Towards integrated, interactive, and extensible text data analytics with LEAM

Peter Griggs*
MIT
pgriggs@mit.edu

Çağatay Demiralp*
Sigma Computing
cagatay@sigmacomputing.com

Sajjadur Rahman
Megagon Labs
sajjadur@megagon.ai

Abstract

From tweets to product reviews, text is ubiquitous on the web and often contains valuable information for both enterprises and consumers. However, the online text is generally noisy and incomplete, requiring users to process and analyze the data to extract insights. While there are systems effective for different stages of text analysis, users lack extensible platforms to support interactive text analysis workflows end-to-end. To facilitate integrated text analytics, we introduce LEAM, which aims at combining the strengths of spreadsheets, computational notebooks, and interactive visualizations. LEAM supports interactive analysis via GUI-based interactions and provides a declarative specification language, implemented based on a visual text algebra, to enable user-guided analysis. We evaluate LEAM through two case studies using two popular Kaggle text analytics workflows to understand the strengths and weaknesses of the system.

1 Introduction

The growth of e-commerce has contributed to the proliferation of digital text, particularly user-generated text (reviews, Q&As, discussions), which often contain useful information for improving the services and products on the web. Enterprises increasingly adopt text mining technologies to extract, analyze, and summarize information from such unstructured text data. However, online text collections are incomplete, ambiguous, and often sparse in informational content. Cleaning, featurizing, modeling, visualizing, extracting information from, and identifying topics in such text collections can be daunting and time-consuming without integrated systems that take the whole text analytics pipeline into account.

The characteristics of online text make interactive workflows and visualizations essential for rapid iterative analysis (Ittoo et al., 2016). Therefore we focus on visual interactive text analysis (VITA hereafter) and related systems. Few commercial and open-source tools can support different stages of VITA, *e.g.*, spreadsheets, computational notebooks, and visualization tools (Liu et al., 2018; Smith et al., 2020). Customized visual text analytics tools focus on specific use-cases like review exploration (Zhang et al., 2020a), sentiment analysis (Kucher et al., 2018), and text summarization (Carenini et al., 2006). None of these solutions accommodate the inherently cyclic, trial-and-error-based nature of VITA pipelines end-to-end (Drosos et al., 2020; Wu et al., 2020).

Designing and building VITA systems can be difficult. The primary challenge is the number and diversity of the tasks that need to be supported. Programmatic tools such as computational notebooks can provide extensibility and expressivity to incrementally build such support but they often lack in interactivity and do not facilitate direct data manipulation, impeding analysis.

In response, we propose LEAM , that provides an integrated environment for VITA. LEAM combines the advantages of spreadsheets, computational notebooks, and visualization tools by integrating a Code Editor with interactive views of raw (Data View) and transformed data (Chart View). Figure 1 shows a snapshot of LEAM. A key component in the design of LEAM is the instrumentation of text analysis operations via VITAL, a python API. These built-in operations can also be used directly from the interactive Operations Menu. To evaluate LEAM, we conduct two case studies using two popular Kaggle text analytics workflows. The

DaSH-LA 2021, June 11, 2021, Virtual Conference.

*Work done while authors were at Megagon Labs.

Proceedings of the 2nd Workshop on Data Science with Human in the Loop: Language Advances, pages 52–58

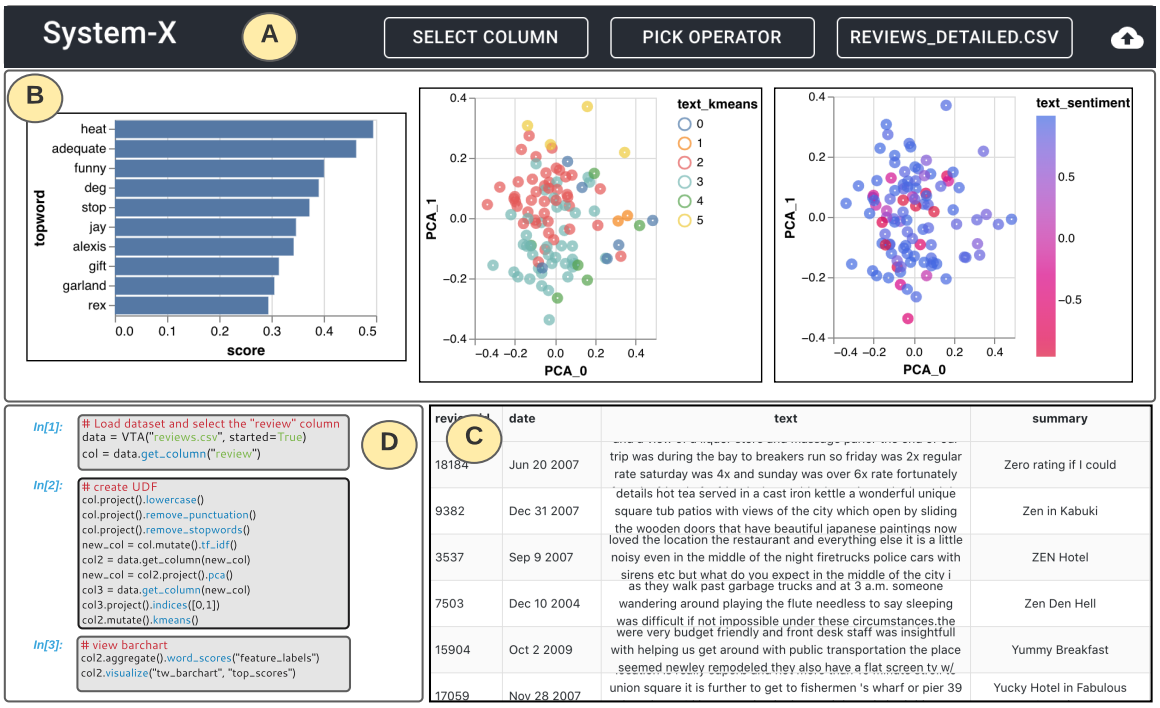


Figure 1: LEAM user interface. (A) Operations Menu enables users to perform visual interactive text analytics (VITA) operations using drop-down menus, (B) Chart View holds a carousel of interactive visualizations created by users, (C) Data View displays the data and its subsequent transformations, and (D) Code Editor allows users to compose and run VITA operations using a declarative specification called VITAL.

study showed that participants preferred the integrated analysis environment and the ability to specify various workflows both interactively (via Operations Menu) and declaratively (via VITAL). However, participants asked for enhanced workflow transparency and consistency of operations. We have released the source-code of LEAM at `https://github.com/megagonlabs/leam`.

2 Related Work

LEAM draws from prior work on interactive text analysis, computational notebook, and declarative specification of analysis workflows.

Interactive visual text analytics. Prior research on visual text analytics have limitations in flexibility and extensibility due to their fixed choices of models, visualizations, and interactions (Kucher et al., 2018; Liu et al., 2018). LEAM adopts the vision of a VITA system outlined in our prior work (Rahman et al., 2020). In this paper, we primarily focus on expressivity (*e.g.*, declarative workflow specification), resusability (*e.g.*, reusing operators and models), on-demand coordination (*e.g.*, linking visualizations and data), and transparency (*e.g.*, GUI interaction logging).

Computational notebooks. Computational notebooks such as Jupyter (Jupyter, 2020) allow programmers to interleave code with visualizations. This linear layout often introduces a physical distance between related charts, limiting an analyst's ability to derives insights by visually comparing different charts. Tools like B2 (Wu et al., 2020) and LUX (Lee, 2020), provide a non-linear interface where charts are placed in a separate visualization pane. While LEAM shares the same principle, it additionally features a Data View and enables coordination between visualization and the data—a desirable property of such interactive programming environments (Chattopadhyay et al., 2020).

Declarative data analysis and visualization. Prior work on data analysis workflow specification focused on several different stages, from data cleaning to exploration. To support data cleaning, Wrangler (Kandel et al., 2011) combines a mixed-initiative interface with a declarative transformation language. Text Extension python library (Codait, 2021) enables users to operate on intermediate data, *e.g.*, spans and tensors, in all phases of an NLP workflow. Grammars of graphics like Vega-Lite (Satyanarayan et al., 2016) and ggplot2 (Wick-

ham, 2016) support visualization specification via abstractions, *e.g.*, JSON. However, users cannot dynamically add new interactions to the visualizations using these abstractions. LEAM enables users to add new interactions to visualizations and create coordination among data and visualizations on-the-fly using declarative specifications developed based on grammar for visual text analysis introduced in our prior work (Rahman et al., 2020).

3 Design Considerations

We now outline our design considerations for creating LEAM. Table 1 shows which of these design considerations are supported by existing tools discussed in Section 2. These design considerations were informed by prior work on identifying challenges related to live programming interfaces (Chattopadhyay et al., 2020; Rule et al., 2018; Kery et al., 2020), studies on exploratory data science practices (Alspaugh et al., 2018; Kery et al., 2018; Zhang et al., 2020b), and guidelines for multiple coordinated view design (Wang et al., 2000), and refined through our experiences working with user-generated text data at MEGAGON LABS:

Design Crietria	Notebooks			Visualization Platforms	VITA (LEAM)
	Jupyter	LUX	B2		
D1/D2. Code	✓	✓	✓	x	✓
D1. Visualization	✓	✓	✓	✓	✓
D1. Data	x	x	x	x	✓
D3. On-demand Coordination	x	x	x	x	x
D4. Reusability	x	x	x	x	✓
D5. Transparency	x	x	✓	x	x

Table 1: Unlike existing tools, LEAM supports all of the design considerations ($D1 - D5$) outlined in Section 3.

D1. Enable integrated analytics. VITA systems should provide a single platform where users can directly manipulate (spreadsheets) and visualize (visualization tools) data while writing codes (notebooks) without context switching between tools.

D2. Specify operations declaratively. VITA systems should provide an expressive specification language to represent and communicate the entire breadth of workflows within the domain.

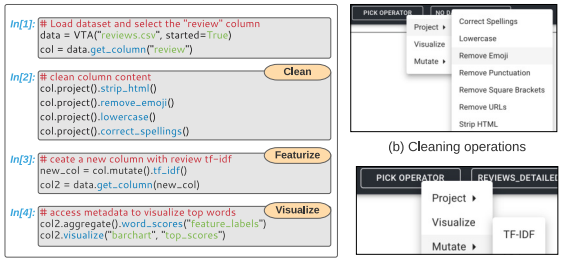
D3. Facilitate on-demand coordination. Within an integrated environment, VITA systems should enable users to specify coordination between all the available views on demand.

D4. Ensure reusability of operations. Users should be able to craft their analysis pipeline and share and reuse the workflow across use-cases.

D5. Ensure transparency of operations. VITA systems should ensure transparency of interactions on the interface—effect of direct manipulation and programmatic interactions should be immediately visible via visual cues or prompts.

4 LEAM User Interface

The four key components of the interface are a Code Editor, an Operations Menu, a Data View, and a Chart View. We discuss how these components enable integrated visual text analysis (**D1**).



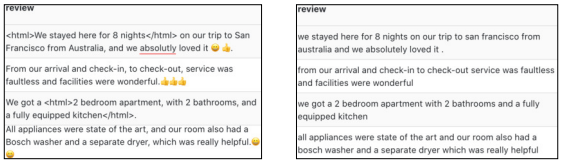
(a) VITAL commands (b) Cleaning operations (c) Featurization operations

Figure 2: (a) Users writes scripts in Code Editor using the VITAL API for cleaning, featurizing, and visualizing data. Alternatively, users can also utilize the operators in Operations Menu, *e.g.*, cleaning (b) and featurization (c).

Code Editor and Operations Menu. While the Code Editor design (see Figure 1C) is inspired by computational notebooks, it only supports writing, editing, and executing scripts—visualizations and data tables are displayed separately in Chart View and Data View, respectively. The multi-view representation is intended to help users relate their workflows with the underlying data and their visualizations—a benefit of multiple coordinated views. Users can write scripts in the Code Editor in Python. We also implement a Python-based visual interactive text analysis library, VITAL, for issuing various text analysis and visualization operations in the Code Editor (discussed in Section 5). These operations are derived from an algebra for visual text analysis introduced in our prior work (Rahman et al., 2020). Users can also utilize the Operations Menu to execute built-in text analysis and visualization operations. Figure 2a shows an example workflow in the Code Editor consisting of data cleaning, featurization, and visualization operations. Users can also perform these operations from Operations Menu without writing any scripts (see Figure 2b, and 2c).

Data View. Data View (see Figure 1C) shows a tabular representation of the underlying data. The

underlying data structure in LEAM is a dataframe. Data View is kept in sync with the dataframe—any changes made to the dataframe is immediately reflected in Data View (**D3**). For example, in Figure 3 when a user cleans the review column in the dataframe, the corresponding cleaned data is displayed in the Data View. In traditional script-based systems like computation notebooks, users are required to explicitly specify a print operation to view and inspect data.



(a) Noisy text (b) Cleaned text

Figure 3: As (a) user performs various cleaning operations on the "review" column as shown in Figure 2, (b) the cleaned column data is immediately displayed in Data View (**D3**).

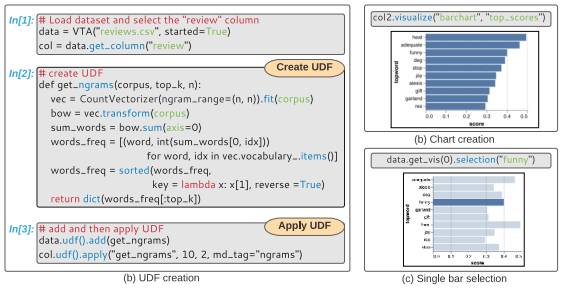
Chart View. LEAM enables users to generate visualizations either from the Code Editor or Operations Menu and displays those visualizations in the Chart View (see Figure 1B). Unlike computation notebooks, where analyzing visualizations in distant cells can be cumbersome, the side-by-side presentation of charts in Chart View enables users to compare and analyze related visualization without scrolling. We create the visualizations by extending Vega-Lite (Satyanarayan et al., 2016). These visualizations can be generated from Operations Menu or using VITAL commands and can be dynamically updated to add new interactions (discussed in Section 5).

5 Visual Text Analysis Using LEAM

The text analysis operations in LEAM are developed based on a visual text algebra, VTA (Rahman et al., 2020). LEAM provides a Python API called visual interactive text analysis library, VITAL, that enables users to write VTA commands in Code Editor. We now briefly introduce VTA and then demonstrate the corresponding specification library VITAL that we have developed.

5.1 VTA Operators and VITAL

VTA supports various operators for selecting a subset of the data (`selection`), transforming selected data into various representations for analysis (`transformation`), coordinating different views within the interface (`coordination`), and creating new operators by combining existing ones (`composition`). The JSON-style specification format of VTA is quite different from scripting languages widely used by analysts, such as R and Python. Composing operations in VTA can be cumbersome as users are required to specify multiple nested objects. Therefore, we have developed VITAL for declaratively specifying VTA commands in Code Editor of LEAM (**D2**). The VITAL commands are compiled and executed by the backend Python runtime of LEAM. We show several examples of VITAL commands that implement the VTA operators as well as newly introduced features next.



(b) UDF creation (b) Chart creation (c) Single bar selection

Figure 4: Declarative specification (**D2**): (a) using VITAL user creates and applies a UDF to compute top-K n-grams of reviews. Transparency (**D5**): (b) user generates a barchart of top words from Operation Menu which is logged as a VITAL script in Code Editor. Coordination (**D3**): (c) a user selected bar is highlighted on-demand.

5.2 Towards Integrated Text Analysis

We now explain how users can perform text analysis in LEAM.

5.2.1 User-guided Analysis

In Figure 2, we show how a user can analyze a text reviews dataset using various VITAL commands or menu operations like `project` (data cleaning) and `mutate` (featurization). Moreover, users can also combine multiple existing operators to declaratively specify user-defined operators (**D2**). For example, as shown in Figure 4, a user creates a new function to generate top n-grams in a given text corpus and then uses VITAL to load and then apply the UDF. Users can use the `visualize` command to create visualizations of the underlying data (see Figure 4b) and interactions (Figure 4c).

5.2.2 Programmatic Coordination

A key feature of LEAM is the ability to dynamically add coordination to existing visualizations

using VITAL (**D3**). Existing libraries like Vega-Lite only allow users to predefine the visualization and corresponding interactions without supporting any dynamic coordination specification.

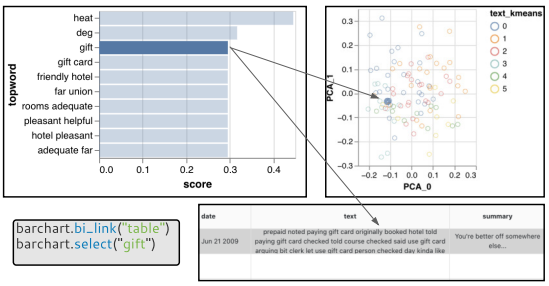


Figure 5: To relate a word in the chart with reviews both in Data View and the scatterplot (**D3**), the user issues a VITAL command in Code Editor (see inset). Clicking a bar in the barchart filters reviews in Data View and highlights relevant reviews in the scatterplot.

As shown in Figure 4c, users can update the selection type of the barchart in Figure 4b to enable single bar selection. Moreover, using VITAL, users can also dynamically specify external coordinations (a) among charts in the Chart View and (b) between Data View and charts. Vega-Lite does not provide a formal interaction grammar for such external coordination. For example, Figure 5 shows how users can enable coordination between the barchart, scatterplots, and data. Such dynamicity allows users to augment the visualizations instead of recreating charts and connect different views on demand to investigate data relationships. LEAM maintains a coordination graph to keep track of the linked views, which we discuss in Section 4.

5.2.3 Reusability and Transparency

Both VITAL and Operation Menu enable users to issue both analysis and coordination operations across different projects and workflows. Moreover, users can add their UDFs as new operators to VITAL and menu operations using the `add_UDF` command (see Figure 4a), thus ensuring reusability (**D4**). Users can also upload pre-trained models (*e.g.*, classification, regression) from Operations Menu and then access and reuse the models using the `get_model` and `predict` commands. To ensure transparency of the user interactions (**D5**) on the Operation Menu, LEAM logs the corresponding VITAL command in a new cell in Code Editor (see Figure 4b). The logging feature enables users to track their interactions, debug the logs if required, and re-execute those interactions.

6 LEAM Architecture

LEAM is developed as a web application and is implemented using ReactJS and Flask framework. We depict the architecture in Figure 6. LEAM client is responsible for capturing user input, and for rendering the views based on results returned by the back-end. Given any user interaction on the front end, the LEAM Request Processor issues a request to the backend LEAM Controller. This controller manages the uploaded data and sessions while propagating user interactions to the *session manager*.

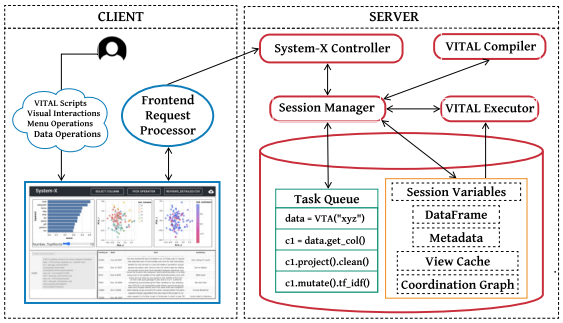


Figure 6: LEAM architecture. The front-end is a web application. The back-end features various components such as task queue, coordination graph, VITAL compiler, and executor to handle and execute user requests.

The session manager interprets the user interaction—any interactions on the Operations Menu is sent to a lightweight VTA Compiler while the VITAL commands on the Code Editor are pushed in a task queue. The VTA compiler translates the user-selected operator to a VITAL command which is then executed by the VTA Executor. LEAM backend employs a Task Queue to keep track of the VITAL commands in Code Editor. LEAM session manager also employs a View Cache to track the states of the front end views. LEAM employs a Coordination Graph to manage coordination among linked views—for any interaction on a view, all the views in its adjacency list are updated. For example, selecting a bar in the barchart in Figure 5 updates the scatterplot and Data View in its adjacency list.

7 Case Studies

To assess the impact of LEAM in performing visual text analysis and collect early feedback, we evaluated it through two case studies.

7.1 Study Design and Tasks

Design. The study consisted of three phases: (a) an introductory phase to help participants familiarize themselves with LEAM, (b) a workflow execution phase where the participants used LEAM to implement a text analysis workflow, and (c) a semi-structured interview to collect qualitative feedback regarding LEAM.

Participants. We recruited two participants within our professional network. Participant P_a was a researcher in natural language processing with extensive experience in review analysis and designing personal assistants and conversational bots. Participant P_b was a software engineer with experience in NLP pipelines and text analysis.

Tasks. We selected a spam detection workflow (Kaggle, 2021b) and a tweet analysis workflow (Kaggle, 2021a) from Kaggle, that are related to analyzing user-generated text as the respective tasks of our use cases. We chose the workflows based on their popularity and relevance to everyday text data analytics workflows in practice. For both the workflows, participants were provided pre-trained models. They were asked first to explore and preprocess a separate test dataset and then classify the data using the respective pre-trained model. For the preprocessing tasks, participants had to create a UDF. Participants were free to use any feature of LEAM or write code in Code Editor.

7.2 Observations

Both participants were able to complete their tasks with varying degrees of help from the experimenters. Participants appreciated the ability to perform the analysis both using Operations Menu (interactive) and Code Editor (declarative). They also found the user interface of LEAM more structured, commenting on the "messiness" of analysis using computational notebooks, also highlighted in prior work (Alspaugh et al., 2018). Moreover, participants found having visualizations within their eyesight without the need for scrolling up and down useful, a benefit of integrating multiple views (Rahman et al., 2021). They appreciated the ability to specify interactive coordination between visualizations and Data View using VITAL. P_a appreciated the ability to reuse operations from Operations Menu for bootstrapping the analysis.

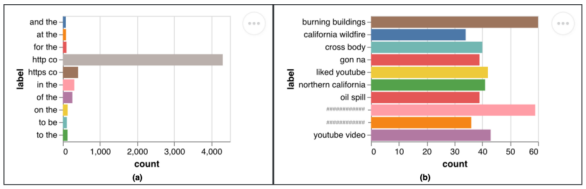


Figure 7: Bigram visualizations on Tweets dataset (Kaggle, 2021a) cleaned with a UDF. Users can immediately see the impact of the cleaning operation: (a) before and (b) after applying the UDF.

Participants also appreciated the ability to visualize the impact of their operations. Figure 7a displays a bi-gram visualization of the unprocessed tweets. After applying the cleaning operator on the tweets, the visualization was automatically updated (see Figure 7b). Such dynamic coordination highlights the importance of supporting context switching between stages in the data science pipeline, such as cleaning and visualization.

Participants also provided feedback for improvement. The most frequently raised issue was the need for improved communication of errors and the support for debugging, a requirement identified in earlier work (Chattopadhyay et al., 2020). Moreover, participants were occasionally confused about the effects of their operations, suggesting the need for visual guidance and better cues. Recent work explores such error detection methods for computational notebooks (Macke et al., 2021). Participants also pointed out a few syntactic inconsistencies of VITAL commands and suggested a more consistent design for ease of learning.

8 Conclusion and Future Work

This paper presents LEAM, a tool that enables users to perform interactive text analysis in-situ. Our declarative specification API VITAL provides support for a suite of operators to author diverse VITA workflows on-demand and enable different modes of interactive coordination among views. Preliminary evaluation of LEAM highlights the benefits of integrating multiple views, supporting both interactive and declarative specification of tasks, enabling reusability of operations, and ensuring transparency of interactions. While the initial results are promising, there is room for improvement in adding more transparency and providing wider operations coverage. LEAM can further benefit from addressing challenges related to scalability, workflow optimization, and version control that related work also explores.

References

Sara Alspaugh, Nava Zokaei, Andrea Liu, Cindy Jin, and Marti A Hearst. 2018. Futzing and moseying: Interviews with professional data analysts on exploration practices. *IEEE transactions on visualization and computer graphics*, 25(1):22–31.

Carenini et al. 2006. Interactive multimedia summaries of evaluative text. In *IUI*, pages 124–131.

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*, pages 1–12.

Codait. 2021. Text extensions for pandas.

Ian Drosos, Titus Barik, Philip J Guo, Robert DeLine, and Sumit Gulwani. 2020. Wrex: A unified programming-by-example interaction for synthesizing readable code for data scientists. In *ACM Human Factors in Computing Systems (CHI)*, pages 1–12.

Ashwin Ittoo, Antal van den Bosch, et al. 2016. Text analytics in industry: Challenges, desiderata and trends. *Computers in Industry*.

Project Jupyter. 2020. Project jupyter.

Kaggle. 2021a. Basic eda, cleaning and glove.

Kaggle. 2021b. Simple eda with data cleaning & glove.

Sean Kandel, Andreas Paepcke, Joseph Hellerstein, and Jeffrey Heer. 2011. Wrangler: Interactive visual specification of data transformation scripts. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 3363–3372.

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*, pages 1–11.

Mary Beth Kery, Donghao Ren, Kanit Wongsuphasawat, Fred Hohman, and Kayur Patel. 2020. The future of notebook programming is fluid. In *Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems*, pages 1–8.

Kucher et al. 2018. The state of the art in sentiment visualization. In *Computer Graphics Forum*, volume 37, pages 71–96. Wiley Online Library.

Doris Lee. 2020. Lux: A python api for intelligent visual discovery.

Liu et al. 2018. Bridging text visualization and mining: A task-driven survey. *IEEE TVCG*, 25(7):2482–2504.

Stephen Macke, Hongpu Gong, Doris Lee, Andrew Head, Doris Xin, and Aditya Parameswaran. 2021. Fine-grained lineage for safer notebook interactions. *Proceedings of the VLDB Endowment*, 14(6):1093–1101.

Sajjadur Rahman, Mangesh Bendre, Yuyang Liu, Shichu Zhu, Zhaoyuan Su, Karrie Karahalios, and Aditya Parameswaran. 2021. Noah: Interactive spreadsheet exploration with dynamic hierarchical overviews. *Proceedings of the VLDB Endowment*, 14(6):970–983.

Sajjadur Rahman, Peter Griggs, and Çağatay Demiralp. 2020. Leam: An interactive system for in-situ visual text analysis. In *Conference on Innovative Data Systems Research*.

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*, pages 1–12.

Satyanarayan et al. 2016. Vega-lite: A grammar of interactive graphics. *IEEE TVCG*, 23(1):341–350.

Smith et al. 2020. The machine learning bazaar: Harnessing the ml ecosystem for effective system development. In *ACM SIGMOD*, pages 785–800.

Wang et al. 2000. Guidelines for using multiple views in information visualization. In *Proceedings of the working conference on Advanced visual interfaces*, pages 110–119. ACM.

Hadley Wickham. 2016. *ggplot2: elegant graphics for data analysis*. springer.

Yifan Wu, Joe Hellerstein, and Arvind Satyanarayan. 2020. B2: Bridging code and interactive visualization in computational notebooks. In *ACM UIST*.

Zhang et al. 2020a. Teddy: A system for interactive review analysis. In *SIGCHI*, pages 1–13.

Ge Zhang, Mike A Merrill, Yang Liu, Jeffrey Heer, and Tim Althoff. 2020b. Coral: Code representation learning with weakly-supervised transformers for analyzing data analysis. *arXiv preprint arXiv:2008.12828*.