Lodestar: Supporting Independent Learning and Rapid Experimentation Through Data-Driven Analysis Recommendations

Deepthi Raghunandan, Zhe Cui, Kartik Krishnan, Segen Tirfe, Shenzhi Shi,
Tejaswi Darshan Shrestha, Leilani Battle, and Niklas Elmqvist, *Senior Member, IEEE*

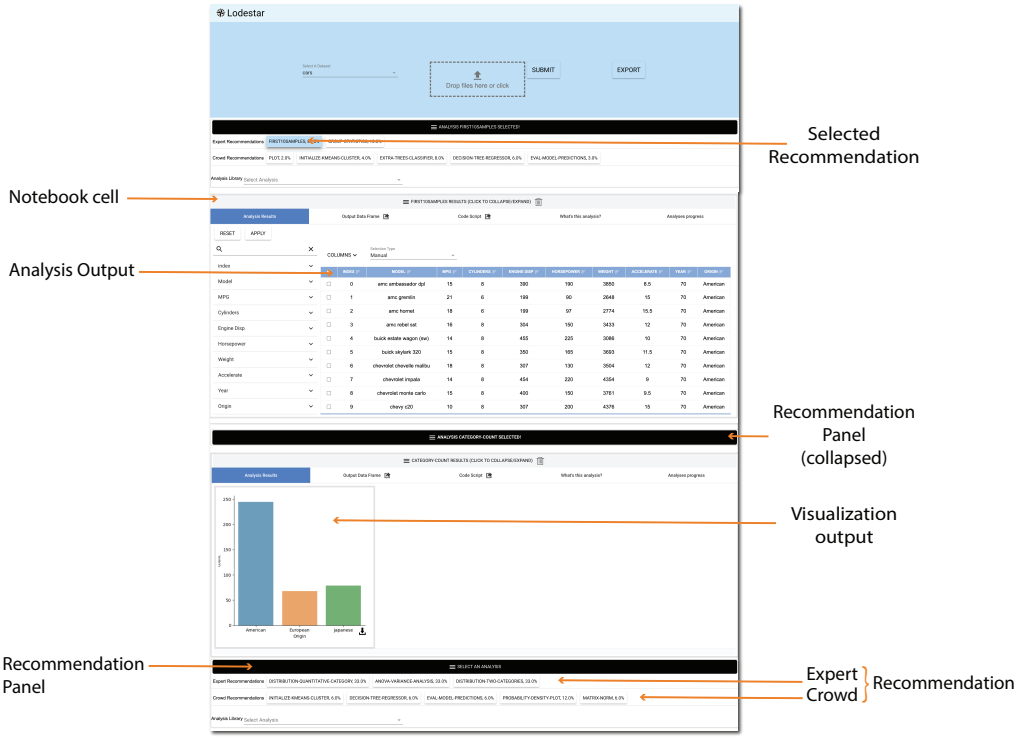

Fig. 1: **Lodestar web interface.** The top panel, above the selected recommendations, provides a data selection menu. The black dividers between sections are recommendation panels combining suggested analysis steps from various sources (called advisors). Areas that have graphs and analysis outputted are analysis cells, each with multiple tabs: "Analysis Results" gives charts or tables, "Output Dataframe" and "Code Script" shows the outputs and current code block, and "What's this analysis?" gives a brief description of the analyses. Outputs, code, and charts can also be exported.

Abstract— Keeping abreast of current trends, technologies, and best practices in visualization and data analysis is becoming increasingly difficult, especially for fledgling data scientists. In this paper, we propose Lodestar, an interactive computational notebook that allows users to quickly explore and construct new data science workflows by selecting from a list of automated analysis recommendations. We derive our recommendations from directed graphs of known analysis states, with two input sources: one manually curated from online data science tutorials, and another extracted through semi-automatic analysis of a corpus of over 6,000 Jupyter notebooks. We evaluate Lodestar in a formative study guiding our next set of improvements to the tool. Our results suggest that users find Lodestar useful for rapidly creating data science workflows.

Index Terms—Computational notebook, visualization recommendation, Markov chain, data science, Python.

- *Deepthi Raghunandan, Department of Computer Science, University of Maryland, E-mail: draghun1@umd.edu.*
- *Zhe Cui, Google, Inc., E-mail: zhecui@google.com.*
- *Kartik Krishnan, Department of Computer Science, University of Maryland, E-mail: kkrishn1@umd.edu.*
- *Segen Trife, Department of Information Science, University of Maryland, E-mail: segent@umd.edu.*
- *Shenzhi Shi, Department of Computer Science, University of Maryland, E-mail: sshi1234@umd.edu.*
- *Tejaswi Darshan Shrestha, Department of Computer Science, University of Maryland, -mail: shrestha.tj@gmail.com.*
- *Leilani Battle, Department of Computer Science and Engineering, University of Washington, E-mail: leibatt@cs.washington.edu.*
- *Niklas Elmqvist, College of Information Studies, University of Maryland, E-mail: elm@umd.edu*

Manuscript received xx xxx. 201x; accepted xx xxx. 201x. Date of Publication xx xxx. 201x; date of current version xx xxx. 201x. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org. Digital Object Identifier: xx.xxxx/TVCG.201x.xxxxxxx

1 INTRODUCTION

Data science is still a nascent and emerging discipline, which makes it challenging for analysts to learn and keep up with new tools and techniques. There is already a dizzying array of libraries, such as Scikit-Learn [40], Pandas [36], and TensorFlow [1], and best practices and workflows change often. Furthermore, few standardized methods exist for data analysis: many times, the exact data transformations, computations, and analyses needed depends on the data, task, and user. This means that cookbook methods or simple workflow templates are insufficient to teach fledgling analysts how to tackle realistic and ever-changing data science problems.

In response, we present LODESTAR, an interactive and visual sandbox environment for independent learning of new data analysis methods and best practices in data science. Our aim in developing Lodestar is to simplify the process of finding and experimenting with new analysis methods by providing automated, data-driven recommendations. We want Lodestar to be a self-contained environment for rapid learning and iteration, where everything the user needs to infer the function and purpose of an analysis step is available in one place.

The Lodestar system uses a computational notebook interface (similar to a Jupyter notebook [24]) showing a sequence of analysis steps in the form of Python code cells (see Figure 1), but enables the user to initially select from and interact with self-contained code cells without having to write any code. The user merely selects which data frame to analyze, and the system displays a ranked list of recommendations of analysis steps to be executed on that data. Each analysis step is represented by an interactive visualization in the notebook interface, giving the user insights into its output and behavior. Furthermore, users can view the corresponding code for any analysis step, and even export the resulting notebook from Lodestar, providing flexibility in how users learn from and interact with Lodestar's analysis recommendations.

Lodestar provides recommendations for the user's next analysis step based on the current state of the analysis and the dataset being analyzed. Recommended analysis steps and workflows are derived from two sources representing current best practices in data science: (1) existing data science tutorials from online academies and training materials (i.e., an *expert recommendation*), and (2) common analysis patterns mined from a large corpus of publicly available Jupyter notebooks [47] (i.e., a *crowd recommendation*). The code cells extracted from each source are manually curated, then programmatically clustered into synonymous analysis steps, and inserted into a large directed graph of connected cells representing common analysis workflows. The Lodestar recommendation engine can then identify and rank the most relevant analysis steps given a specific position in the graph.

We developed Lodestar using an iterative design process. We used early feedback from six participants to improve the interface design. Our findings show that key Lodestar interactive features, such as automated recommendations, a visualization of the full analysis workflow, a code review pane for suggested analysis steps, and export support for Jupyter Notebooks, provide significant value to those who are learning data science.

In this work, we make the following contributions: (1) a holistic recommendation process involving two sources of data analysis practice: crowd-based and expert-based; (2) a sandbox interface design integrating visualizations, interactions, and code to facilitate learning about new data analysis techniques; (3) results from a formative study evaluating the Lodestar design; and (4) a unique data analysis architecture that integrates a recommender system with a computational notebook interface. All our materials, including source code, documentation, and study results, have been made available on the following OSF page: `https://osf.io/pztva/`

2 BACKGROUND

2.1 Sensemaking

Lodestar was architected to aid and encourage best practices in sensemaking. Richard Hamming described sensemaking as "the process of searching for a representation and encoding data in that representation to answer task-specific questions" [48]. Dubbed the *sensemaking loop* [42], each sensemaking iteration works to refine and build on the previous insights—ultimately enabling the analyst to address less specialized audiences [60]. In combination, these iterations make up the data science workflow. Analysts usually use visualizations or other types of intermediate results to motivate further analysis. However, these results can sometimes be dead ends. Kandel et al. [26] found that analysts will overcome dead ends by backtracking and exploring new branches.

2.2 Interactive Visualization Design Environments

Many visualization systems and toolkits are designed around specific data analysis tasks, making the analysis process easier to perform. Excel supports basic visualization and data transformations. Shelf-based visualization environments such as Tableau (née Polaris [56]) allow easy configuration of visualizations through drag-and-drop of data attributes and metadata onto "shelves" representing visual channels. This approach is flexible enough for even novice users to construct a wide range of visualizations. Interactive visual design environments such as Lyra [50], iVoLVER [37], and iVisDesigner [44] utilize direct manipulation to allow users to bind data to visual representations. More recently, Data-Driven Guides [28], Data Illustrator [32], DataInk [71], and Charticulator [45] provide advanced tools for representing data items as visual elements and mapping their attributes to data dimensions. Keshif [72], a faceted visualization tool, generates grids of predefined charts to support visual exploration by novice users.

Visualization development toolkits such as D3 [8] and Protovis [7] provide fine-grained control over designing interactive visualizations, but require significant programming expertise to use. Visualization grammars, such as ggplot2 [67], Vega [52], and Vega-Lite [51], abstract away implementation details, but still require programming knowledge to use. Furthermore, even advanced visualization tools, toolkits, and grammars offer only limited functionality for manipulating the data, and only support a small number of statistical functions.

2.3 Visualization Recommendation

The purpose of visualization recommendation is to suggest relevant visualizations to the user to facilitate data analysis [21], where the visualizations are fully designed in advance and therefore directly accessible to the user. It was first proposed by Mackinlay [33] in 1986 with automatic design of effective presentations based on input data. The work combines expressiveness and effectiveness criteria from studies such as those by Bertin [6] and Cleveland et al. [9] to recommend appropriate visualizations. In 2007, Tableau's Show Me feature [34] revealed a commercial product with the implementation of these ideas. Following the idea of Mackinlay's automatic visualization, Roth et al. [46] enhances user-oriented design by completing and retrieving partial design graphics based on their appearance and data contents. The rank-by-feature framework [53] ranks histograms, scatterplots, and boxplots over 1D or 2D projections to find important features in multidimensional data. SeeDB [66] generates all possible visualizations given a query of the database and identifies interesting ones. Perry et al. [41] as well as van den Elzen and van Wijk [63] tackle the problem of generating small multiple visualizations shown as thumbnails using their statistical properties.

In the last few years, recommender systems have become widely used for visualization. Voyager [68] generates a large number of visualizations given a user-specified partial specification, and organizes them by data attributes. The generated visualizations are rendered as cards on a scrolling view. Saket et al. [49] propose the Visualization-by-Demonstration framework, which allows users to provide incremental changes to the visual representation. The system recommends potential transformations such as data mapping, axes, and view specification transformations. Zenvisage [54] automatically identifies and recommends desired visualizations from a large dataset. Voyager 2 [69] extended the original Voyager through wildcard functionality that explores all possible combinations of attributes. Most recently, Draco [38] even automates visualization design itself using partial specifications and a database of design knowledge expressed as constraints. VizML

learns what visualizations to recommend by training neural network models on millions of visualization designs made using Plotly [22].

Several tools extend these ideas to recommending analytical insights and data processing steps. "Top-K insights" [59] provides a theory for generating top K insights from multidimensional data. Similarly, Foresight [12] presents the top K insights in a dataset from 12 insight classes using a corresponding visualization. DataSite [11] organizes significant automatic findings in a specific feed of notifications. Finally, Voder [55] builds on a similar feed as DataSite to provide "interactive data facts" using visualizations.

Our proposed Lodestar system combines these ideas from visualization recommendation with an analytical perspective, and allows stringing together such analytical steps into a sequence. There are some existing efforts on recommending data analysis techniques and workflows. Yan et al. [73] demonstrate that online repositories of computational notebooks can be a valuable resource for modeling and testing a recommendation system for data cleaning techniques. Milo et al. [4] take this a step further by automatically generating entire data exploratory workflows using deep reinforcement learning techniques. Our system builds on these works by presenting a holistic model and code mining pipeline for deriving new recommendation features in a data-driven way, whether for data visualization, data preparation, or data analysis workflows. Essentially, Lodestar extends the idea of automated recommendations to the entire data science pipeline, rather than visualizations only.

2.4 Interactive Notebooks

Donald Knuth's notion of a "literate" form of programming [30], which merges source code with natural language and multimedia, has extended to the concept of *literate computing* in the form of computational notebooks [24, 29, 58], that combine executable code, its output, and media objects in a single document. This has proven to be very useful for rapid prototyping and exploration as well as for replicability and communication, particularly for data science and analysis [47].

Because of their success, with adoption even at the level of entire organizations [62], notebooks have enjoyed significant progress in recent years. The new generation of computational notebooks, such as Google Colaboratory [18] and Codestrates [43], enable synchronous collaboration. Beyond such features, the JavaScript-based Observable notebook [23] also supports one-way reactive execution flows.

Visualization in particular has recently begun to adopt computational notebooks. Altair [65] builds on Vega [50] and Vega-Lite [51] to provide statistical visualizations in Python, and thus in Jupyter Notebooks as well. Idyll [10] supports a notebook-like markup language to create interactive data-driven document for communication. Vistrates [3] provides a collaborative visualization workflow in a notebook. Observable [23] leverages the computational notebook environment to also provide a collaborative visualization platform. Literate visualization [70] integrates the visualization design process with the choices that led to the implementation.

End-user and live programming paradigms have proven useful in creating intuitive interactions with visualizations found in computational notebooks. For example, Wrex [13] and Mage [27] leverage user interactions on data visualizations to automatically generate exemplar code. As in traditional end-user programming platforms, Mage and Wrex demonstrate the link between code and visual interactions. Torii [19] uses a live programming model to enable easy maintenance and reuse of source code, for the purposes of building tutorials. These systems not only add to the number of ways users can interact with their literate document, but, they also create loops between code and visualization that lend well to data iteration.

3 Motivating Scenario

Here we describe a motivating data analysis scenario that highlights a knowledge gap among existing work that the Lodestar system aims to address. Suppose that an undergraduate student has just started learning about data analysis techniques in a university data science course. She is interested in selecting a dataset for her first assignment in the course. However, before testing out any new analysis methods on an unfamiliar dataset, the user wants to start with the simpler case of analyzing the popular Cars dataset, which is made available via a drop-down menu in the Lodestar interface upon startup. Upon selecting the Cars dataset, the user is presented with a list of initial data analysis steps, which have been recommended based on the user's choice of dataset (top of Figure 1). Each recommendation is represented as a clickable button. As the user reads the labels of the recommendation buttons, she may position her mouse over each, triggering tooltips with a brief description of each recommendation's behavior. The tooltip associated with the "first 10 samples" recommendation suggests that the user look at "the first 10 rows of the data frame."

To gain a better sense of what the Cars dataset contains, the user decides to select the "first 10 samples" recommendation. In response, Lodestar adds a notebook cell to the interface. This notebook cell contains results comprised of the Python code used to compute the analysis step (Figure 2), a brief description of the analysis technique, and a visualization of the results, in this case a table displaying the first 10 rows of the cars dataset (middle of Figure 1). The table shows that the Cars dataset contains 10 attributes, where at least three attributes appear to be categorical. The user is curious about what these categorical attributes contain.

Lodestar generates a new panel of recommendations below the new analysis cell, allowing the user to select the next step of her analysis workflow. In this round of recommendations, she sees a recommendation for generating "group statistics," which, according to the associated tooltip, promises to generate descriptive statistics of the dataset. These statistics could help her better identify the categorical attributes. Upon selecting "group statistics," another notebook cell is added to the interface, along with a third recommendation panel with potential follow-up analyses (bottom of Figure 1, collapsed).

In the table of summary statistics produced by the "group statistics" recommendation, the user notices that there are three unique categories in the "Origin" attribute. Curious about the distribution of cars per country of origin, she scans the previous round of recommendations for one that will allow her to dive deeper into categorical attributes. She finds the "category count" option within the second recommendation panel from which she originally chose "group statistics," where "Category count" will show "the distribution of categorical attributes across different values."

When she updates her selection to be "category count" instead of "group statistics," the original notebook cell for "group statistics" is removed and replaced with a new notebook cell displaying the results of "category count"; the third panel of recommendations is also updated accordingly. The output of the "category count" cell is a bar chart showing the number of cars for each country from the Origin attribute (bottom of Figure 1). The user observes that there are more "American" cars than any other kind.

The user now knows more about both the dataset and how to implement some of the statistical techniques she had encountered in class. She is curious to explore the Python code she observed in the notebook cells and the characteristics of cars from different countries. She scrolls up to the menu panel at the top of the interface and selects "export notebook"to save her current workflow as a Jupyter Notebook file (`cars_analysis.ipynb`) for further analysis and manual editing.

4 Design Requirements

Our goal is to make Lodestar an interactive and visual sandbox environment for learning and experimenting with new data science methods in a data-driven way. We also wanted to make data science universally accessible to fledgling data analysts and enthusiasts alike. These core ideas helped us compile a set of design requirements and some preliminary prototypes. In this section, we outline our major design requirements, and report on a formative study conducted to validate and refine our approach to the Lodestar interface design and system development processes.

- **D1: Informed by best practices.** Generated recommendations should be drawn from current practice, empowering those new to data science to learn how to effectively analyze their data [31, 34].

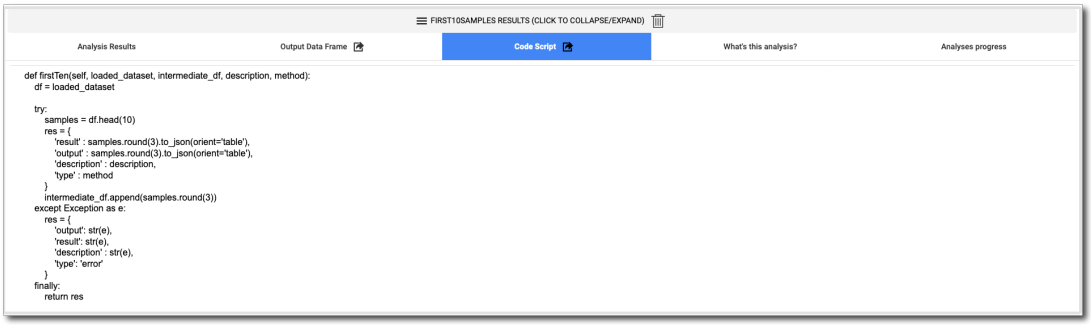

Fig. 2: Our fledgling analyst can explore the tabs within the analysis cell for details about the analysis step she chose to execute from the recommendation panel.

- **D2: Prioritize analysis steps over code.** Our intended users are trying to analyze data in a fast and fluid fashion, but may not yet be familiar with specific libraries or modules needed to complete different analysis steps. Lodestar needs to build a bridge between the high level analysis steps common in data science, and the low-level code needed to accomplish these steps [35]. For example, recommendations should be immediately relevant and situated within the overall data science pipeline to enable users to progress in their analysis.
- **D3: Enable independent exploration.** To ensure that users can explore their data independently, educational interface elements must also be incorporated to automatically provide documentation and clarification of system behavior [14]. Furthermore, intermediate and final results should be presented using visual representations that can be easily interpreted regardless of user expertise [61].

4.1 Formative Study

We conducted a formative user study to evaluate the usability of an early prototype of the Lodestar system, which we used to validate our initial design requirements and refine the system design. This study was approved by our home institution's IRB.

4.1.1 Study Design

The study was conducted over a period of one month in which we interviewed 6 fledgling analysts and data scientists; all undergraduate university students. We focused on recruiting university students, since they are generally learning data science methods for the first time and thus could provide helpful insights in our design process. Each student had demonstrated knowledge of data science fundamentals through attending a university-level introductory data science course and/or other relevant machine learning/data science experience. Although not a prerequisite of the recruitment process, some students also had experience performing analysis on platforms such as Excel and Tableau.

4.1.2 Method

Each interview lasted for 60 minutes and was divided into three phases. Prior to the interview, each participant signed a consent form, allowing us to record audio and screen capture throughout the duration of the interview. The first phase consisted of questions, delivered verbally, that assessed the participant's recent experience in learning data science techniques and tools through classes, side projects, research, and other such activities. During this section, participants were also asked specific questions regarding their view on recommender systems.

The second phase of the interview was dedicated to introducing an early prototype of the Lodestar system in which participants were given a brief 2-minute description of Lodestar and associated goals. The next 5 minutes were spent giving the participant a cursory tutorial of the system. For each participant, the tutorial was given using a pre-written script and with the same sample data set to give each of them equal knowledge of the system prior to their exploration. The participants then spent the next 15-20 minutes using the Lodestar system to conduct exploratory data analysis on a data set of their choosing. We restricted their choices to two datasets; the Boston House dataset from a Udacity tutorial,[1] and the ubiquitous Cars dataset. During this exploratory session, participants verbalized their thought process, questions, and comments with a think-aloud protocol. We encouraged participants to "to use any and all features of the Lodestar system" and to "explore whatever aspects of the data [they found] interesting." Participants were allowed to end the session before the allotted time expired if they were satisfied with their results.

The third and final phase of the interview consisted of a post-exploration questionnaire that asked participants to describe the utility of Lodestar for their common data analysis tasks. They were specifically asked if they would adopt Lodestar to learn new data science techniques and whether they trusted the recommendations.

All sessions were held in a lab environment using Google Chrome on a Macbook Pro with a 15-inch Retina display. Audio was recorded using the built-in voice recording application on a mobile device. Screen capture was done using Apple's QuickTime Player. Observational notes from the study coordinators, text responses from our questionnaires, and audio and video recordings were collected for further analysis and prioritization of design requirements and functional features of the existing prototype.

4.1.3 Results

Our formative study found that a majority of participants were in favor of using Lodestar in their daily work, but suggested several modifications to make the system more useful. For the sake of brevity, we focus primarily on summarizing their constructive feedback below (participant IDs start with "FP"):

Provide Clear Documentation & Context Our early prototypes did not include tooltips or descriptions of analysis steps. Several participants highlighted the need for increased transparency in the interface. Specifically, they wanted clearer naming conventions, documentation of features and methodologies (e.g., the difference between expert and crowd recommendations), and explanation of expected system behavior. For example, some participants had difficulties understanding the meaning of certain user interface elements. Participants asked questions such as "*what are these percentages?*" (FP6), or "*[what do] the columns on the left side represent?*" (FP5). Participants FP1, FP2, FP3, and FP5 also asked if there "*is actually a way to view the entire dataset?*" (FP2).

There were many questions specific to the meaning of recommendations. For example, FP3 said "*I think the names [are] misleading... there were some really complicated names for just a simple linear regression. [It] should just be changed [to more] obvious names.*" Similarly, FP2 suggested that there should be "*a longer description [...] [or] some way to show their effectiveness without the user having to Google search them.*" These misconceptions indicate that better documentation is needed to help new users understand the interface.

[1]The Udacity tutorial is available here: `https://github.com/sajal2692/data-science-portfolio/blob/master/boston_housing/boston_housing.ipynb`.

Improve Tracking of Analysis Progress Several participants wanted to be able to see what phase of the data science process they were in based on the current state of their analysis workflow. Our early prototypes did not include the feature to track previously selected analysis. FP4 drew parallels with a restaurant order tracker, where Lodestar should partition each part of the data science process into separate steps, and group analysis recommendations into these steps. Users would then be able to better understand their progress within the data science process.

Enable More Granular Control The early Lodestar prototype only allowed users to choose from pre-loaded datasets, and did not provide any export or customization functionality for analysis steps. However, multiple participants expressed the desire to import their own dataset and export their own code for later sharing and reuse. Participant FP4 said that they would be frustrated if they wanted to "*export it or make some changes in the data or [try] to do something that is not supported by Lodestar [while] not having any way of doing so.*" Participants also highlighted the need for more control over what parameters or attributes were being passed into different analysis steps, such as selecting specific attributes when generating visualizations or executing regression analyses. These observations suggest that users should be able to import their own dataset, customize analysis steps, and export their current analysis workflows.

4.1.4 Further Refinement of Lodestar

Though participants could see promise in providing automated recommendations (design requirement **D1**), the expressed need for more tracking of workflow structure and progress also reinforces design requirement **D2**. Without additional context to help users situate themselves within the broader data science process, users can easily lose their train of thought, hindering their analytic flow. The need for more documentation and control observed in our formative study supports design requirement **D3**. Without adequate information, users are unable to explore new data analysis techniques and interpret the results in Lodestar on their own. Users also find it difficult to tailor their explorations to their specific needs without access to the code.

These points of feedback served as motivation for additional iteration on the Lodestar feature design. Specific features that were added as a result of this study included the ability to export the user's notebook to an `.ipynb` file for use outside of the system, a visual tracker that displays the progress of the user's analysis in each output cell, showing which recommendations have been chosen so far, and descriptive tooltips of the different analysis techniques in each output cell.

5 System Overview

LODESTAR is *data analysis recommender*, i.e., a system that interactively suggests the next step to take in an analysis workflow. Lodestar is designed in the style of an interactive computational notebook, and generally inspired by the designs of existing notebooks such as Jupyter [24], Observable [23], and Google Colaboratory [18]. Given Python's broad popularity in data science contexts [47], we chose to focus on Python as our target computation environment.

Lodestar consists of four main components, shown in Figure 3: a browser-based *notebook interface*, an *interactive computing protocol*, a *recommendation engine* to suggest analysis steps, and a server-side *kernel* [24] to execute analysis steps. The protocol manages communication between the client and server (commands as well as computational results), and the kernel on the server side runs each analysis step that the user selects using an interpreter. The Flask server handles all of the client requests for data processing, analysis, and recommendations, with different endpoints.

Lodestar emphasizes an iterative workflow design where analysis steps are added progressively, one at a time, providing finer-grained control to the user. To help users focus more on analysis steps and best practices rather than low-level code, Lodestar allows the user to rapidly choose from a list of recommended analysis steps. These recommendations are displayed in the form of buttons, so a user can easily select and execute an analysis step of interest with a single click. Furthermore, these recommendations are mined from recent Python tutorials and active GitHub repositories of Jupyter Notebooks, enabling the user to construct new analysis workflows based on best practices in a data-driven way.

We describe the Lodestar notebook interface in more detail in section 6. We describe our process for extracting, curating, and recommending analysis steps in section 7.

6 Notebook Interface

The Lodestar interface (shown in Figure 1) is an interactive notebook providing a literate computing environment [30] that runs in a web browser on the client. Similar to existing notebooks, the Lodestar notebook is essentially a linear document that the user can selectively edit and execute. The interface contains three major components: a menu panel at the top, one or more notebook cells, and recommendation panels for each cell. The notebook cells and recommendation panels dynamically appear and update within the notebook interface in response to user interactions.

The user begins their analysis using the menu panel to load an existing dataset or a new dataset (in CSV format) into the system. Once a dataset has been loaded, Lodestar generates a recommendation panel within the notebook interface, providing the user with an initial set of recommended analysis steps. We refer to the actual code behind each analysis step as an *analysis block*, and the displayed result of executing the analysis step as a *notebook cell*. From this point onward, the analysis process forms a cycle that repeats until the user is satisfied with their new workflow:

1. The user **selects** an analysis step from a *recommendation panel*;
2. The kernel **executes the matching analysis block** on the server;
3. The notebook **displays the output** by appending a new cell; and
4. The notebook **generates a new panel of recommendations**, based on the user's previous selection.

When the user is ready to migrate their workflow to a complementary tool, for example to iterate on the code directly within a code editor, they can export the Lodestar workflow as a Jupyter notebook file.

6.1 Recommendation Panel

Every notebook cell in the Lodestar interface has an accompanying recommendation panel, allowing the user to extend their latest analysis step by one cell. When the user selects an analysis step from a recommendation panel, a new notebook cell is generated for the selected recommendation, along with a new recommendation panel underneath. Lodestar uses the output of the preceding notebook cell as the input for executing any analysis step selected in this recommendation panel. Each panel provides two sets of recommendations, one from a *crowd advisor* and one from an *expert advisor*. The crowd advisor sources recommendations from online data analysis repositories such as GitHub. The expert advisor sources recommendations from educational resources such as textbooks, online classes or online tutorials. We describe the Lodestar advisors in section 7.

If a user is unsatisfied with a given set of recommendations, they can choose from Lodestar's full catalog of analysis steps in a drop-down menu at the bottom of each recommendation panel. This list is available in the supplementary materials.

6.2 Notebook Cell

Once a selection is made in a recommendation panel, the selected analysis step is highlighted and the results are displayed in a new notebook cell, allowing the user to review their past selections and the corresponding results with each subsequent step. Furthermore, the user is able to go back and update the results at any time by selecting a different analysis step in any of the previous recommendation panels. Any cell can also be deleted, which triggers the removal of all downstream cells that depend on the deleted cell. In this way, Lodestar maintains a linear structure in the notebook, making it easier for users to navigate within the analysis workflow.

To help users understand the functionality of each recommended analysis step and its purpose within the context of the larger data science process, notebook cells consist of five tabs. Each tab describes the behavior of the analysis block represented by this notebook cell.

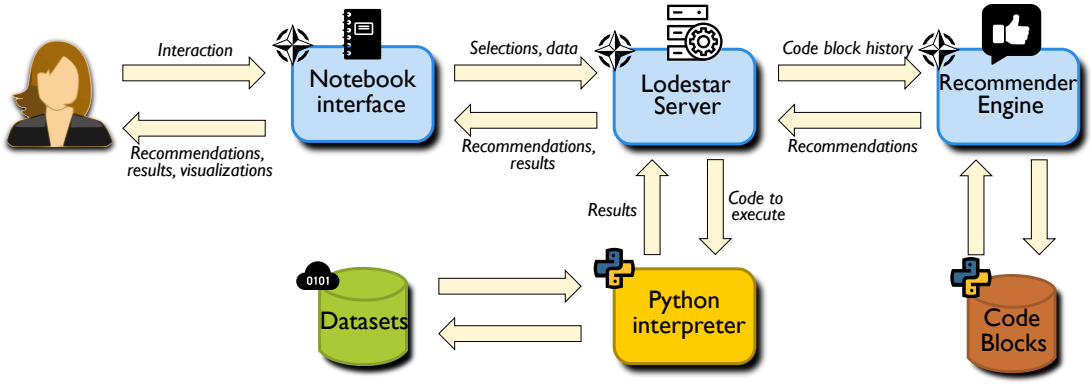

Fig. 3: Overview of the Lodestar architecture. The user interacts with the notebook interface and selects either a data set to bootstrap the notebook or an analysis step within a guided workflow. The notebook interface sends the selection as a request to the Lodestar server. The Lodestar server sends requests to the recommendation engine for subsequent recommendations based on current selections (data or analysis). Lodestar server also sends a request to the Python interpreter to execute any selected analysis. Results from both these requests are sent back from the Lodestar server to the notebook interface for the user to view and interact.

We refined the design of each tab based on the feedback we received from the formative study (see subsection 4.1):

- **Output Data Frame:** Default view that renders the output data frame produced by executing the analysis step as a table.
- **Analysis Results:** Displays the raw results produced by the analysis step (e.g., a `print` statement, or Seaborn visualization).
- **Script:** Displays the Python code within the corresponding analysis block.
- **"What's this Analysis?":** Provides a brief, high-level description of the analysis step.
- **Analysis Progress:** Displays the chain of analyses leading to the current analysis step, where each step has an intuitive name.

6.3 Exporting Code and Results

When the user is ready to migrate their analysis workflow to a related tool, they can export content directly from Lodestar. To export the code for a specific analysis step into an independent Jupyter notebook file, the user can click on the export button next to the *Code Script* tab of the corresponding cell. To export the entire analysis workflow, the user can click on the export button on the menu panel at the top of the interface. Similarly, Lodestar enables users to export the output data of any displayed notebook cell in the form of a CSV file. To do this, the user clicks on the export button next to the *Output Data Frame* tab. The user can also download the visualizations displayed in any notebook cell as separate PNG files.

7 Advisors and Recommendations

The Lodestar recommendation engine is based on the notion of an *advisor*: a source of analysis recommendations. Lodestar supports multiple advisors, each consisting of a library of analysis steps and a set of advisor-recommended transitions between analysis steps (i.e., a recommendation graph). In our current implementation, we use two advisors: a "crowd" advisor drawn from our semi-automatic code analysis, and an "expert" advisor drawn from the manual code curation. For each advisor, the recommendation panel will show a list of up to five recommendations, ordered by probability, or how frequently this analysis step came next in the respective recommendation graph.

In this section, we describe how we build our recommendation graphs for the expert and crowd advisors, and how we enable Lodestar to identify equivalent or related states across both graphs.

7.1 Recommendation Graph

Lodestar models transitions between analysis steps by treating analysis workflows (e.g., existing tutorials or computational notebooks) as paths taken through a network graph. Each node in the graph is an analysis step, and a directed edge appears in the graph for each pair of consecutive analysis steps observed in a workflow. Lodestar leverages

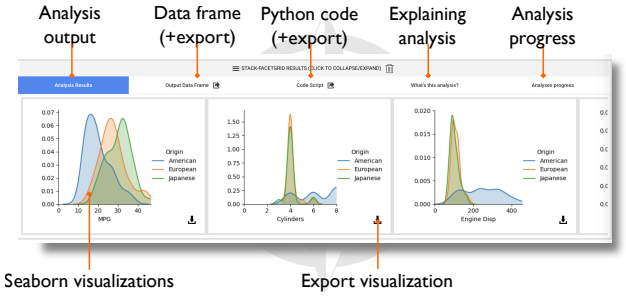

Fig. 4: Figure grid generated by an analysis block using the Seaborn statistical data visualization package for Python.

the relative frequency of these transitions to predict which analysis steps are likely to occur next. The particular graph structure used in Lodestar is a Markov chain, and the final computed graph we refer to as a *recommendation graph*.

Lodestar traverses the recommendation graph one state at a time for each user input (i.e., choice of analysis step). As a result, our recommendation approach does not require maintaining specific state about the analysis itself. Instead, the location in the Markov chain serves as state, and transitions (e.g., recommendations) thus depend only on the current state.

We can infer these recommendation graphs programmatically by mining analysis blocks (i.e., code snippets) from existing computational notebooks. In this case, the analysis blocks are used as the graph states, in place of their corresponding analysis steps. Figure 5 shows the general approach for mining analysis blocks into this recommendation graph. We extract the analysis blocks from existing computational notebooks and recover the transitions between states from the sequences observed in each notebook, with the weights signifying the frequency of observed transitions. Analysis blocks become nodes B_i in this graph, and edges represent probabilistic transitions $Pr(j|i) = P_{i,j}$, where the probabilities $P_{i,j}$ are taken from a stochastic matrix $\mathbb{P}$ that simply represents the frequency of transitions between blocks in the individual sequences.

To infer the full recommendation graph, we first construct a separate Markov chain for each notebook (or tutorial) identified as a source for our advisors. Specifically, we model each notebook as a Markov chain with one state per block and the transition probability to move from block B_i to the next block B_{i+1} for each time step (e.g., user input) expressed as $Pr(i+1|i) = 1$. Similar analysis steps are labeled with the same high-level identifier, representing a broader category of computation that transcends individual notebooks (e.g., $B1$, $B2$, etc. in

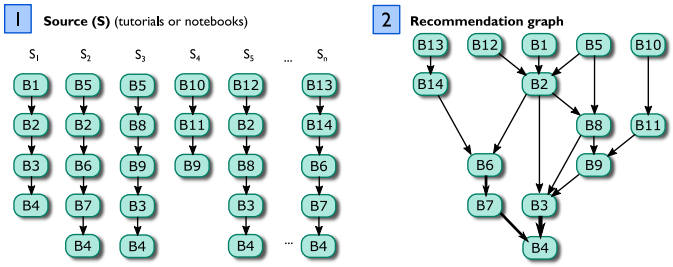

Fig. 5: Mining blocks into a recommendation graph representing a Markov chain. In Step 1 (left), sources $S_1, \ldots, S_n$ (manually curated or automatically extracted) yield (ordered) sequences of blocks $S_i = (B_1, \ldots, B_m)$. In Step 2, a recommendation graph can be derived by matching blocks that appear in multiple sequences and joining the sequences at those nodes. Edges between blocks in the graph are the frequency-weighted state transitions in the chain.

Figure 5). The result is a larger two-dimensional nested list, where each notebook is one row within the list (i.e., the left side of Figure 5), and each column a sequence of analysis step categories.

We can then merge the resulting sequences into a single graph (e.g., merging S_1, S_2, etc. in Figure 5), and aggregate the relative frequencies associated with the different categories to determine transition weights (i.e., how often do we see blocks from category B1 executed before blocks from category B2?).

Specifically, the transition probability $P_{i,j}$ (and thus edge weight in the recommendation graph) for the i^{th} row and the j^{th} column is the number of edges from B_i to B_j across all the sequences, divided by the out-degree of B_i. In other words, the graph will have no edges (weight 0) between blocks that never appeared in sequence, and will have normalized weights for blocks that fan out to multiple different destinations (because they are used by many notebooks). To bootstrap the recommendation, we recommend the first analysis in all the sequences (the root nodes in the graph).

7.2 Extracting Analysis Blocks for the Expert Advisor

We extracted analysis blocks for our expert advisor from online tutorials[2]. These tutorials were either Jupyter Notebooks or blogs which clearly delineated code from text. Analysis blocks correspond directly to code cells found in tutorial notebooks, or self-contained code snippets found in blog posts.

While there exist many data science resources online, their focus and depth varies widely, from simple hands-on learning for beginners (e.g., software installation, basic Python knowledge, and Jupyter functionality), to expert-level guides on deep learning, sensitivity analysis, and model building and tuning. As a rule, we picked individual resources focused on teaching how to complete a specific analysis task.

We narrowed our search to end-to-end data science examples, which provide concrete sequences of analysis steps along the data science pipeline. Specifically, we selected examples that have: an explicit purpose for the data analysis, step-by-step explanations and results, and runnable code. These requirements helped to ensure that the extracted analysis blocks will have similar functionality across examples.

7.3 Formatting Analysis Blocks for the Expert Advisor

To ensure that the extracted analysis blocks are executable in Lodestar, we also apply a separate code curation process. From our experience, each source has a specific analysis goal, and the blocks across different sources may use different libraries, data attributes, and variables to achieve it. For example, a tutorial using the Boston housing dataset, may generate a scatter to examine a linear relationship between four housing attributes, while in a school test-scores dataset it only makes sense to examine a linear relationship in between two attributes. This is useful nuance for manual analysis, but cannot be directly used in a generic data analysis system such as Lodestar. In other words, the analysis blocks must be curated—typically generalized—to be applicable across multiple applications.

The block curation process is idiosyncratic, but consists of the following steps: (1) adding missing dependencies, (2) replacing data-specific labels and attributes, (3) setting appropriate default parameters, and (4) generalizing code to operate on general data frames and output data frames too. This process is very similar to our curation strategy for recommendations from our "crowd" advisor. We manually compared new blocks to exiting blocks within the library, to ensure there were no duplicates. Upon completion of the curation process, each new analysis block is added to the library for the recommendation graph.

7.4 Managing Analysis Blocks for the Crowd Advisor

We extracted analysis blocks for our crowd advisor from a corpus of approximately 6,000 Jupyter notebooks, originally collected by Rule et al. [47][3]. We filtered out notebooks which did not contain import statements and API calls using common data science libraries, such as Numpy [64], Scikit-Learn [40], or Pandas [36]. We first partition each notebook into discrete analysis blocks. For Jupyter notebooks, the code is often already partitioned by the notebook authors through the use of Jupyter notebook code *cells*. Our straightforward approach is to identify existing cells in the Jupyter notebook corpus as separate analysis blocks for Lodestar.

Our key insight for this process is that *similar data analysis steps often use similar API calls* in the code. Using this idea, we construct a term vector to represent each analysis block, where the vector represents the normalized frequency of each API call that appears within the block. Each cell in the vector represents a unique API call observed in *any* notebook in the dataset, allowing the vectors for any analysis block to be compared with any other block in the dataset.

We use these term vectors to cluster the analysis blocks. Specifically, the normalized vectors are passed to a k-means clustering algorithm to be clustered for similarity. After some iteration, we identified 200 clusters as an ideal number for grouping the analysis blocks extracted from our corpus (please see our supplemental materials for more details). Each resulting cluster represents a set of analysis blocks that share similarities in functionality, and thus could also represent a shared or synonymous analysis step across the corresponding Jupyter notebooks.

Of the 200 representatives (one for each cluster), we ultimately selected 22 blocks as a starting set for the Lodestar library. For any given cluster, Lodestar needs a way of recommending a single analysis block to users that represents the corresponding analysis step. We use code-line count as a heuristic to pick a representative analysis blockfrom each cluster. Specifically, we pick the blocks which have a median number of lines relative to all other blocks within a cluster.

Blocks for both the crowd and expert advisors are formatted to follow the same consistent structure assumed by the Lodestar system. We format each analysis block to be a Python function, include necessary imports, convert the function's input and output to a data frame, and remove print statements and irrelevant comments.

7.5 Identifying Synonymous States Across Advisors

Of course, managing multiple advisors means that the system must track the state of the analysis in the recommendation graph for *all* advisors when the user selects a recommendation from a specific advisor. Our current solution uses a multi-level tagging mechanism where each block is manually tagged given its functionality; for example, a decision tree block could be tagged with `train-model` and `test-model`. Tags correspond to steps in the data analysis workflow. We developed an understanding of these steps using previous studies [5, 20, 25, 73, 74]. Much like Yan et al. [73], we cast particular Python APIs to specific analysis steps. For example, Pandas *dropna* function was cast as a data-cleaning operation since dropping

[2]Please see our supplemental materials for a detailed report on our full process for extracting and curating analysis blocks for the expert advisor: `https://osf.io/3gpsy/`

[3]Please see our supplemental materials for a detailed report on our full process for extracting and curating analysis blocks for the crowd advisor.

empty elements is a common way to clean data. Our tags include: `statistical-sampling`, `visualization`, `data-organization`, `data-cleaning`, `data-formatting` and `statistical-summary`.

In tagging analysis in this way, we allow for matching the new state of the specific advisor, chosen by the user, to relevant states in the other advisors. More specifically, if the user chooses a recommendation from the expert advisor that suggests running a specific decision tree block, the Lodestar engine will advance the crowd advisor to a state in its recommendation graph that corresponds to the `train-model` and `test-model` tags. This design, as well as ordering recommendations by probability ordering, allows Lodestar() to guide best practices.

The same functionality is used when the user eschews all of the recommendations and instead selects directly from the library through the drop-down box in the recommendation panel. In this case, all of the advisor models will be advanced to the appropriate state matching the block that the user executed. This allows the user to iterate and sandbox different techniques, unhindered by a guided system. Though, this is the limit to manual user control that Lodestar supports.

8 Discussion

We have presented Lodestar, a computational notebook for rapid experimentation and learning of new data science practices. Instead of forcing fledgling analysts to search for and apply relevant data analysis methods by hand, Lodestar recommends suitable next steps for the current workflow using both manually curated as well as automatically crowd-sourced guidance. Our work on Lodestar has uncovered several interesting discussion points: the prospect for data science for novices, the actual "wisdom" of crowd recommendations, and alternate recommendation mechanisms.

8.1 Data Science for Non-Experts

The real power of Lodestar lies not in its data sources, which are publicly available to anyone online, but in its ability to synthesize the knowledge from these diverse sources into a single unified model. By sharing this knowledge in the form that data scientists are most familiar—Python (or R) source code—Lodestar provides reusable building blocks that can easily be transferred across data science workflows.

However, for the tool to be truly effective for its purpose, the library of analysis blocks must be expanded and drawn from a large set of sources. For example, new data sources could be incorporated to customize Lodestar for specific disciplines such as bio-informatics, computational journalism, and computer vision. Lodestar's advisor model may be one way to support this; instead of the "expert" vs. "crowd" dichotomy that our current implementation uses, a more robust implementation could support a plethora of pluggable advisors drawn from a central repository. In this way, the advisors, analysis blocks, and library could be community-driven and improved by anyone.

Choosing an analysis step or interpreting results in our current prototype still requires baseline data science knowledge, such as from a university data science course (indeed, all our participants had this). However, the Lodestar approach does alleviate lack of *expertise* in data science practice, which is often the case for academic learning.

8.2 On the "Wisdom of the Crowd" for Data Analysis

While we are excited about the prospects of the "wisdom of the crowd" [57] for data science and analysis, it has become clear that this is an area that will require significantly more work. For example, our current approach is not entirely automated; manual curation is still required in choosing a representative block from the clustering analysis and in editing the block into the appropriate form that Lodestar expects, including eliminating side effects, removing output statements, and resolving dependencies. We plan to automate these steps in the future.

The need for manual curation, or at least review, is exacerbated by the fact that a significant portion of the code we analyzed in Rule et al.'s Jupyter notebook corpus [47] was of low quality: some notebooks had cells with a single line of code, or all of the source code in a single cell. Many had non-functional code, syntax errors, or code that was never used. While we have filtered these notebooks from our analysis, the signal-to-noise ratio in crowdsourced code is often low.

The remedy for many of these challenges can often be found in sheer scale. While we studied the "sampler" dataset containing 6,530 notebooks in this paper, the full 600 GB dataset contains more than 1.25 million notebooks. With access to this many examples, we could afford to discard more problematic ones. Furthermore, frequency of use would help ensure that best practices are easier to identify. Of course, a dataset of this size brings with it a new set of scalability challenges. Existing data processing [39] and code analysis [15, 16] techniques could help address this big data challenge in the future.

8.3 Different Recommendation Strategies

The Lodestar recommendation engine is based on Markov chains, which are useful for representing a sequence of chained states or commands, as in a data science script. However Markov chains may oversimplify the relationships between analysis steps and data science users in some ways. It would be interesting to study how to use more sophisticated methods as part of the Lodestar recommendation engine. For example, state-of-the-art recommender systems tend to be organized into collaborative filtering, content-based filtering, and hybrid filtering [2]. Collaborative filtering is based on a social view of recommendation, where behavior by other users such as navigation, ratings, and their personal traits are used to match content to a specific user. In the case of Lodestar, this would enable the historical preferences of Lodestar users to guide other users. For content-based filtering, recommendations can be derived by comparing items to recommend with user preferences and auxiliary information. This approach could enable Lodestar users to be matched to specific analysis steps based on, e.g., workflows they have created in the past, specific data types, and metadata for existing datasets and code. Finally, we could combine methods to develop new hybrid recommendation strategies.

A recent development in artificial intelligence is to build recommender systems using deep learning techniques (or *deep recommenders*) [4, 75], particularly for content-based approaches. Given our large available corpus of potential training data, unsupervised methods such as Recurrent Neural Networks [17] could prove useful, since they are ideal for sequential data. The Lodestar advisor model provides a useful framework from which to incorporate and merge future recommendation strategies for data science. However, these topics are beyond the scope of this paper.

8.4 Limitations and Future Work

Two participants from our formative study suggested that they would either appreciate being able to toggle off the recommendations or view them all without a filter. Based on these comments, it remains unclear how effective a code-free recommendation environment can be in teaching data science best practices. Thus, in our future work, we will test the strength of Lodestar recommendations and its effectiveness in teaching novice data scientists new techniques.

Due to the many challenges of automatic code analysis, we currently do not allow users to write their own code directly in Lodestar, or even to modify existing code. To make online code editing possible, we would need an automatic classification process that could determine how new code fits into the recommendation graph so that the system could resume the analysis with new recommendations after manual code block. Such live updates to the recommender are not currently part of Lodestar, but are an interesting direction for future work.

We made several design decisions to the Lodestar notebook that will need to be revisited for a general implementation.

Lodestar currently does not consider specifics about each input dataset while making recommendations—only display recommendations which do not programmatically fail to execute on the selected dataset. This is a point of future work.

All of our analysis blocks take a Pandas data frame as input, and generate a new data frame as output. Also, other disciplines use other data representations, and some computations may require passing multiple data objects as arguments. To address these limitations, we look to improving our existing design and thoroughly evaluating these improvements in our future work.

REFERENCES

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. TensorFlow: A system for large-scale machine learning. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation*, pp. 265–283. USENIX, Berkeley, CA, USA, 2016.

[2] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17(6):734–749, 2005. doi: 10.1109/TKDE.2005.99

[3] S. K. Badam, A. Mathisen, R. Rädle, C. N. Klokmose, and N. Elmqvist. Vistrates: A component model for ubiquitous analytics. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):586–596, 2019. doi: 10.1109/TVCG.2018.2865144

[4] O. Bar El, T. Milo, and A. Somech. Automatically generating data exploration sessions using deep reinforcement learning. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pp. 1527—-1537. ACM, New York, NY, USA, 2020. doi: 10.1145/3318464.3389779

[5] L. Battle and J. Heer. Characterizing exploratory visual analysis: A literature review and evaluation of analytic provenance in tableau. *Computer Graphics Forum*, 38(3):145–159, 2019. doi: 10.1111/cgf.13678

[6] J. Bertin. *Semiology of Graphics: Diagrams, Networks, Maps*. University of Wisconsin Press, Madison, WI, USA, 1983.

[7] M. Bostock and J. Heer. Protovis: A graphical toolkit for visualization. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1121–1128, 2009. doi: 10.1109/TVCG.2009.174

[8] M. Bostock, V. Ogievetsky, and J. Heer. D^3: Data-driven documents. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2301–2309, 2011. doi: 10.1109/TVCG.2011.185

[9] W. S. Cleveland and R. McGill. Graphical perception: Theory, experimentation, and application to the development of graphical methods. *Journal of the American Statistical Association*, 79(387):531–554, 1984. doi: 10.1080/01621459.1984.10478080

[10] M. Conlen and J. Heer. Idyll: A markup language for authoring and publishing interactive articles on the web. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, pp. 977–989. ACM, New York, NY, USA, 2018. doi: 10.1145/3242587.3242600

[11] Z. Cui, S. K. Badam, A. Yalçin, and N. Elmqvist. DataSite: Proactive visual data exploration with computation of insight-based recommendations. *Information Visualization*, 18(2):251–267, 2019. doi: 10.1177/1473871618806555

[12] Ç. Demiralp, P. J. Haas, S. Parthasarathy, and T. Pedapati. Foresight: Recommending visual insights. *Proceedings of the Very Large Database Endowment*, 10(12):1937–1940, 2017. doi: 10.14778/3137765.3137813

[13] I. Drosos, T. Barik, P. J. Guo, R. DeLine, and S. Gulwani. Wrex: A unified programming-by-example interaction for synthesizing readable code for data scientists. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, CHI '20, p. 1–12. Association for Computing Machinery, New York, NY, USA, 2020. doi: 10.1145/3313831.3376442

[14] J. Drozdal, J. Weisz, D. Wang, G. Dass, B. Yao, C. Zhao, M. Muller, L. Ju, and H. Su. Trust in automl: Exploring information needs for establishing trust in automated machine learning systems. In *Proceedings of the 25th International Conference on Intelligent User Interfaces*, IUI '20, p. 297–307. Association for Computing Machinery, New York, NY, USA, 2020. doi: 10.1145/3377325.3377501

[15] E. L. Glassman, J. Scott, R. Singh, P. J. Guo, and R. C. Miller. OverCode: Visualizing variation in student solutions to programming problems at scale. *ACM Transactions on Computer-Human Interaction*, 22(2):7:1–7:35, 2015. doi: 10.1145/2699751

[16] E. L. Glassman, T. Zhang, B. Hartmann, and M. Kim. Visualizing api usage examples at scale. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*, p. 1–12. ACM, New York, NY, USA, 2018. doi: 10.1145/3173574.3174154

[17] I. J. Goodfellow, Y. Bengio, and A. C. Courville. *Deep Learning*. Adaptive Computation and Machine Learning. MIT Press, Cambridge, MA, USA, 2016.

[18] Google. Google Colaboratory. `https://colab.research.google.com/`, 2020.

[19] A. Head, J. Jiang, J. Smith, M. A. Hearst, and B. Hartmann. Composing flexibly-organized step-by-step tutorials from linked source code, snippets, and outputs. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, CHI '20, p. 1–12. Association for Computing Machinery, New York, NY, USA, 2020. doi: 10.1145/3313831.3376798

[20] J. Heer, J. Mackinlay, C. Stolte, and M. Agrawala. Graphical histories for visualization: Supporting analysis, communication, and evaluation. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1189–1196, 2008.

[21] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems*, 22(1):5–53, 2004. doi: 10.1145/963770.963772

[22] K. Hu, M. A. Bakker, S. Li, T. Kraska, and C. Hidalgo. Vizml: A machine learning approach to visualization recommendation. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI '19, p. 1–12. Association for Computing Machinery, New York, NY, USA, 2019. doi: 10.1145/3290605.3300358

[23] O. Inc. Observable. `https://observablehq.com`, 2020.

[24] P. Jupyter. Jupyter Notebook. `https://jupyter.org/`, 2019.

[25] S. Kandel, A. Paepcke, J. Hellerstein, and J. Heer. Wrangler: Interactive visual specification of data transformation scripts. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*, pp. 3363–3372. ACM, New York, NY, USA, 2011.

[26] S. Kandel, A. Paepcke, J. M. Hellerstein, and J. Heer. Enterprise data analysis and visualization: An interview study. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2917–2926, 2012. doi: 10.1109/TVCG.2012.219

[27] M. B. Kery, D. Ren, F. Hohman, D. Moritz, K. Wongsuphasawat, and K. Patel. *Mage: Fluid Moves Between Code and Graphical Work in Computational Notebooks*, p. 140–151. Association for Computing Machinery, New York, NY, USA, 2020.

[28] N. W. Kim, E. Schweickart, Z. Liu, M. Dontcheva, W. Li, J. Popovic, and H. Pfister. Data-driven guides: Supporting expressive design for information graphics. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):491–500, 2017. doi: 10.1109/TVCG.2016.2598620

[29] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. E. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. B. Hamrick, J. Grout, S. Corlay, P. Ivanov, D. Avila, S. Abdalla, and C. Willing. Jupyter notebooks – a publishing format for reproducible computational workflows. In *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, pp. 87–90. IOS Press, Amsterdam, Netherlands, 2016. doi: 10.3233/978-1-61499-649-1-87

[30] D. E. Knuth. Literate programming. *The Computer Journal*, 27(2):97–111, 1984. doi: 10.1093/comjnl/27.2.97

[31] S. Kross and P. J. Guo. *Practitioners Teaching Data Science in Industry and Academia: Expectations, Workflows, and Challenges*, p. 1–14. Association for Computing Machinery, New York, NY, USA, 2019.

[32] Z. Liu, J. Thompson, A. Wilson, M. Dontcheva, J. Delorey, S. Grigg, B. Kerr, and J. Stasko. Data illustrator: Augmenting vector design tools with lazy data binding for expressive visualization authoring. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*, pp. 1—-13. Association for Computing Machinery, New York, NY, USA, 2018. doi: 10.1145/3173574.3173697

[33] J. Mackinlay. Automating the design of graphical presentations of relational information. *ACM Transactions on Graphics*, 5(2):110–141, 1986. doi: 10.1145/22949.22950

[34] J. Mackinlay, P. Hanrahan, and C. Stolte. Show me: Automatic presentation for visual analysis. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1137–1144, 2007. doi: 10.1109/TVCG.2007.70594

[35] A. Mathisen, T. Horak, C. N. Klokmose, K. Grønbæk, and N. Elmqvist. Insideinsights: Integrating data-driven reporting in collaborative visual analytics. *Computer Graphics Forum*, 38(3):649–661, 2019. doi: 10.1111/cgf.13717

[36] W. McKinney. pandas: a foundational Python library for data analysis and statistics. *Python for High Performance and Scientific Computing*, 14, 2011.

[37] G. G. Méndez, M. A. Nacenta, and S. Vandenheste. iVoLVER: Interactive visual language for visualization extraction and reconstruction. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*, pp. 4073–4085. ACM, New York, NY, USA, 2016. doi: 10.1145/2858036.2858435

[38] D. Moritz, C. Wang, G. L. Nelson, H. Lin, A. M. Smith, B. Howe, and J. Heer. Formalizing visualization design knowledge as constraints: Actionable and extensible models in Draco. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):438–448, 2019. doi: 10.1109/TVCG.2018.2865240

[39] S. Mudgal, H. Li, T. Rekatsinas, A. Doan, Y. Park, G. Krishnan, R. Deep, E. Arcaute, and V. Raghavendra. Deep learning for entity matching: A design space exploration. In *Proceedings of the ACM Conference on Management of Data*, pp. 19–34. ACM, New York, NY, USA, 2018. doi: 10.1145/3183713.3196926

[40] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12(Oct):2825–2830, 2011.

[41] D. B. Perry, B. Howe, A. M. F. Key, and C. Aragon. VizDeck: Streamlining exploratory visual analytics of scientific data. In *Proceedings of the iConference*, pp. 338–350. iSchools, Fort Worth, TX, 2013. doi: 10.9776/13206

[42] P. Pirolli and S. Card. The sensemaking process and leverage points for analyst technology as identified through cognitive task analysis. In *Proceedings of the International Conference on Intelligence Analysis*, vol. 5, pp. 2–4. The MITRE Corporation, McLean, VA, USA, 2005.

[43] R. Rädle, M. Nouwens, K. Antonsen, J. R. Eagan, and C. N. Klokmose. Codestrates: Literate computing with webstrates. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, pp. 715–725. ACM, New York, NY, USA, 2017. doi: 10.1145/3126594.3126642

[44] D. Ren, T. Höllerer, and X. Yuan. iVisDesigner: Expressive interactive design of information visualizations. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2092–2101, 2014. doi: 10.1109/TVCG.2014.2346291

[45] D. Ren, B. Lee, and M. Brehmer. Charticulator: Interactive construction of bespoke chart layouts. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):789–799, 2019. doi: 10.1109/TVCG.2018.2865158

[46] S. F. Roth, J. Kolojejchick, J. Mattis, and J. Goldstein. Interactive graphic design using automatic presentation knowledge. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*, p. 112–117. Association for Computing Machinery, New York, NY, USA, 1994. doi: 10.1145/191666.191719

[47] A. Rule, A. Tabard, and J. D. Hollan. Exploration and explanation in computational notebooks. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*, pp. 32:1–32:12. ACM, New York, NY, USA, 2018. doi: 10.1145/3173574

[48] D. M. Russell, M. J. Stefik, P. Pirolli, and S. K. Card. The cost structure of sensemaking. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*, pp. 269–276. ACM, New York, NY, USA, 1993. doi: 10.1145/169059.169209

[49] B. Saket, H. Kim, E. T. Brown, and A. Endert. Visualization by demonstration: An interaction paradigm for visual data exploration. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):331–340, 2017. doi: 10.1109/TVCG.2016.2598839

[50] A. Satyanarayan and J. Heer. Lyra: An interactive visualization design environment. *Computer Graphics Forum*, 33(3):351–360, 2014. doi: 10.1111/cgf.12391

[51] A. Satyanarayan, D. Moritz, K. Wongsuphasawat, and J. Heer. Vega-Lite: A grammar of interactive graphics. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):341–350, 2017. doi: 10.1109/TVCG.2016.2599030

[52] A. Satyanarayan, R. Russell, J. Hoffswell, and J. Heer. Reactive Vega: A streaming dataflow architecture for declarative interactive visualization. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):659–668, 2016. doi: 10.1109/TVCG.2015.2467091

[53] J. Seo and B. Shneiderman. A rank-by-feature framework for interactive exploration of multidimensional data. *Information Visualization*, 4(2):96–113, 2005. doi: 10.1057/palgrave.ivs.9500091

[54] T. Siddiqui, A. Kim, J. Lee, K. Karahalios, and A. Parameswaran. Effortless data exploration with zenvisage: An expressive and interactive visual analytics system. *Proceedings of the Very Large Database Endowment*, 10(4):457–468, 2016. doi: 10.14778/3025111.3025126

[55] A. Srinivasan, S. M. Drucker, A. Endert, and J. Stasko. Augmenting visualizations with interactive data facts to facilitate interpretation and communication. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):672–681, 2019. doi: 10.1109/TVCG.2018.2865145

[56] C. Stolte, D. Tang, and P. Hanrahan. Polaris: A system for query, analysis, and visualization of multidimensional relational databases. *IEEE Transactions on Visualization and Computer Graphics*, 8(1):52–65, 2002. doi: 10.1109/2945.981851

[57] J. Surowiecki. *The Wisdom of Crowds: Why the Many are Smarter Than the Few and How Collective Wisdom Shapes Business, Economies, Societies, and Nations*. Anchor Books, New York, NY, USA, 2004.

[58] A. Tabard, W. E. Mackay, and E. Eastmond. From individual to collaborative: the evolution of Prism, a hybrid laboratory notebook. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work*, pp. 569–578. ACM, New York, NY, USA, 2008. doi: 10.1145/1460563.1460653

[59] B. Tang, S. Han, M. L. Yiu, R. Ding, and D. Zhang. Extracting top-k insights from multi-dimensional data. In *Proceedings of the ACM Conference on Management of Data*, pp. 1509–1524. ACM, New York, NY, USA, 2017. doi: 10.1145/3035918.3035922

[60] J. J. Thomas and K. A. Cook, eds. *Illuminating the Path: The Research and Development Agenda for Visual Analytics*. IEEE Computer Society Press, 2005.

[61] E. Tufte. The visual display of quantitative information, 2001.

[62] M. Ufford, M. Pacer, M. Seal, and K. Kelley. Beyond interactive: Notebook innovation at Netflix. `https://medium.com/netflix-techblog/notebook-innovation-591ee3221233`, 2018.

[63] S. van den Elzen and J. J. van Wijk. Small multiples, large singles: A new approach for visual data exploration. *Computer Graphics Forum*, 32(3pt2):191–200, 2013. doi: 10.1111/cgf.12106

[64] S. Van Der Walt, S. C. Colbert, and G. Varoquaux. The numpy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22, 2011.

[65] J. VanderPlas, B. E. Granger, J. Heer, D. Moritz, K. Wongsuphasawat, A. Satyanarayan, E. Lees, I. Timofeev, B. Welsh, and S. Sievert. Altair: Interactive statistical visualizations for python. *The Journal of Open Source Software*, 3(32), 2018.

[66] M. Vartak, S. Madden, A. Parameswaran, and N. Polyzotis. SeeDB: Automatically generating query visualizations. *Proceedings of the Very Large Database Endowment*, 7(13):1581–1584, 2014. doi: 10.14778/2733004.2733035

[67] H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer, New York, NY, USA, 2016. doi: 10.1007/978-3-319-24277-4

[68] K. Wongsuphasawat, D. Moritz, A. Anand, J. Mackinlay, B. Howe, and J. Heer. Voyager: Exploratory analysis via faceted browsing of visualization recommendations. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):649–658, 2016. doi: 10.1109/TVCG.2015.2467191

[69] K. Wongsuphasawat, Z. Qu, D. Moritz, R. Chang, F. Ouk, A. Anand, J. Mackinlay, B. Howe, and J. Heer. Voyager 2: Augmenting visual analysis with partial view specifications. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*, pp. 2648–2659. ACM, New York, NY, USA, 2017. doi: 10.1145/3025453.3025768

[70] J. Wood, Kachkaev, and J. Dykes. Design exposition with literate visualization. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):759–768, 2019. doi: 10.1109/TVCG.2018.2864836

[71] H. Xia, N. H. Riche, F. Chevalier, B. R. D. Araújo, and D. Wigdor. DataInk: Direct and creative data-oriented drawing. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*, pp. 223:1–223:13. ACM, New York, NY, USA, 2018. doi: 10.1145/3173574

[72] M. A. Yalçin, N. Elmqvist, and B. B. Bederson. Keshif: Rapid and expressive tabular data exploration for novices. *IEEE Transactions on Visualization and Computer Graphics*, 24(8):2339–2352, 2017. doi: 10.1109/TVCG.2017.2723393

[73] C. Yan and Y. He. Auto-suggest: Learning-to-recommend data preparation steps using data science notebooks. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pp. 1539—-1554. Association for Computing Machinery, New York, NY, USA, 2020. doi: 10.1145/3318464.3389738

[74] E. Zgraggen, Z. Zhao, R. Zeleznik, and T. Kraska. Investigating the effect of the multiple comparisons problem in visual analysis. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*, pp. 1—-12. Association for Computing Machinery, New York, NY, USA, 2018. doi: 10.1145/3173574.3174053

[75] S. Zhang, L. Yao, and A. Sun. Deep learning based recommender system: A survey and new perspectives. *ACM Computing Surveys*, 52(1):5:1–5:35, 2018. doi: 10.1145/3285029