documentations in high-quality notebooks. **What does a well-documented computational narrative look like?** We identify “well-documented” computational narratives with ratings from a broader data scientist community (Kaggle), and analyze their characteristics specifically around the documentation. We consider the community voting number is a good indicator to reflect a computational notebook’s quality for our research goal. Based on this premise, we then conduct a formative study to analyze the characteristics of a set of most voted computational narratives, and explore how the data scientists create documentations for these notebooks.

#### 3.1 Data Collection

We collected notebooks from two popular Kaggle competitions—House Price Prediction<sup>4</sup> and Titanic Survival Prediction.<sup>5</sup> We chose these two competitions because they are the most popular competitions (5,280 notebooks submitted for House Price and 6,300 notebooks submitted for Titanic Survival) and because many data science courses use these two competitions as a tutorial for beginners [4, 11].

<sup>4</sup><https://www.kaggle.com/c/house-prices-advanced-regression-techniques>.

<sup>5</sup><https://www.kaggle.com/c/titanic/>.



We collected the top 1% of the submitted notebooks from each competition based on their voting numbers, which resulted in 53 for House Price and 63 for Titanic Survival. We then filtered out the notebooks that were not written in English and the ones that are not relevant to the particular challenge (e.g., a computational notebook as a tutorial on how to save memories can win lots of votes, but it is not a solution to the challenge), which returned 80 valid notebooks for analysis (39 for House Price and 41 for Titanic Survival).

### 3.2 Data Analysis

Five members of the research team conducted an iterative open coding process to analyze the collected notebooks. Differing from [57], where their qualitative coding stopped at the notebook level, our analysis goes deep to the cell granularity: we code each cell's purposes and types of content; and which step (stages) in the data science lifecycle that the cell belongs (e.g., data cleaning or modeling training [68, 77]). Our analysis covered 4,427 code cells and 3,606 markdown cells within the 80 notebooks. Each notebook took around 1 hour to code as we coded the notebook at the cell level.

Each coder independently analyzed the same six notebooks to develop a codebook. After discussing and refining the codebook, they again went back to recode the six notebooks and achieved pairwise inter-rater reliability ranged 0.78–0.95 (Cohen's  $\kappa$ ). After this step, the five coders divided and coded the remaining notebooks.

### 3.3 Results

We found that these 80 well-documented computational notebooks all contain rich documentation. In total, we identified **nine categories for the content** of the markdown cells. In addition, we found the markdown cells covered **four stages and 13 tasks** of the data science workflow [68]. Note that a markdown cell may belong to multiple categories.

*3.3.1 Descriptive Statistics of the Notebook.* We found that on average, each notebook contains 55.3 code cells and 45.1 markdown cells. We replicated the notebook descriptive analysis that Rule et al. used to analyze 1 million computational notebooks on Github [57]. As shown in Figure 2, the left side represents the descriptive visualization of the 80 well-documented computational notebooks from Kaggle (noted as Sample A) and the right side represents the descriptive visualization of the 1 million computational notebooks on Github (noted as Sample B). We found that the Sample A has more total cells per notebook (Median = 95) than Sample B (Median = 18). Sample A has a roughly equal ratio of markdown cells and code cells per notebook, while Sample B is unbalanced with the majority cells being code cells. Notably, Sample A has more total words in markdown cells (Median = 1,728) than Sample B (Median = 218). This result indicates that the 80 well-documented computational notebooks are better documented than general Github notebooks.

*3.3.2 Data Scientists Use Markdown Cells to Document a Broad Range of Topics.* As shown in Table 1, our analysis revealed that markdown cells are mostly used to describe what the adjacent code cell is doing (Process, 58.65%). Second to the Process category, 32.36% markdown cells are used to specify a headline for organizing the notebook into separate functional sections and for navigation purposes (Headline).

Markdown cells can also be used to explain beyond the adjacent code cells. We found that many markdown cells are created to describe the outputs from code execution (Result, 19.19%), to explain results or critical decisions (Reason, 6.30%), or to provide an outline for the readers to know what they are going to do in a list of todo actions (Todo, 5.60%), and/or to recap what has been done so far (Summary, 1.41%).

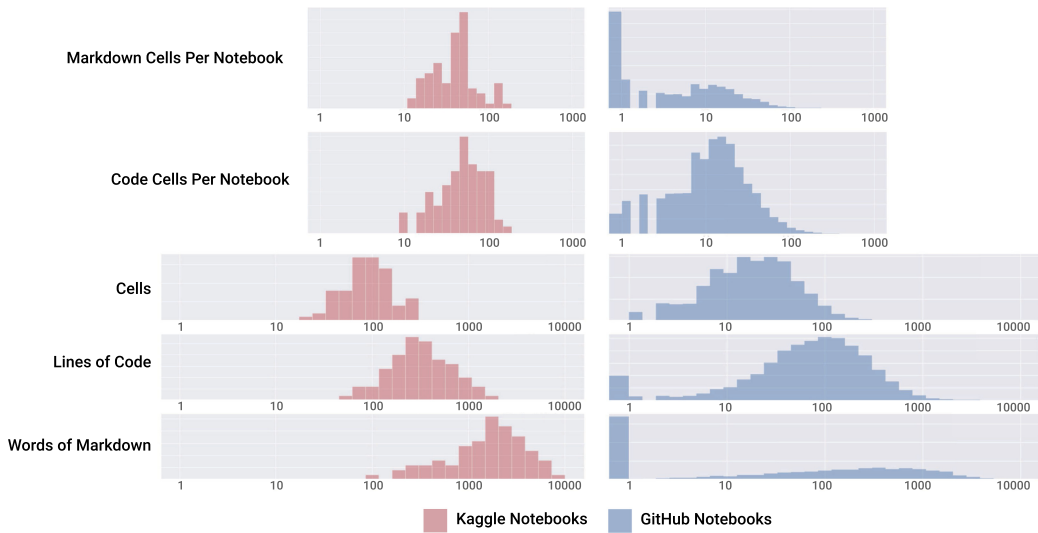


Fig. 2. We replicated the notebook-level descriptive analysis by Rule et al. [57] to the 80 well-documented notebooks on Kaggle. The left side represents the descriptive visualization of the 80 well-documented computational notebooks from Kaggle (noted as Sample A) and the right side represents the descriptive visualization of the 1 million computational notebooks on Github (noted as Sample B). The highly-voted notebooks on Kaggle are better documented compared to the Github notebooks.

We observed that 11.48% markdown cells explain what a general data science concept means, or how a function works (Background Knowledge), while 5.54% markdown cells are connected with external references for readers to further explore the topics (Reference). We believe these are the extra efforts that the notebook owners dedicated, to attract a broader audience, especially beginners in the Kaggle community. In addition, we found that authors approached the story in different styles. For example, some authors want to leave their own signature, and so they spend spaces at the beginning of the notebooks to debrief the project, to add the author’s information, or even to add their mottos (Meta-Information, 3.91%). Some authors prefer to use concise and accurate language to convey important information; while others write documentation in more creative and entertaining ways—for example, making analogies between data science workflow and starting a business.

**3.3.3 Data Science Stages.** We coded markdown cells based on where they belong in the data science workflow [70]. As shown in Table 2, we identified four stages and 13 tasks. The four stages include **environment configuration** (4.50%), **data preparation** and exploration (37.05%), **feature engineering and selection** (10.40%), and **model building and selection** (27.57%). At the finer-grained task level, in particular, notebook authors create more markdown cells for documenting exploratory data analysis tasks (26.62%) and model training tasks (10.45%). The rest of the markdown cells are evenly distributed along with other tasks.

### 3.4 Design Implications

In summary, our analysis of markdown cells in well-documented notebooks suggests that data scientists document various types of content in a notebook, and the distribution of these markdown cells generally follows an order of the data science lifecycle, starting with data cleaning, and ending



Table 1. We Identified Nine Categories Based on the Purpose of Markdown Cells

Category	N	Description	Example
Process	2,115 (58.65%)	The markdown cell describes what the following code cell is doing. This always appears before the relevant code cell.	Transforming Feature X to a new binary variable
Headline	1,167 (32.36%)	The markdown cell contains a headline in markdown syntax. The cell is used for navigation purposes or marking the structure of the notebook. It may be relevant to a nearby code cell.	# Blending Models
Result	692 (19.19%)	The markdown cell explains the output. This type always appears after the relevant code cell.	It turns out there is a long tail of outlying properties...
Background Knowledge	414 (11.48%)	The markdown cell provides a rich content for background knowledge, but may not be relevant to a specific code cell.	Multicollinearity increases the standard errors of the coefficients.
Reason	227 (6.30%)	The markdown cell explains the reasons why certain functions are used or why a task is performed. This may appear before or after the relevant code cell.	We do this manually, because ML models won't be able to reliably tell the differences.
Todo	202 (5.60%)	The markdown cell describes a list of actions for upcoming analysis. This normally is not relevant to a specific code cell.	1. Apply models 2. Get cross validation scores 3. Calculate the mean
Reference	200 (5.55%)	The markdown cell contains an external reference. This is also relevant to the adjacent code cell.	Gradient Boosting Regression Refer [here](https://...)
Meta-Information	141 (3.91%)	The markdown cell contains meta-information such as project overview, author's information, and a link to the data sources. This often is not relevant to a specific code.	The purpose of this notebook is to build a model with Tensorflow.
Summary	51 (1.41%)	The markdown cell summarizes what has been done so far for a section or a series of steps. This often is not relevant to a specific code.	**In summary** By EDA we found a strong impact of features like Age, Embarked.

Note that a markdown cell may belong to multiple categories of contents or none of the categories.

with model building and selection. Based on these findings, we synthesize the following actionable design considerations:

- **The system should support more than one type of documentation generation.** Data scientists benefit from documenting not only the behavior of the code, but also interpreting

Table 2. We Coded Each Markdown Cell to which Data Science Stage (or Task) They Belong

Stage	Total	Task	N
Environment Configuration	162 (4.49%)	Library Loading	33 (0.92%)
		Data Loading	129 (3.58%)
Data Preparation and Exploration	1,336 (37.05%)	Data Preparation	91 (2.52%)
		Exploratory Data Analysis	960 (26.62%)
		Data Cleaning	285 (7.90%)
Feature Engineering and Selection	375 (10.40%)	Feature Engineering	120 (3.32%)
		Feature Transformation	178 (4.94%)
		Feature Selection	77 (2.14%)
Model Building and Selection	994 (27.57%)	Model Building	247 (6.85%)
		Data Sub-Sampling and	61 (1.69%)
		Train-Test Splitting	
		Model Training	377 (10.45%)
		Model Parameter Tuning	81 (2.25%)
		Model Validation and	288 (6.32%)
		Assembling	

We identified four stages with 13 tasks out of the data science lifecycle [68]. Note that a markdown cell may belong to multiple stages or none of the stages.

the output, and explaining rationales. Thus, a good system should be flexible to support more than one type of documentation generation.

- **Some types of documentations are highly related to the adjacent code cell.** We found at least the Process, Result, Reason, and Reference types of documentations are highly related to the adjacent code cell. To automatically generate interpretations of results or rationale for a decision may be hard, as both involve deep human expertise. But, with the latest neural network algorithms, we believe we can build an automation system to generate Process type of documentation, and we can also retrieve Reference for a given code cell.
- **There are certain types of documentations that are irrelevant to the code.** Various types of documentations do not have a relevant code piece upon which the automation algorithm can be trained. Together with the Reason and Result types, the system should also provide a function that the human user can easily switch to the manual creation mode for these types.
- **For different types of documentation, it could be at the top or the bottom of the related code cell.** This design insight is particularly important to the Process, Result, and Reason types of documentation. It may be less preferable to put Result documentation before the code cell, where the result is yet to be rendered. The system should be flexible to render documentation at different relative locations to the code cell.
- **External resources such as Uniform Resource Locators (URLs) and the official API descriptions may also be useful.** Some types of documentation, such as Background Knowledge and Reference, are not easy to be generated with the NN-based models, but they are easy to retrieve from the Internet. So the system should incorporate the capability to fetch relevant web content as candidate documentation.
- **There is an ordinality in markdown cells that is aligned with the data science project's lifecycle.** The system should consider that Library Loading types of cells are often at the beginning section of the notebook, and the Model Training type of content may be more likely to appear near the end of the notebook. In our system prototype, though, we did not take this design consideration into account, it will be our future work.

- **The notebook would be nice to have documentation with a problem overview at the beginning and a summary at the end.** We considered this design implication not in the system design, but our evaluation study design. For the two barebone notebooks, we used in the experiment, we always provide a problem overview as a markdown cell at the top of the notebook.

## 4 THE THEMISTO SYSTEM: DESIGN AND IMPLEMENTATION

Based on findings from the formative study and design insights from related works, we design and implement Themisto, an automatic documentation generation system that supports data scientists to write better-documented computational narratives. In this section, we present the system architecture, the **user interface (UI)** design, and the core technical capability of generating documentation.

### 4.1 System Architecture

The Themisto system has two components: the client-side UI is implemented as a Jupyter Notebook plugin using TypeScript code, and the server-side backend is implemented as a server using Python and Flask.

The client-side program is responsible to render UI, and also to monitor the user actions on the notebook to edit in code cells. When the user's cursor is focused on a code cell, the UI will send the current code cell content to the server-side program through **Hypertext Transfer Protocol HTTP** requests.

The server-side program takes the code content and generates documentation using both the deep-learning-based approach and the query-based approach. For the deep-learning-based approach, the server-side program first tokenizes the code content and generates the AST. It then generates the prediction with the pre-trained model. For the query-based approach, the server-side program matches the curated API calls with the code snippets and returns the pre-collected descriptions. For the prompt-based approach, the server-side program sends different prompts (e.g., for interpreting results or for explaining reason) base on the output type of the code cell.

### 4.2 User Interface Design

Figure 3 shows the UI of Themisto as a Jupyter Notebook plugin. Each time the user changes their focus on a code cell, as they may be inspecting or working on the cell, the plugin is triggered. The plugin sends the user-focused code cell's content to the backend. Using this content, the backend generates a code summarization using the model and retrieves a piece of documentation from the **API** webpage. When such a documentation generation process is done, the generated documentation is sent from the server-side to the frontend, and a light bulb icon appears next to the code cell, indicating that there are recommended markdown cells for the selected code cell (as shown in Figure 3(A)).

When a user clicks on the light bulb icon which appears next to any selected code cells, Themisto render all the three options in the dropdown menu: (1) a deep-learning-based approach to generate documentation for source code (Figure 3(B)); (2) a query-based approach to retrieve the online **API** documentation for source code (Figure 3(C)); and (3) a user prompt approach to nudge users to write more documentation (Figure 3(D)). If the user likes one of these three candidates, they can simply click on one of them, and the selected documentation candidate will be inserted into above the code cell (if it is the Process, Reference, or Reason type), or below it (if it is the Result type).

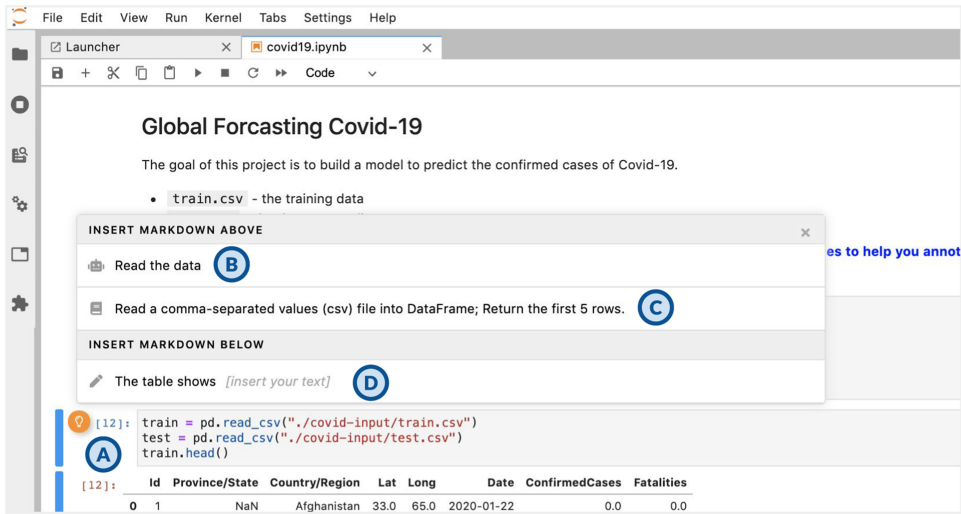


Fig. 3. The Themisto UI is implemented as a Jupyter Notebook plugin: (A) When the recommended documentation is ready, a lightbulb icon shows up to the left of the currently-focused code cell. (B–D) shows the three options in the dropdown menu generated by Themisto, (B) A documentation candidate generated for the code with a deep-learning model, (C) A documentation candidate retrieved from the online API documentation for the source code, and (D) A prompt message that nudges users to write documentation on a given topic.

### 4.3 Three Approaches for Documentation Generation

In this subsection, we describe the rationale and implementation detail of the three different approaches for documentation generation (Figure 4):

- Our formative study suggests that the system should be able to **generate multiple types of documentation** (e.g., Process, Result, Background Knowledge, Reason, and Reference).
- Some types of documentation can be **directly derived from the code**, thus the automated approaches can help. The Process type of documentation directly describes the coding process, and existing ML literature suggest that the deep-learning-based approach is most suitable for generating it; The Reference type does not need a learning-based approach, it can be achieved with a traditional query-based approach, which locates and retrieves the most relevant online documentation as candidates;
- Some other types of documentation (e.g., Education, Result, and Reason) are **not directly related to the code**, thus the fully automated approaches are not capable of generating such contents. We design the prompt-based approach for users to complete the generation process.

**4.3.1 Deep-Learning-Based Approach.** We trained a deep learning model<sup>6</sup> using the GNN architecture based on LeClair et al. [32]. These GNN models can take both the source code's structure (extracted as AST) and the source code's content as input. Thus, it outperforms the traditional sequence-to-sequence model architectures, which only takes the source code's content as an input sequence, in source code summarization tasks for Python code.<sup>7</sup> We did not consider T5, BeT

<sup>6</sup>We release a larger dataset and a refined version of the model in a separate paper [35].

<sup>7</sup>All the collected data science notebooks are in Python.

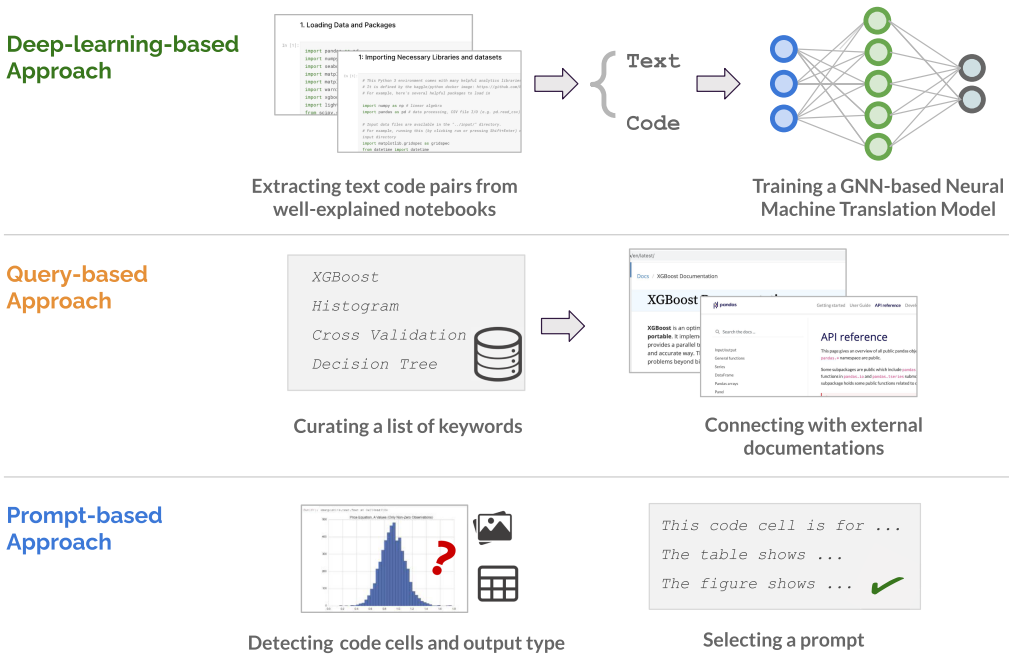


Fig. 4. An illustration of the three different approaches for documentation generation in Themisto.

or GPT-3 architectures as these models can take minutes to make one inference (i.e., generate one summary) even with a cluster of GPUs (costing thousands of dollars per hour), whereas our GNN-based model can make an inference within a second with one GPU.

In order to fine-tune the model, we constructed a training dataset for our particular context. We collected the top 10% highly-voted notebooks from two popular Kaggle competitions—House Price Prediction and Titanic Survival Prediction (N = 1,158). For each of the notebooks, we first extracted code cells and the markdown cells adjacent above as a pair of input and output (similar to the data collection approach in [1]). If there is an inline comment in the first line of the code cell, we replaced the output of the pair using the inline comment. In total, our dataset has 5,912 pairs of code and its corresponding documentation. Following the best practice of model training, we split the dataset into training, testing, and validation subsets with an 8 to 1 to 1 ratio.

Before feeding data into the training process, we have a three-step pre-processing stage, as illustrated in Figure 5. Step 1 removes the style decoration, formats, and special characters that are not in Python grammar (e.g., Notebook Magics). We also generate an Abstract Syntax Tree (AST) for the source code input in step 2 with Python AST library.<sup>8</sup> The AST result is equivalent to the source code but with more contextual and relational information. In step 3, we tokenize source code to a sequence of tokens with an input dictionary, and parse the AST nodes as a sequence of tokens with the same input dictionary. We parse the relationship between AST nodes as a matrix of edges. Finally, we tokenize the output documentation as a sequence of tokens with a separate output dictionary. After this process, all the tokens are transformed into an array of word embeddings—vectors of real numbers. We use these data to train the network for 100 epochs, 30 batch sizes, and 15 early stop points on a two Tesla V100 GPU cluster. Out of all the epochs, we selected the model with the highest validation accuracy score.

<sup>8</sup><https://docs.python.org/3/library/ast.html>.

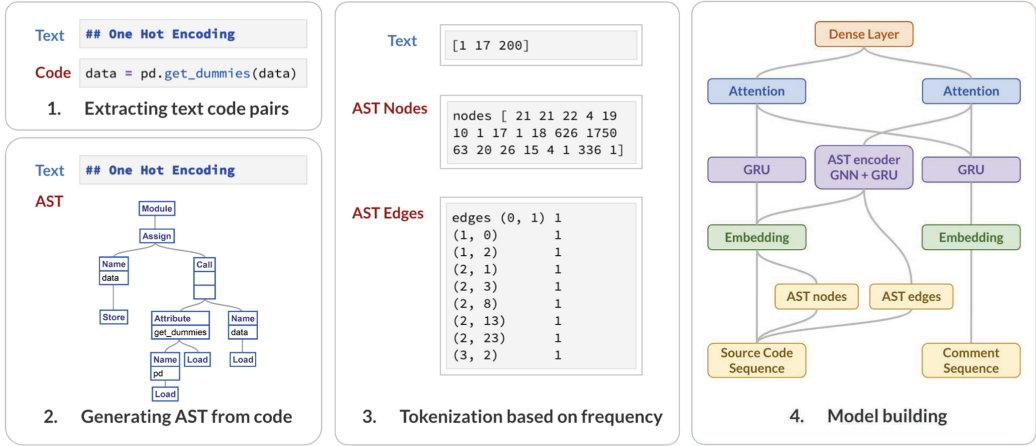


Fig. 5. A code summarization model for the deep-learning-based documentation generation approach via GNN. There are three steps of data pre-processing (1) We first extract text code pairs from existing notebooks. (2) We generate AST from code. (3) We tokenized each word and translated them into embeddings. And (4), the GNN model architecture.

Table 3. Example Output from the Model

Example	Code Cell	Output From the Model
Example A	<code>train = pd.read_csv('./house-input/train.csv')</code> <code>test = pd.read_csv('./house-input/test.csv')</code>	Read the data
Example B	<code>all_data = pd.get_dummies(all_data)</code>	Convert all the data
Example C	<code>pred = Tree_model.predict(x_test)</code> <code>pred = pd.DataFrame(pred)</code> <code>pred.columns = ["ConfirmedCases_prediction"]</code>	Predicate to use a predict function for tests

(Example A) The generated text well describes the code. (Example B) The generated text vaguely describes the code. (Example C) The generated text is poorly readable, but still captures the keywords of the descriptions.

To evaluate our model's performance against baseline models, we conducted both quantitative and qualitative evaluations, as suggested by [53]. For the automated quantitative evaluation, we use BLEU scores [46] as the model performance metric. BLEU scores are commonly used in the source code summarization tasks. It evaluates the word similarity between the generated text and the ground truth text. We selected and trained Code2Seq model [2] and Graph2Seq model [75] with the same data split.

Our model achieves 11.41 (BLEU-a), which outperforms the baseline models Code2Seq (BLEU-a = 9.61) and Graph2Seq (BLEU-a = 11.05). These scores suggest that the data science documentation task is more difficult than the benchmark code summarization tasks in the software engineering field. For example, in data science, a notebook code cell can contain multiple code snippets and functions.

In addition to the automated quantitative evaluation, we also conduct a qualitative analysis of the generated documentation pieces. We found that despite the word-to-word similarity score is low, the general quality of the content is reasonable and satisfying for building a prototype system. As an illustration, we provide three examples with both input and model generated outputs, as shown in Table 3. In the Appendix, we provide full code cells and model-generated outputs for the two experimental notebooks that we used in the user study.



**4.3.2 Query-Based Approach.** Our formative study indicates that the well-documented Kaggle notebooks often have the description of frequently-used data science code functions for educational purposes. And sometimes data scientists directly paste in a link or a reference to the external **API** documentation for a code function. Thus, we implement a query-based approach that curates a list of **APIs** from commonly used data science packages, and the short descriptions from external documentation sites. In our system, we only cover Pandas,<sup>9</sup> Numpy,<sup>10</sup> and Scikit-learn<sup>11</sup> these three libraries as a starting point to explore this approach. We argue that it can be easily expanded to include other packages in the future. We collected both the **API** names and the short descriptions by building a crawling script with Python. When users trigger this query-based approach for a code cell, Themisto matches the **API** names with the code snippets and concatenate all the corresponding descriptions.

**4.3.3 Prompt-Based Approach.** Lastly, the system also provides a prompt-based approach that allows users to manually create the documentation. Because our formative study found that a well-documented notebook not only documents the process of the code, but also interprets the output, and explains rationales. These types of documentation are hard to generate with automated solutions. To achieve it, we implement a prompt-based approach. It detects whether the code cell has a cell output or not: if the cell outputs a result, Themisto assumes that the user is more likely to add interpretation for the output result, thus the corresponding prompt will be inserted below the code cell. Otherwise, the system assumes the user may want to insert a reason or some educational type of documentations, thus it changes its prompt message.

## 5 USER EVALUATION OF THEMISTO

To evaluate the usability of Themisto and its effectiveness in supporting data scientists to create documentation in notebooks, we conducted a within-subject controlled experiment with 24 data scientists. The task is to add documentation to the given notebook. And each participant is asked to finish two sessions, one with the Themisto support and one without its support. The evaluation aims at understanding (1) how well Themisto can facilitate documenting notebooks and (2) how data scientists perceive the three approaches that are used by Themisto for generating documentation.

### 5.1 Participants

We recruited 24 data science professionals as our evaluation participants in a multinational IT company. We used a snowball sampling approach to recruit participants, where we sent recruitment messages to friends and colleagues, various internal mailing lists, and Slack channels. We then asked participants to refer their friends and colleagues. Our recruitment criteria are that the participant must have had experience in data science projects and they are familiar with Python and Jupyter Notebook environment. As shown in Table 4, participants reported a diverse job role backgrounds, including expert data scientists (N = 8), novice data scientists (N = 9), **AI Operators (AIOP)** or ML engineers (N = 2), subject matter experts (N = 1), and application developer (N = 4).

### 5.2 Study Protocol

We conducted a within-subject controlled experiment with 24 data scientist participants. Their task was to add documentation to a given draft notebook, which only has code and no documentation

<sup>9</sup><https://pandas.pydata.org/docs/reference/index.html>.

<sup>10</sup><https://numpy.org/doc/stable/reference/>.

<sup>11</sup><https://scikit-learn.org/stable/modules/classes.html>.

Table 4. Demographics of Participants

PID	Gender	Job Role	Work Experience in Data Science
P1	M	Expert Data Scientist	5–10 years
P2	M	Application Developer	3–5 years
P3	M	Novice Data Scientist	less than 3 years
P4	M	Novice Data Scientist	0 year (just start learning data science)
P5	M	AI Operator or ML Engineer	3–5 years
P6	M	Novice Data Scientist	less than 3 years
P7	M	Application Developer	3–5 years
P8	M	Novice Data Scientist	less than 3 years
P9	F	Expert Data Scientist	3–5 years
P10	M	Expert Data Scientist	5–10 years
P11	M	Expert Data Scientist	more than 10 years
P12	F	Novice Data Scientist	3–5 years
P13	F	Expert Data Scientist	5–10 years
P14	M	Novice Data Scientist	0 year (just start learning data science)
P15	M	Expert Data Scientist	more than 10 years
P16	M	AI Operator or ML Engineer	3–5 years
P17	M	Subject Matter Expert	3–5 years
P18	M	Expert Data Scientist	more than 10 years
P19	F	Application Developer	3–5 years
P20	F	Expert Data Scientist	3–5 years
P21	M	Novice Data Scientist	3–5 years
P22	M	Novice Data Scientist	less than 3 years
P23	M	Application Developer	less than 3 years
P24	M	Novice Data Scientist	less than 3 years

at all. The participants were told that they were adding documentation for the purpose of sharing those documented notebooks as tutorials for data science students who just started learning data science. Each participant is asked to finish two sessions, one with the Themisto support (Experiment condition) and one without its support (Control Condition). We prepared two draft notebooks, one for each session, shown in the Appendix. The two experiment notebooks are adapted from winning notebooks from two Kaggle challenges, which are not included in the model training dataset: (1) House Price Prediction;<sup>12</sup> (2) COVID Case Prediction.<sup>13</sup> The two notebooks have the same length (9 code cells) and a similar level of difficulty. Although the two notebooks are simplified versions from winning notebooks, they cover most stages in data science lifecycles. In addition, the length of the notebooks falls into the middle range of the notebook length distribution on the GitHub corpus (as refer to Figure 2). To counterbalance the order effect, we randomized the order of the control condition and the experiment condition for each participant, so some participants encountered Themisto in their first session, some others experienced it in their second session.

Each participant was given up to 12 minutes (720 seconds) to finish one session. We conducted three pilot run sessions, and all the three pilot participants were able to finish a single task within 10 minutes, with or without the support of Themisto. Before the experiment condition session,

<sup>12</sup><https://www.kaggle.com/c/house-prices-advanced-regression-techniques>.

<sup>13</sup><https://www.kaggle.com/c/covid19-global-forecasting-week-1>.

we gave the participant a 1-minute quick demo on the functionality of Themisto. All study sessions were conducted remotely via a teleconferencing tool. We asked the participants to share their screens and we video recorded the entire session with their permission. After finishing both sessions, we conducted a post-experiment semi-structured interview session to ask about their experience and feedback. We had a few pre-defined questions such as “How do you compare the experience of the documenting task with or without the support of Themisto?” or “Did you notice the multiple candidates in the dropdown menu? Which one do you like the most?” In addition, participants were encouraged to tell their stories and experience outside these structured questions. The interview sections of the video recordings were transcribed into text.

### 5.3 Data Collection and Measurements

We have three data sources: the observational notes and video recording for each session ( $N = 48$ ), the final notebook artifact out of each session ( $N = 48$ ), and the post-task questionnaire and interview transcripts ( $N = 24$ ).

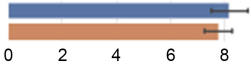
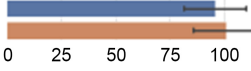
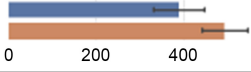
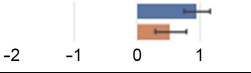
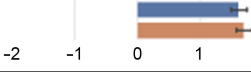

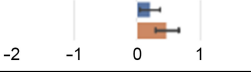
Our first group of measurements are from coding participants’ behavioral data from the session recordings. In particular, we counted *the task completion time* (in seconds) for all sessions. Then, for experiment condition only, we also counted the following: how many times a participant clicked on the light bulb icon to check for suggestions (*code cells checked for suggestions*); how many times a participant directly used the generated documentation (*markdown cells created by Themisto*); how many times a participant ignored the generated recommendations and manually created the documentation (*markdown cells created by human*); and how many times a markdown cell is co-created by human and Themisto (*markdown cells co-created by human and Themisto*). The result of this analysis is reported in Tables 5 and 6.

Secondly, to evaluate the quality of the final notebook artifact, we define our second group of measurements by counting: the *number of added markdown cells*, and the *number of added words* as these two are indicators of the quantity and effort each participant spent in a notebook. Also, we asked the participants to give a self-reported satisfaction score to each of the two documented notebooks. We considered that score ( $-2$  to  $2$ ) as a self-reported subject feeling of the notebook satisfaction. In parallel, we asked two experts to rate the notebook-level quality ( $N = 48$ ) with a 3-dimensional rubric (based on [12]) to evaluate the documentation’s *readability*, *accuracy*, and *informativeness* in a notebook. We considered these three scores ( $-2$  to  $2$ ) as an objective quality of the notebook. Readability concerns whether the documentation is in readable English grammar and words, while accuracy concerns how the documentation matches the code content, and informativeness evaluates whether the documentation covers more information units. Two experts iteratively discussed and evaluated the notebooks until the independent ratings achieved an agreement ( $\alpha = 0.76$ , Krippendorff’s alpha). The result of this analysis is reported in Table 5.

For the experiment session only, we conducted a cell-level expert rating ( $N = 194$ ) using the same approach as in notebook-level expert rating. Two experts iteratively discussed and evaluated the notebooks until the independent ratings achieved an agreement ( $\alpha = 0.88$ , Krippendorff’s alpha). The result of this analysis is reported in Table 6. In addition, we asked the participants to finish a post-experiment survey (5-point Likert Scale,  $-2$  as strongly disagree and  $2$  as strongly agree, Figure 6) to collect their feedback specific on the system’s *usability*, *accuracy*, *trust*, *satisfaction*, and *adoption propensity* (based on [72]).

Lastly, for the interview transcripts, four researchers of this research project conduct an iterative open coding method to get the code, theme, and representative quotes as the third group of data. They each independently coded a subset of interview transcripts, and discussed the codes and themes together. After the discussion, they when back and reiterated the coding practice to apply the codes and themes to their assigned notebooks. Some examples of the identified themes are:

Table 5. Performance Data in Two Conditions (M: mean, SD: Standard Deviation): the Task Completion Time (Seconds), Participants' Satisfaction with the Final Notebook (from -2 to 2), Graded Notebook Quality, Number of Markdown Cells, and Number of Words

	Condition	M	SD	
Number of Added Markdown Cells	Experiment	8.04	2.40	
	Control	7.79	1.91	
Number of Added Words	Experiment	95.75	50.56	
	Control	100.92	53.27	
**Task Completion Time (secs)	Experiment	391.12	200.15	
	Control	494.75	184.28	
*Satisfaction with the Final Notebook (-2 to 2)	Experiment	0.96	0.69	
	Control	0.54	0.83	
Expert Rating: Accuracy (-2 to 2)	Experiment	1.60	0.47	
	Control	1.62	0.52	
Expert Rating: Readability (-2 to 2)	Experiment	0.65	0.83	
	Control	0.90	0.57	
Expert Rating: Informativeness (-2 to 2)	Experiment	0.67	0.64	
	Control	0.75	0.63	

In particular, participants spent less time to complete the task in the experimental condition than the control condition ( $p = .001$ ); participants were more satisfied with the final notebook in the experimental condition than the control condition ( $p = .04$ ).

pros and cons of Themisto; preference of the three document generation approaches; future adoption, and suggestions for design improvement. We will report the qualitative results as supporting materials together with reporting the quantitative results.

## 5.4 Results

In this section, we present the user study results on: how Themisto improved participants' performance on the task, how participants perceived the documentation generation methods in Themisto, and how participants described the practical applicability of Themisto.

**5.4.1 Themisto Supports Participants to Easily Add Documentations to a Notebook.** Our experiment revealed that Themisto improved participants' performance on the task by reducing task completion time and improving the satisfaction with the final notebooks.

We performed a two-way repeated measures ANOVA to examine the effect of the two notebooks and the two conditions (with or without Themisto) on task completion time. As shown in Table 5, participants spent significantly **less time** ( $p < .001$ ) to complete the task using Themisto in the experiment condition ( $M(SD) = 391.12 (200.15)$ ) than in the control condition ( $M(SD) = 494.75 (184.28)$ ). In addition, there was not a statistically significant effect of notebooks on task completion time, nor a statistically significant interaction between the effects of notebooks and conditions on completion time.

The post-experiment survey result supported our findings. Most participants believed it was easier to accomplish the task with Themisto's help (22 out of 24 rated agree or higher), as shown

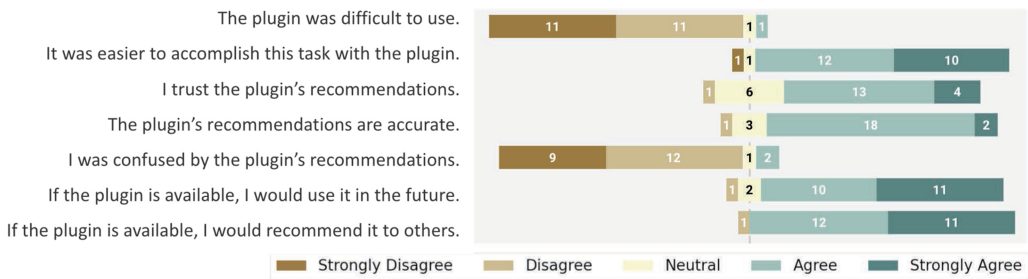


Fig. 6. Results of the post-task questionnaire. Note that the disagrees are not from the same participant.

in Figure 6. And the Themisto generated recommendation was accurate (20 out of 24 rated agree or higher).

Looking into the qualitative interview data, we can find some potential explanations for why participants believed so. Participants reported that Themisto provided them something to begin with, thus it was easier than starting from scratch: “*The plugin makes it easy to just pick it and have something simple. And then I got a couple of times where I went back and said, ‘Oh let me add a few more words.’*” (P21).

#### 5.4.2 Co-Creation Yields Longer Documentation and Improves Accuracy and Readability.

Through coding the video recordings for only the experiment-condition sessions, we were able to examine the following questions: while the Themisto was available, how did the participants use it? Did they check the recommendations it generated? Did they actually use those recommendations in their documentations added into notebooks?

As shown in Table 6, we found that while Themisto is available, for 86.11% of code cells, participants checked the recommended documentation by clicking on the light bulb icon to show the dropdown menu. Then, 46.90% of the created markdown cells were directly adopted from Themisto’s recommendation; while 11.86% of the created markdown cells were manually crafted by humans alone. **The most interesting finding is that 41.24% markdown cells were co-created by Themisto and human participants together:** Themisto suggests a markdown cell, human participants take it, and modify on top of it. This result suggested that most participants used Themisto in the creation of documentation, and some of them formed a small collaboration between humans and the AI. This finding inspires us to further explore how participants co-create the documentation with Themisto [43]. By looking at the log data, we discover several editing patterns. For example, many participants added supplemental details (e.g., expanding the steps into substeps) to Themisto’s suggested documentation. Participants also added stylistic edits, including modifying document hierarchies, polishing sentences, and changing conversational tones.

In order to explore the differences among documentation created by three methods (created by Themisto only, co-created by human and Themisto, created by human only), we conducted a cell-level expert rating ( $N = 194$ ) along the dimension of accuracy, readability, and informativeness. We also calculated the word count of the documentation length. We performed a one-way ANOVA to examine the differences among the three groups. As shown in Table 6, markdown cells that are co-created by humans and Themisto have significantly more word count ( $M(SD) = 15.45 (10.97)$ ) than markdown cells that are manually written by humans alone ( $M(SD) = 10.26 (7.41)$ ) and the markdown cells that are directly adopted from Themisto’s recommendation ( $M(SD) = 8.88 (7.14)$ ), with  $F = 11.83, p < 0.001$ . Markdown cells co-created by humans and Themisto also yield better results in terms of accuracy ( $F = 9.43, p < 0.001$ ) and readability ( $F = 3.28, p = 0.04$ ), while for

Table 6. Usage Data of the Plugin in Experimental Condition

	<i>N</i>	%	Word Count	Accuracy	Readability	Informativeness
Code Cells Checked for Suggestions	186	86.11	–	–	–	–
Markdown Cells Created by Themisto Only	91	46.90	8.88 (7.14)	1.36 (0.54)	1.94 (0.23)	1.34 (0.56)
Markdown Cells Co-created by Humans and Themisto	80	41.24	15.45 (10.97)	1.68 (0.47)	1.96 (0.17)	1.51 (0.55)
Markdown Cells Created by Humans Only	23	11.86	10.26 (7.41)	1.28 (0.69)	1.83 (0.36)	1.35 (0.65)

The results indicate that participants used the plugin for recommended documentation on most code cells (86.11%). For markdown cells in the final notebooks, 46.90% were directly adopted from the plugin’s recommendation, while 41.24% were modified from the plugin’s recommendation and 11.86% were created by participants from scratch.

informativeness, there are no significant differences across the three groups. Our posthoc analysis suggested that no significant differences were found between markdown cells created by Themisto and markdown cells created by humans only along all dimensions (including word count, accuracy, readability, and informativeness).

*5.4.3 Themisto Increases Participant’s Satisfaction, while Maintaining a Similar Quality of the Final Notebook.* The post-task questionnaire revealed that participants were **more satisfied** with the final notebook after using Themisto in the experiment condition than in the control condition ( $p = .04$ ) (Table 5). The interview results also supported this finding. P14 believed that Themisto helped with the wording: “*Sometimes I knew what the cells were doing but I did not know how to put things in a really good sentence for others.*”

Themisto also motivates participants to document the analysis details. Although we did not see a difference in the number of markdown cells created in two conditions or the number of words in total, Themisto helps them overcome the procrastination of writing documentation and reminds them to document things that they might ignore.

I think I definitely overlooked some details when I was commenting without the tool, because I just made the assumption that people should know from the code... To be honest, I do not usually follow a good coding practice. My notebooks are really messy and I am the only person who can understand it. I feel sorry for anybody else that has to see it. (P19)

Moreover, participants believed that Themisto can help them form a better documenting practice in the long term: “*It is very useful to remind me to always put some documentation in a timely manner.*” (P13).

The two experts’ gradings for the notebook quality suggest that there was not a significant difference for the three dimensions of the quality rubric (**accuracy, readability, and completeness**). In the post-task interview, participants mentioned that the accuracy of the generated recommendations plays a role in participant’s experience: “*My experience with the plugin is definitely better. For the most part, the suggestions are pretty accurate. Although sometimes I did make a few minor changes like rearranging the text.*” (P5). Some participants also mentioned that they needed to edit the format of the generated document to fit their context. We believe that while Themisto offers convenience to improve the data scientists’ productivity and saves their time, it may not provide the same level of readability as those notebooks well articulated by humans. Thus, data scientists may want to further revise the formatting and wording of the Themisto generated documentation.

In summary, our experiment indicated that Themisto improves participants’ productivity for creating documentation. It also increases their perceived satisfaction with the final notebook, compared to the notebooks written by participants themselves.

*5.4.4 The Three Approaches of Generating Documentation are Suitable for Different Scenarios.* In this section, we have an in-depth analysis of how participants perceived the three different



approaches that Themisto implemented to generate documentations: the deep-learning-based approach, the query-based approach, and the prompt-based approach. In the post-experiment interview, we explained how Themisto generated the documentation with these three approaches, and asked participants if they like or dislike one particular approach.

Participants reported that they felt the deep-learning-based approach provided concise and general descriptions of the analysis process: *“I think the AI suggestion gives me an overview. It is short, and has some useful keywords.”* (P12).

Participants also suggested that the deep-learning-based approach sometimes generated inaccurate or very vague documentation: *“The first one gives me a very short summary, though it didn’t always say what the cell is doing.”* (P1). But the deep-learning-based approach is still perceived as useful. As it is short and with only a couple of keywords, many participants believe it may be more suitable for some quick and simple documentation task, or for the analyst audience who can understand these short keywords.

In terms of the query-based approach, participants believed that the documentation generated from this approach contains longer and more descriptive information. This approach is further perceived to be more suitable for educational purposes: *“This one gives you really good information. For some specific methods or calls, you don’t have to come up with a high-level summary for others and you can directly use it.”* (P14).

Participants also acknowledged that such a query-based approach may not work for some scenarios. For example, participants found that the query-based approach was not useful for summarizing the very fundamental level data manipulations, as there was no core **API** method in it. Some participants mentioned that the usefulness of this query-based approach depends on the audience.

The [deep-learning-based approach] was really useful. The [query-based approach]... it depends on the audience. It is much more appropriate for a novice programmer. (P18)

We observed in the video recordings that participants rarely used the prompt-based approach in the session. The interview data confirmed our speculation. Some participants said that they liked the idea of user prompts, but they did not use it because the deep-learning-based approach and the query-based approach already gave them the actual content. Other participants pointed out that the prompts were not intelligent enough, so they did not use it: *“It always asks the same thing and I just ignored the prompts.”* (P18).

Participants suggested that the prompts could be designed to better fit the context.

Perhaps the system can infer what the code cell was doing [from the deep-learning-based approach], and show prompts accordingly. Like if I delete a data point from the dataset, there is a prompt asking why I considered it as an outlier or something. (P5)

Last but not the least, many participants preferred a hybrid approach to combine the deep-learning-based approach and the query-based approach. For example, P12 mentioned,

The first one (deep-learning-based) tells me what the code cell is doing in general and the second one (query-based) tells me the details of the function. I would go with a hybrid approach. (P12)

**5.4.5 Will Participants Use Themisto in Their Future Data Science Project?** Most participants indicated that they would like to use Themisto in the future when answering the survey question as shown in Figure 6. The interview data provides more detail and evidence to elaborate on this

result. Participants suggested various scenarios in which the Themisto could be useful in their future work, such as they need to add documentation during the exploration process for future selves, or they need to document a notebook in a post-hoc way for sharing it with collaborators, or they need to mentor a team member who is a novice data scientist, or they need to refactor an ill-documented notebook written by others:

When I am doing data analysis, I tend to write the code first because there is a flow in my head of what I need to do. And then I will go back afterward to use the plugin and add the comments needed. I will definitely do that before sharing that file or handing it over to others. (P12)

There was one participant who did not think Themisto could fit into his workflow: “*I always write documentation before writing code. Maybe Themisto does not work for people like me.*” (P3)

In our experiment, we provided the scenario as they were documenting the notebook as a tutorial for some data science students. In the interview, we asked how participants would document a notebook differently if they were created documentation for the notebooks for non-technical domain experts audience. Some participants suggested that computational notebooks may not be a good medium to present the analysis to non-technical domain experts. They would prefer to curate all the textual annotations in a standalone report or slide decks. Some others believed that the notebook could work as the medium but they would change the documentation by using less technical terminology, adding more details on topics that the non-technical domain experts would be interested in (e.g., how data is collected, potential bias in the analysis).

**5.4.6 Participants Suggest Various Design Implications for Automated Code Documentation.** In the interview, participants provided various design suggestions to improve Themisto and to design future technologies that can support data scientists to document the notebook.

Participants expected Themisto to have more functionalities than simply generating documentation for the code. For example, P13 proposed maybe Themisto can also create a description to document the changes of versions and the editing histories from different team members. P3 and P4 believed that the automatic generation of Reason is very much needed for explaining decisions such as why selecting a particular algorithm. P19 wanted the system to automatically add explanations to the execution errors. Participants also mentioned that Themisto should add more varieties into the generated content’s formatting. They would like to see suggested documentation with a better presentation.

And lastly, some participants suggested that maybe such a documentation generation system can take consideration of the purpose of the notebook, the domain-specific terminology, or the individuals’ habits for writing documentation.

**5.4.7 Summary of the Results.** In summary, our study found that Themisto can support data scientists in generating documentation by significantly reducing their time spent on the task, and improving the perceived satisfaction level of the final notebook. When Themisto is available, participants are very likely to check the generated documentation as a reference. Many of them directly used the generated documentation, a few of them still prefer to manually type the documentation, while many of them adopted a human-AI co-creation approach that they used the AI-generated one as a baseline and keep improving on top of it. Participants perceived the documentation generated by the deep-learning-based approach as a short and concise overview, the documentation generated by the query-based approach as descriptive and useful for educational purposes, but they rarely used the prompt-based approach. Overall, participants enjoyed Themisto and would like to use it in the future for various documenting purposes.

## 6 DISCUSSION

### 6.1 The Documentation Practices in Data Science is Different from in Software Engineering

The practice of documentation in data science has both overlaps and strong contrasts in relation to the ones in software engineering in many facets. Software engineers write inline comments in their work-in-progress code to help collaborators understand the behavior of the code without the burden of going through thousands of lines of code; they document changes of their code for better version management and improving awareness of their collaborators; when others need to build upon their work, they write formal documentation and Readme files to describe how to use functions and API in their packages or services [37, 58, 77]. Data scientists write computational narratives as a practice of literate programming [26, 48], and as a way to think and explore alternatives. Thus, notebooks often have orphan code cells or out-of-order code snippets, which leads to lower reusability of the notebook and further highlights the importance of documentation in the notebook. As we found in our formative study, well-documented notebooks explain more than the behavior of the code. Notebooks cover various topics including describing and interpreting the output of the code, explaining reasons for choosing certain algorithms or models, educating the audience from different levels of expertise, and so on.

Thus, many interventions and lessons learned about documentation in software engineering may not apply in the data science context. For example, how can we evaluate the quality of the documentation? Software documentation can be assessed based on attributes like completeness, organization, relevance of content, readability, and accuracy [12]. Our experiment found that the quality scores assessed by these rubrics does not reflect users' satisfaction with the final notebooks. Despite many people's efforts to creating a standard documentation practice [29, 55], it remains questionable whether there is a one-size-fits-all solution. For example, Rule et al. [55] suggested 10 rules for writing and sharing computational analysis in Jupyter notebooks. The first rule they proposed is to tell a story to the audience. However, this description is very general as people may approach storytelling differently. As we observed in Kaggle notebooks, some notebook authors prefer to use concise and accurate language while others use more colloquial and creative language. These creative notebooks stand out and receive many votes and compliments from the Kaggle community. As we recognize documentation in data science as a fluid activity, traditional template-based approaches to aid documentation writing may not work in data science because they cannot capture a broader aspect of documentation, and limit the expressiveness of storytelling.

We argue that future work should recognize the difference between data science and software engineering, and tailor the documentation experience for data scientists. For example, Callisto [65] harnessed the fact that data scientists engage in synchronous work and discussion, and used contextual links between discussion messages and notebook content to aid the explanation of notebooks.

### 6.2 Human-AI Collaboration in Code Documentation in Data Science

We argue that AI-assisted code documentation process can be viewed as a co-creative process in which machine learning fits into the human workflow and collaborate together to create documents in a notebook. The notion of a "partnership relationship" between human data scientists and AI has been discussed by Wang et al. [70], and is part of a larger research discussion by many others (e.g., [5, 40, 61]). We consider this partnership as broadly defined where an AI system does not need an avatar or a conversational interface, but this AI system should be designed to fit into the existing human workflow and assist some parts of the human task to improve the quality or productivity. Human-AI collaboration, as opposed to human-AI competition (portrayed by

AlphaGo or DeepBlue), should be the ultimate goal of human-AI interaction research. Various human-centered AI design principles (such as human-in-the-loop) are means to get to this end goal. Our study demonstrated another means to achieve human-AI collaboration, where we combined the fully automated neural network approach and the less advanced rule-based or prompt-based approaches. This design is to acknowledge the limitation of today’s neural-network modeling. Our result showed that the combined human + AI effort produced a satisfied level of quality at a *more rapid pace* than what human or AI could achieve alone.

We also observed the user interaction pattern in which the AI creates “first draft” of the documentation, followed by human review and editing, which resulted in a final artifact that not only met the bar for quality, but exceeded it for the level of satisfaction. Participants were happier with their code documentation when they were assisted by the AI system to create it, rather than when they worked alone. Thus, we conclude that the benefits of having an AI partner in this task stem from being able to produce the same high level of objective quality, but at a much more rapid pace (20% faster on average) and with a higher level of satisfaction with the end product.

We speculate that one of the contributing factors for why people were accepting of the AI’s suggestions is because the final decision of taking those suggestions was up to the human. As an alternative, we could have designed the system to always automatically produce a markdown documentation cell for each identified code cell, but we decided not to. Because this fully-automated design is an extreme in the framework of automation put forth by Parasuraman et al. ([47]; see also [18]), which people may feel being replaced. Our results confirmed our assumption—participants reported that they enjoyed being able to see multiple suggestions, created using different algorithms, and select the one that was the closest match to their intent in documenting a code cell. This level of interaction corresponds to “AI executes a selection only after a human has approved” in the Parasuraman et al. model [47].

Our result also shed light on the research question in [70] about the conditions under which human data scientists will enjoy working with AI partnership. In our case, maintaining control of the initiative and the final decision is an important aspect for people’s enjoyment and acceptance of the AI system. It remains to be studied whether people prefer both to control their own initiative *and* the initiative of a machine teammate, as proposed in Shneiderman’s recent two-dimensional model [63]. Also, we did not focus on the explainability or trust aspect of the designed AI system, such as how to visualize the connection between the generated documentation and the original code. In the future, the explainability and trust aspects of the AI system in the data science context is a very critical research topic (e.g., [9, 71]), and should also be prioritized in the research agenda.

There are many other tasks in a data science project’s lifecycle that could use AI’s help, such as model presentation or feature engineering [68]. In the future, we plan to extend our work to design more human-centered AI systems to support users in these data science tasks as well.

In the future, we plan to explore whether the identified benefits and tradeoffs persist or not after a long period of adoption by users. One of the potential benefits could be: the human-AI collaboration work style helps users to learn more from the AI suggested/reminded documentations, thus they realize more of the value of adding code documentation to notebooks; in contrary, maybe users become over-reliance on AI systems thus they de-skill in this code documentation task, both hypotheses await future research to evaluate.

### 6.3 Design Implications

We offer designers and tool builders the following suggestions to encourage data scientists to write better documentation:

**6.3.1 Towards Hybrid and Adapted Code Summarization.** Our evaluation of Themisto indicates that instead of a fully automatic approach, data scientists prefer to use a hybrid method for helping them write documentation. We argue that future work for code summarization should investigate a hybrid and adapted approach. We suggest that *adaptive interactive prompting* may be a worthwhile research topic. For example, prompts could be based on the contents of the code cell which the user was trying to document. Another possibility is that prompts be based on the user's own history of writing markdown cells, and could either appeal to the user's strengths, or could anticipate and accommodate the user's weaknesses. In a more socially-oriented approach, users within an organization might rate the initial set of prompts, voting some prompts up or down depending on their usefulness. An evolution of this idea might allow users to propose new prompts for use by selves and others (e.g., [8]). Furthermore, we argue that future code summarization tools would benefit from a reinforcement learning approach, which learns from users' modifications to the original proposed texts, and could anticipate the users' preference in subsequent documentation.

**6.3.2 Customizing the Recommendations Based on Usage Scenarios.** As prior work stated [77], data science workers engage in various collaborations during different stages of the data science lifecycle. Documentation plays an important role in many scenarios. For example, handing off work between data engineers, communicating results with stakeholders, or informal notes to future self. Data science workers may have different needs of the documentation for different usage scenarios. Designers and tool builders should take a user-centered approach to understand the purpose of documentation, the appropriate level of details, and the best way to present the documentation. For example, participants suggested that future versions of Themisto being able to document the changes of versions, errors, and related online forum posts. Participants also suggested that they would like to see more varieties into the generated narrative's formatting.

**6.3.3 Inverting Themisto—Automatic Code Generation from Documentation.** The premise of Themisto was to generate descriptive material based on program code. Following some of the ideas in Seeber et al. [61], we might invert this strategy. We recall that P3 told us that he wrote documentation in advance of writing the code itself. If there are other people who use the same discipline as P3, could we generate code from the descriptive text? We suspect that this idea would not work for just *any* textual description. However, there could be certain stylized ways of writing descriptions that might be translatable into code; pseudocode could provide a starting point for the design of such a stylized type of description. We recognize that this kind of approach would need to have a representation of code packages and libraries, so that it could generate code that was appropriately structured for those packages. Of course, package documentation could be used to construct such a representation.

## 6.4 Limitation

Our formative study only explores notebooks from the Kaggle corpus, which may leave out some varieties of markdown cells that only exist in messy notebooks that can benefit from the support of documentation generation. However, notebooks on the Kaggle platforms are based on real data and real problems, and they aim for rich explanations and narratives, where other places do not have high-quality notebooks with rich documentation. Future work should expand the exploration on the other notebook corpus, for example, notebooks published with scientific article which contain fine-grained documentation.

Our experiment has several limitations: it focuses only on the documenting (instead of coding) process, it is a controlled experiment study, and participants did not work on notebooks created by themselves. Thus, for example, we do not know how participants would perceive the usefulness of the tool in realistic notebooks, which may be longer and more complicated (e.g., having out-of-

order cells) than the notebooks we provided. However, we believe the result is still promising to shed light for future research and future system design. Future work can explore the generalizability of Themisto through a long-term deployment study.

As for the Human-AI Collaboration research initiative, our work only reports the findings on how human and AI collaborated at a coarse-grained level (Table 6). In future work, we will have an in-depth analysis to break down the level of individual cells, and further analyze the difference between automatically-generated, co-edited, or manually-produced cells. This detailed analysis will help us to understand how human behave and perceive the fine-grained collaboration and interaction with the AI partner. And such findings and its derived design insights could also help researchers who are studying Human-AI Collaborations in other usage scenarios (e.g., in Healthcare or in Educational settings [76]) beyond the notebook documentation context in this article.

## 7 CONCLUSION

In this article, we have designed and built Themisto to support human data scientists in the notebook documentation task. This research prototype also serves as a prompt to explore the human-AI collaboration research agenda within the automated notebook documentation user scenario. The system design is driven by insights from previous literature, and also by a formative study that analyzed 80 highly-voted Kaggle notebooks to understand how human data scientists document notebooks. The follow-up user evaluation suggested that the collaboration between data scientists and Themisto significantly reduced task completion time and resulted in a final artifact that not only met the bar of quality, but also exceed it for the level of satisfaction.

## APPENDIX

### A EXAMPLE OF DOCUMENTATION GENERATION IN THEMISTO



Table 7. Example Notebook (House Prediction), Documentation Generation in Themisto, and User Submitted Documentation (T—Markdown Cells Created by Themisto Only, C—Markdown Cells Co-Created by Humans and Themisto, H—Markdown Cells Created by Humans Only)

Source Code	DL-Based	Query-Based	Prompt-Based	P2
<pre>import pandas as pd import numpy as np from sklearn.linear_model import LassoCV from sklearn.model_selection import cross_val_score train = pd.read_csv('train.csv') test = pd.read_csv('test.csv')</pre>	Importing libraries	Pandas is for data manipulation and analysis; NumPy is a library for ...	This code cell is for -----	Importing libraries (T)
<pre>train.head()</pre>	Read the data values	Read a comma-separated values (csv) file into DataFrame; Return the first 5 rows.	This code cell is for -----	Read the data (T)
<pre>all_data = pd.concat(( train.loc[:, 'SaleClass': 'SaleCond'], test.loc[:, 'SaleClass': 'SaleCond']))</pre>	Let's see the values A generator for feature	Return the first 5 rows Concatenate pandas objects along a particular axis with optional set logic along the other axes.	The table shows ----- This code cell is for -----	Return the first 5 rows. (defValue=5) (C) Concat train and test col "SaleCondition" (C)
<pre>all_data = pd.get_dummies(all_data)</pre>	Convert all the data	Convert categorical variable into dummy/indicator variables	This code cell is for -----	Convert categorical variable into dummy/indicator variables. (T)
<pre>all_data = all_data .fillna(all_data.mean()) X_train = all_data[train.shape[0]] X_test = all_data[train.shape[0]:] y = train.SalePrice</pre>	Check the missing values Create the target and the test data	Fill NA/NaN values using the specified method Slice string	This code cell is for ----- This code cell is for -----	
<pre>model_lasso = LassoCV( alphas = [1, 0.1, 0.001, 0.0005]) .fit(X_train, y)</pre>	Model	Lasso linear model with iterative fitting along a regularization path.	This code cell is for -----	Fit regression model (H)
<pre>def rmse_cv(model): rmse = np.sqrt(-cross_val_score( model, X_train, y, scoring = "neg_mean_squared_error", cv = 5)) return(rmse) rmse_cv(model_lasso).mean()</pre>	A simple example model with the lasso	Evaluate a score by cross-validation	The result indicates that -----	Define score function and evaluate (H)

Table 8. Example Notebook (Covid Prediction), Documentation Generation in Themisto, and User Submitted Documentation (T—Markdown Cells Created by Themisto Only, C—Markdown Cells Co-Created by Humans and Themisto, H—Markdown Cells Created by Humans Only)

Source Code	DL-Based	Query-Based	Prompt-Based	P5
<pre>import numpy as np import pandas as pd from sklearn.ensemble import RandomForestClassifier</pre>	Importing libraries	Pandas is for data manipulation and analysis; NumPy is a library for ...	This code cell is for _____	Importing libraries (T)
<pre>train = pd.read_csv("train.csv") test = pd.read_csv("test.csv") train.head()</pre>	Read the data	Read a comma-separated values (csv) file into DataFrame; Return the first 5 rows.	The table shows _____	Read and sanity check the data (C)
<pre>train.describe()</pre>	Let's see the values	Generate descriptive statistics. Descriptive statistics include ...	The table shows _____	
<pre>train["Date"] = train["Date"] .apply(lambda x: x.replace("-", "")) train["Date"] = train["Date"] .astype(int) train.head()</pre>	Convert all the data	Replace a specified phrase with another specified phrase	The table shows _____	Preprocess the data (C)
<pre>train.isnull().sum()</pre>	Check the missing values	Detect missing values for an array-like object	The result indicates that _____	Check the missing values (T)
<pre>test["Date"] = test["Date"] .apply(lambda x: x.replace("-", "")) test["Date"] = test["Date"]</pre>	Convert all the data	Replace a specified phrase with another specified phrase	This code cell is for _____	Preprocess the date column (C)
<pre>x = train[['Lat', 'Long', 'Date']] y = train[['ConfirmedCases']] x_test = test[['Lat', 'Long', 'Date']]</pre>	Create the target and the test data	Select subsets of data	This code cell is for _____	Create the train/test data and the target (C)
<pre>Tree_model = RandomForestClassifier( max_depth = 200, random_state = 0) Tree_model.fit(x, y)</pre>	Model	A random forest is a meta estimator that fits a number of decision tree classifiers on ...	This code cell is for _____	Define and configure the model A random forest is a meta ... We also train the model with 'fit()' (C)
<pre>pred = Tree_model.predict(x_test) pred = pd.DataFrame(pred) pred.columns = ["ConfirmedCases_prediction"]</pre>	Predicate to use a predicate function for tests	A random forest is a meta estimator that fits a number of decision tree classifiers on ...	This code cell is for _____	Run the model to generate predictions on the test data and store them as a 'DataFrame'(H)

## B CODING BOOK FOR THE INTERVIEW TRANSCRIPTS

Table 9. Coding Book for the Interview Transcripts

Theme	Code
Pros of Themisto	Easy to Use
	Provide Inspirations
	Improve Content
	Efficiency
	Hybrid Approach
	Useful for Long Term
Cons of Themisto	Prefer the Plugin
	Inaccurate
Perceptions of the Deep-Learning-Based Approach	Not Useful
	Concise
	Useful
	Accurate
	Inaccurate
Perceptions of the Query-Based Approach	For Own Use
	For Collaboration Use
	Descriptive
	Too Long
Perceptions of the Prompt-Based Approach	Useful
	Confusing
	Instructive
	Tedious
Future Adoption	Easy to Use
	Inspiring
	Positive Adoption Propensity
Design Improvements	Scenarios for Future Adoption
	Negative Adoption Propensity
	More Options Generated by AI
	Handle Presentation and Formatting
	Summarize Other Information (e.g., Reasons, Summary, Errors)
	Customization
	Optimize UI
	Adaptive Prompts

## ACKNOWLEDGMENTS

We thank all of our participants for their help in the study, and the anonymous reviewers for their valuable feedback.

## REFERENCES

- [1] Rajas Agashe, Srinivasan Iyer, and Luke Zettlemoyer. 2019. Juice: A large scale distantly supervised dataset for open domain context-based code generation. arXiv:1910.02216. Retrieved from <https://arxiv.org/abs/1910.02216>.
- [2] Uri Alon, Shaked Brody, Omer Levy, and Eran Yahav. 2018. code2seq: Generating sequences from structured representations of code. arXiv:1808.01400. Retrieved from <https://arxiv.org/abs/1808.01400>.

- [3] Matthew Arnold, Rachel K. E. Bellamy, Michael Hind, Stephanie Houde, Sameep Mehta, Aleksandra Mojsilović, Ravi Nair, K. Natesan Ramamurthy, Alexandra Olteanu, David Piorkowski, D. Reimer, J. Richards, J. Tsay, and K. R. Varshney. 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] Liang Bai and Yanli Hu. 2018. Problem-driven teaching activities for the capstone project course of data science. In *Proceedings of the ACM Turing Celebration Conference-China*. 130–131.
- [5] Carrie J. Cai, Emily Reif, Narayan Hegde, Jason Hipp, Been Kim, Daniel Smilkov, Martin Wattenberg, Fernanda Viegas, Greg S. Corrado, Martin C. Stumpe, and Michael A Terry. 2019. Human-centered tools for coping with imperfect algorithms during medical decision-making. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. 1–14.
- [6] Souti Chattopadhyay, Ishita Prasad, Austin Z. Henley, Anita Sarma, and Titus Barik. 2020. What’s wrong with computational notebooks? Pain points, needs, and design opportunities. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. 1–12.
- [7] Sergio Cozzetti B. de Souza, Nicolas Anquetil, and Káthia M. de Oliveira. 2005. A study of the documentation essential to software maintenance. In *Proceedings of the 23rd Annual International Conference on Design of Communication: Documenting & Designing for Pervasive Information*. 68–75.
- [8] Joan Morris DiMicco, Werner Geyer, David R. Millen, Casey Dugan, and Beth Brownholtz. 2009. People sensemaking and relationship building on an enterprise social network site. In *Proceedings of the 42nd Hawaii International Conference on System Sciences*. IEEE, 1–10.
- [9] Jaimie Drozdal, Justin Weisz, Dakuo Wang, Gaurav Dass, Bingsheng Yao, Changruo Zhao, Michael Muller, Lin Ju, and Hui Su. 2020. Trust in automl: Exploring information needs for establishing trust in automated machine learning systems. In *Proceedings of the 25th International Conference on Intelligent User Interfaces*. 297–307.
- [10] Brian P. Eddy, Jeffrey A. Robinson, Nicholas A. Kraft, and Jeffrey C. Carver. 2013. Evaluating source code summarization techniques: Replication and expansion. In *Proceedings of the 21st International Conference on Program Comprehension*. IEEE, 13–22.
- [11] Jesus Fernandez-Bes, Jerónimo Arenas-García, and Jesús Cid-Sueiro. 2016. Energy generation prediction: Lessons learned from the use of kaggle in machine learning course. *Group* 7, 8 (2016), 9.
- [12] Golar Garousi, Vahid Garousi, Mahmoud Moussavi, Guenther Ruhe, and Brian Smith. 2013. Evaluating usage and quality of technical software documentation: An empirical study. In *Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering*. 24–35.
- [13] Timnit Gebru, Jamie Morgenstern, Briana Vecchione, Jennifer Wortman Vaughan, Hanna Wallach, Hal Daumé III, and Kate Crawford. 2018. Datasheets for datasets. arXiv:1803.09010. Retrieved from <https://arxiv.org/abs/1803.09010>.
- [14] R. Stuart Geiger, Nelle Varoquaux, Charlotte Mazel-Cabasse, and Chris Holdgraf. 2018. The types, roles, and practices of documentation in data analytics open source software libraries. *Computer Supported Cooperative Work* 27, 3–6 (2018), 767–802.
- [15] Andrew Head, Fred Hohman, Titus Barik, Steven M. Drucker, and Robert DeLine. 2019. Managing messes in computational notebooks. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (Glasgow, Scotland UK). ACM, New York, NY, Article 270, 12 pages. DOI: <https://doi.org/10.1145/3290605.3300500>
- [16] Sarah Holland, Ahmed Hosny, Sarah Newman, Joshua Joseph, and Kasia Chmielinski. 2018. The dataset nutrition label: A framework to drive higher data quality standards. arXiv:1805.03677. Retrieved from <https://arxiv.org/abs/1805.03677>.
- [17] Amber Horvath, Mariann Nagy, Finn Voichick, Mary Beth Kery, and Brad A. Myers. 2019. Methods for investigating mental models for learners of APIs. In *Proceedings of the Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems*. 1–6.
- [18] Eric Horvitz. 1999. Principles of mixed-initiative user interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 159–166.
- [19] Xing Hu, Ge Li, Xin Xia, David Lo, and Zhi Jin. 2018. Deep code comment generation. In *Proceedings of the 2018 IEEE/ACM 26th International Conference on Program Comprehension*. IEEE, 200–20010.
- [20] Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, and Luke Zettlemoyer. 2016. Summarizing source code using a neural attention model. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 2073–2083. DOI: <https://doi.org/10.18653/v1/P16-1195>
- [21] Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, and Luke Zettlemoyer. 2016. Summarizing source code using a neural attention model. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*. 2073–2083.
- [22] Project Jupyter. 2016. JupyterLab: The Next Generation of the Jupyter Notebook. Retrieved 01 September, 2021 from <https://blog.jupyter.org/jupyterlab-the-next-generation-of-the-jupyter-notebook-5c949dabea3>.
- [23] Project Jupyter. 2015. Project Jupyter: Computational Narratives as the Engine of Collaborative Data Science. Retrieved September 15, 2019 from Retrieved from <https://blog.jupyter.org/project-jupyter-computational-narratives-as-the-engine-of-collaborative-data-science-2b5fb94c3c58>.

- [24] Mira Kajko-Mattsson. 2005. A survey of documentation practice within corrective maintenance. *Empirical Software Engineering* 10, 1 (2005), 31–55.
- [25] Mary Beth Kery, Bonnie E. John, Patrick O’Flaherty, Amber Horvath, and Brad A. Myers. 2019. Towards effective foraging by data scientists to find past analysis choices. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. ACM, New York, NY, Article 92, 13 pages. DOI : <https://doi.org/10.1145/3290605.3300322>
- [26] Mary Beth Kery and Brad A. Myers. 2017. Exploring exploratory programming. In *Proceedings of the 2017 IEEE Symposium on Visual Languages and Human-Centric Computing*. 25–29. DOI : <https://doi.org/10.1109/VLHCC.2017.8103446>
- [27] Mary Beth Kery, Marissa Radensky, Mahima Arya, Bonnie E. John, and Brad A. Myers. 2018. The story in the notebook: Exploratory data science using a literate programming tool. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. ACM, New York, NY, Article 174, 11 pages. DOI : <https://doi.org/10.1145/3173574.3173748>
- [28] Donald E. Knuth. 1984. Literate programming. *The Computer Journal* 27, 2 (1984), 97–111. DOI : <https://doi.org/10.1093/comjnl/27.2.97>
- [29] Markus Konkol, Daniel Nüst, and Laura Goulier. 2020. Publishing computational research—a review of infrastructures for reproducible and transparent scholarly communication. arXiv:2001.00484. Retrieved from <https://arxiv.org/abs/2001.00484>.
- [30] Sean Kross and Philip J. Guo. 2021. Orienting, framing, bridging, magic, and counseling: How data scientists navigate the outer loop of client collaborations in industry and academia. arXiv:2105.05849. Retrieved from <https://arxiv.org/abs/2105.05849>.
- [31] Sam Lau, Ian Drosos, Julia M. Markel, and Philip J. Guo. 2020. The design space of computational notebooks: An analysis of 60 systems in academia and industry. In *Proceedings of the 2020 IEEE Symposium on Visual Languages and Human-Centric Computing*. IEEE, 1–11.
- [32] Alexander LeClair, Sakib Haque, Linfgei Wu, and Collin McMillan. 2020. Improved code summarization via a graph neural network. arXiv:2004.02843. Retrieved from <https://arxiv.org/abs/2004.02843>.
- [33] Jiali Liu, Nadia Boukhelifa, and James R. Eagan. 2019. Understanding the role of alternatives in data analysis practices. *IEEE Transactions on Visualization and Computer Graphics* 26, 1 (2019), 66–76.
- [34] Sijia Liu, Parikshit Ram, Deepak Vijaykeerthy, Djallel Bouneffouf, Gregory Bramble, Horst Samulowitz, Dakuo Wang, Andrew Conn, and Alexander Gray. 2020. An ADMM based framework for automl pipeline configuration. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 4892–4899.
- [35] Xuye Liu, Dakuo Wang, April Yi Wang, Yufang Hou, and Lingfei Wu. 2021. HAConvGNN: Hierarchical attention based convolutional graph neural network for code documentation generation in jupyter notebooks. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing: Findings*.
- [36] Ryan Louie, Andy Coenen, Cheng Zhi Huang, Michael Terry, and Carrie J. Cai. 2020. Novice-AI music co-creation via ai-steering tools for deep generative models. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. 1–13.
- [37] Julia S. Stewart Lowndes, Benjamin D. Best, Courtney Scarborough, Jamie C. Afflerbach, Melanie R. Frazier, Casey C. O’ Hara, Ning Jiang, and Benjamin S. Halpern. 2017. Our path to better science in less time using open data science tools. *Nature Ecology & Evolution* 1, 6 (2017), 1–7.
- [38] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective approaches to attention-based neural machine translation. arXiv:1508.04025. Retrieved from <https://arxiv.org/abs/1508.04025>.
- [39] Walid Maalej and Martin P. Robillard. 2013. Patterns of knowledge in API reference documentation. *IEEE Transactions on Software Engineering* 39, 9 (2013), 1264–1282.
- [40] Thomas W. Malone. 2018. How human–computer ‘Superminds’ are redefining the future of work. *MIT Sloan Management Review* 59, 4 (2018), 34–41.
- [41] 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.
- [42] Michael Muller, Ingrid Lange, Dakuo Wang, David Piorkowski, Jason Tsay, Q. Vera Liao, Casey Dugan, and Thomas Erickson. 2019. How data science workers work with data: Discovery, capture, curation, design, creation. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. ACM, New York, NY, Article 126, 15 pages. DOI : <https://doi.org/10.1145/3290605.3300356>
- [43] Michael Muller, April Yi Wang, Steven I. Ross, Justin D. Weisz, Mayank Agarwal, Kartik Talamadupula, Stephanie Houde, Fernando Martinez, John Richards, Jaimie Drozdal, Xuye Liu, David Piorkowski, and Dakuo Wang. 2021. How data scientists improve generated code documentation in jupyter notebooks.
- [44] Stephen Oney and Joel Brandt. 2012. Codelets: Linking interactive documentation and example code in the editor. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 2697–2706.
- [45] Yoann Padioleau, Lin Tan, and Yuanyuan Zhou. 2009. Listening to programmers-Taxonomies and characteristics of comments in operating system code. In *Proceedings of the 2009 IEEE 31st International Conference on Software Engineering*. IEEE, 331–341.

- [46] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: A method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*. 311–318.
- [47] Raja Parasuraman, Thomas B. Sheridan, and Christopher D. Wickens. 2000. A model for types and levels of human interaction with automation. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans* 30, 3 (2000), 286–297.
- [48] Soya Park, Amy X. Zhang, and David R. Karger. 2018. Post-literate programming: Linking discussion and code in software development teams. In *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology Adjunct Proceedings*. ACM, New York, NY, 51–53. DOI: <https://doi.org/10.1145/3266037.3266098>
- [49] Jeffrey M. Perkel. 2018. Why jupyter is data scientists’ computational notebook of choice. *Nature* 563, 7732 (2018), 145. DOI: <https://doi.org/10.1038/d41586-018-07196-1>
- [50] João Felipe Pimentel, Saumen Dey, Timothy McPhillips, Khalid Belhajjame, David Koop, Leonardo Murta, Vanessa Braganholo, and Bertram Ludäscher. 2016. Yin & Yang: Demonstrating complementary provenance from noWorkflow & YesWorkflow. In *Proceedings of the International Provenance and Annotation Workshop*. Springer, 161–165.
- [51] 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. *Proceedings of the ACM on Human-Computer Interaction* 5, CSCW1 (2021), 1–25.
- [52] Mohammed Suhail Rehman. 2019. Towards understanding data analysis workflows using a large notebook corpus. In *Proceedings of the 2019 International Conference on Management of Data*. 1841–1843.
- [53] Marco Tulio Ribeiro, Tongshuang Wu, Carlos Guestrin, and Sameer Singh. 2020. Beyond accuracy: Behavioral testing of NLP models with checklist. arXiv:2005.04118. Retrieved from <https://arxiv.org/abs/2005.04118>.
- [54] Tobias Roehm, Rebecca Tiarks, Rainer Koschke, and Walid Maalej. 2012. How do professional developers comprehend software? In *Proceedings of the 2012 34th International Conference on Software Engineering*. IEEE, 255–265.
- [55] Adam Rule, Amanda Birmingham, Cristal Zuniga, Ilkay Altintas, Shih-Cheng Huang, Rob Knight, Niema Moshiri, Mai H. Nguyen, Sara Brin Rosenthal, Fernando Pérez, and Peter W. Rose. 2019. Ten simple rules for writing and sharing computational analyses in Jupyter Notebooks.
- [56] Adam Rule, Ian Drosos, Aurélien Tabard, and James D. Hollan. 2018. Aiding collaborative reuse of computational notebooks with annotated cell folding. *Proceedings of the ACM on Human-Computer Interaction* 2, CSCW (2018), 150:1–150:12. DOI: <https://doi.org/10.1145/3274419>
- [57] Adam Rule, Aurélien Tabard, and James D. Hollan. 2018. Exploration and explanation in computational notebooks. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. 1–12.
- [58] Jeffrey Saltz, Kevin Crowston, and Ivan Shamshurin. 2017. Comparing data science project management methodologies via a controlled experiment. In *Proceedings of the Hawaii International Conference on System Sciences*.
- [59] Sheeba Samuel and Birgitta König-Ries. 2018. ProvBook: Provenance-based semantic enrichment of interactive notebooks for reproducibility. In *Proceedings of the 17th International Semantic Web Conference*.
- [60] Sheeba Samuel and Birgitta König-Ries. 2020. ReproduceMeGit: A visualization tool for analyzing reproducibility of jupyter notebooks. arXiv:2006.12110. Retrieved from <https://arxiv.org/abs/2006.12110>.
- [61] Isabella Seeber, Eva Bittner, Robert O. Briggs, Triparna de Vreede, Gert-Jan De Vreede, Aaron Elkins, Ronald Maier, Alexander B. Merz, Sarah Oeste-Reiß, Nils Randrup, Gerhard Schwabe, and Matthias Söllner. 2020. Machines as teammates: A research agenda on AI in team collaboration. *Information & Management* 57, 2 (2020), 103174.
- [62] Lin Shi, Hao Zhong, Tao Xie, and Mingshu Li. 2011. An empirical study on evolution of API documentation. In *Proceedings of the International Conference on Fundamental Approaches to Software Engineering*. Springer, 416–431.
- [63] Ben Shneiderman. 2020. Human-centered artificial intelligence: Reliable, safe & trustworthy. *International Journal of Human-Computer Interaction* 36, 6 (2020), 495–504.
- [64] Giriprasad Sridhara, Emily Hill, Divya Muppaneni, Lori Pollock, and K. Vijay-Shanker. 2010. Towards automatically generating summary comments for java methods. In *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*. 43–52.
- [65] April Yi Wang, Zihan Wu, Christopher Brooks, and Steve Oney. 2020. Callisto: Capturing the “Why” by connecting conversations with computational narratives. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. ACM.
- [66] Dakuo Wang. 2016. How people write together now: Exploring and supporting today’s computer-supported collaborative writing. In *Proceedings of the 19th ACM Conference on Computer Supported Cooperative Work and Social Computing Companion*. ACM, 175–179.
- [67] Dakuo Wang, Josh Andres, Justin Weisz, Erick Oduor, and Casey Dugan. 2021. AutoDS: Towards human-centered automation of data science. In *Proceeding of the 2021 CHI Conference on Human Factors in Computing Systems*.
- [68] Dakuo Wang, Q. Vera Liao, Yunfeng Zhang, Udayan Khurana, Horst Samulowitz, Soya Park, Michael Muller, and Lisa Amini. 2021. How much automation does a data scientist want?. arXiv:2101.03970. Retrieved from <https://arxiv.org/abs/2101.03970>.



- [69] Dakuo Wang, Parikshit Ram, Daniel Karl I. Weidele, Sijia Liu, Michael Muller, Justin D. Weisz, Abel Valente, Arunima Chaudhary, Dustin Torres, Horst Samulowitz, and Lisa Amini. 2020. Autoai: Automating the end-to-end ai lifecycle with humans-in-the-loop. In *Proceedings of the 25th International Conference on Intelligent User Interfaces Companion*. 77–78.
- [70] Dakuo Wang, Justin D. Weisz, Michael Muller, Parikshit Ram, Werner Geyer, Casey Dugan, Yla Tausczik, Horst Samulowitz, and Alexander Gray. 2019. Human-AI collaboration in data science: Exploring data scientists’ perceptions of automated AI. *Proceedings of the ACM on Human-Computer Interaction* 3, CSCW (2019), 1–24.
- [71] Daniel Karl I. Weidele, Justin D. Weisz, Erick Oduor, Michael Muller, Josh Andres, Alexander Gray, and Dakuo Wang. 2020. AutoAIViz: Opening the blackbox of automated artificial intelligence with conditional parallel coordinates. In *Proceedings of the 25th International Conference on Intelligent User Interfaces*. 308–312.
- [72] Justin D. Weisz, Mohit Jain, Narendra Nath Joshi, James Johnson, and Ingrid Lange. 2019. BigBlueBot: Teaching strategies for successful human-agent interactions. In *Proceedings of the 24th International Conference on Intelligent User Interfaces*. 448–459.
- [73] John Wenskovich, Jian Zhao, Scott Carter, Matthew Cooper, and Chris North. 2019. Albireo: An interactive tool for visually summarizing computational notebook structure. In *Proceedings of the 2019 IEEE Visualization in Data Science*. IEEE, 1–10.
- [74] Jo Wood, Alexander Kachkaev, and Jason Dykes. 2018. Design exposition with literate visualization. *IEEE Transactions on Visualization and Computer Graphics* 25, 1 (2018), 759–768.
- [75] Kun Xu, Lingfei Wu, Zhiguo Wang, Yansong Feng, Michael Witbrock, and Vadim Sheinin. 2018. Graph2seq: Graph to sequence learning with attention-based neural networks. arXiv:1804.00823. Retrieved from <https://arxiv.org/abs/1804.00823>.
- [76] Ying Xu, Dakuo Wang, Penelope Collins, Hyelim Lee, and Mark Warschauer. 2021. Same benefits, different communication patterns: Comparing Children’s reading with a conversational agent vs. a human partner. *Computers & Education* 161 (2021), 104059.
- [77] Amy X. Zhang, Michael Muller, and Dakuo Wang. 2020. How do data science workers collaborate? Roles, workflows, and tools. arXiv:2001.06684. Retrieved from <https://arxiv.org/abs/2001.06684>.

Received June 2021; revised September 2021; accepted September 2021