`Weedle`: Composable Dashboard for Data-Centric NLP in Computational Notebooks

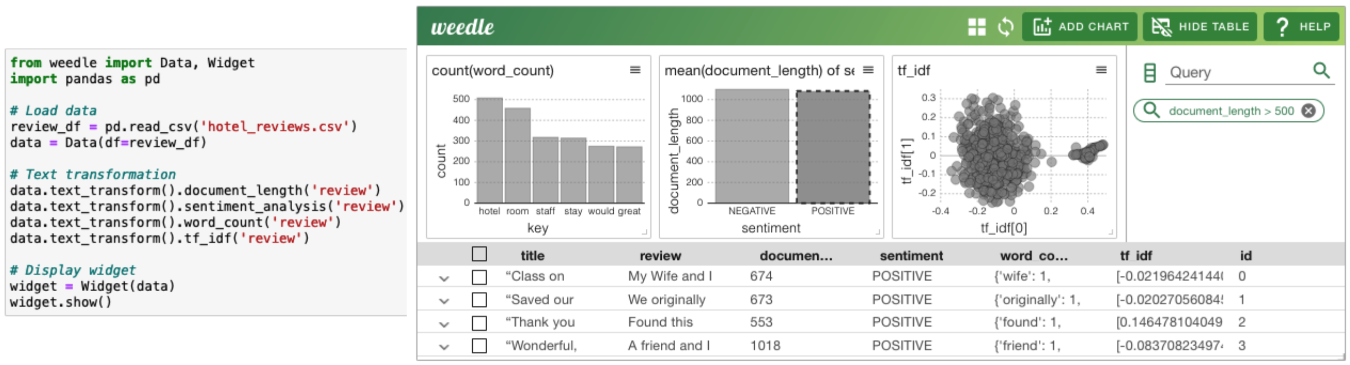
Nahyun Kwon
Texas A&M University
College Station, Texas, USA
nahyunkwon@tamu.edu

Hannah Kim
Megagon Labs
Mountain View, California, USA
hannah@megagon.ai

Sajjadur Rahman
Megagon Labs
Mountain View, California, USA
sajjadur@megagon.ai

Dan Zhang
Megagon Labs
Mountain View, California, USA
dan_z@megagon.ai

Estevam Hruschka
Megagon Labs
Mountain View, California, USA
estevam@megagon.ai


```
from weedle import Data, Widget
import pandas as pd

# Load data
review_df = pd.read_csv('hotel_reviews.csv')
data = Data(df=review_df)

# Text transformation
data.text_transform().document_length('review')
data.text_transform().sentiment_analysis('review')
data.text_transform().word_count('review')
data.text_transform().tf_idf('review')

# Display widget
widget = Widget(data)
widget.show()
```

		title	review	documen...	sentiment	word_co...	tf_idf	id
⌄	☐	"Class on	My Wife and I	674	POSITIVE	{'wife': 1,	[-0.02196424144(	0
⌄	☐	"Saved our	We originally	673	POSITIVE	{'originally': 1,	[-0.02027056084!	1
⌄	☐	"Thank you	Found this	553	POSITIVE	{'found': 1,	[0.146478104049	2
⌄	☐	"Wonderful,	A friend and I	1018	POSITIVE	{'friend': 1,	[-0.08370823497‹	3

Figure 1: (a) Syntax for data/widget instance initialization, transformations, and loading dashboard. (b) A dashboard widget with interconnected charts, which can be composed programmatically or interactively. Users can filter or group data items by brushing or selecting visual objects. Filter view displays currently applied filters and table view shows raw data.

ABSTRACT

Data-centric NLP is a highly iterative process requiring careful exploration of text data throughout entire model development lifecycle. Unfortunately, existing data exploration tools are not suitable to support data-centric NLP because of workflow discontinuity and lack of support for unstructured text. In response, we propose Weedle, a seamless and customizable exploratory text analysis system for data-centric NLP. Weedle is equipped with built-in text transformation operations and a suite of visual analysis features. With its widget, users can compose customizable dashboards interactively and programmatically in computational notebooks.

CCS CONCEPTS

• **Human-centered computing** → **Interactive systems and tools**; **Visual analytics**; • **Computing methodologies** → *Natural language processing*.

KEYWORDS

exploratory data analysis, computational notebooks

ACM Reference Format:
Nahyun Kwon, Hannah Kim, Sajjadur Rahman, Dan Zhang, and Estevam Hruschka. 2023. `Weedle`: Composable Dashboard for Data-Centric NLP in Computational Notebooks . In *Companion Proceedings of the ACM Web Conference 2023 (WWW '23 Companion), April 30-May 4, 2023, Austin, TX, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3543873.3587330

1 INTRODUCTION

With the widespread success of common model architectures such as BERT [4] in various applications, the attention of the machine learning community is shifting towards enhancing the quality of the data used in training and evaluation, i.e., data-centric AI. In contrast to traditional model-centric approaches to build a better model given benchmark datasets, the goal of data-centric AI is to iteratively improve the quality of the underlying data given a fixed model [8]. To achieve this, careful examination and diagnosis of data *throughout the whole machine learning (ML) lifecycle* is essential. During data preparation, researchers need to check for the amount of data (collection), label consistency (annotation), noisy samples or missing entries (wrangling), coverage and bias issues

Visualization	Transformation
Table view	simple overview/statistics of data, class distribution
Bar chart / histogram	class distribution, word count, ngram count, data item count per matching condition, document length, punctuation analysis, feature importance, embedding visualization
Line chart	document length, numerical trend over time
Scatter plot	t-SNE distribution, bivariate correlation, data item distribution, data item distribution with clustering, numerical trend over time
KDE plot	word count, document length
Pie chart	class distribution
Treemap	word count

Table 1: Used transformation functions per visualization type from top 30 most voted Kaggle notebooks with NLP tag.

(assessment), etc. During modeling, error analysis on underperforming data subsets is followed by additional iterations of collecting or wrangling more data and labels. After deployment, they need to monitor outputs to avoid performance degradation due to concept drift or data drift [8].

To understand, characterize, and diagnose the data during various stages of ML lifecycle, data exploration techniques can be employed. Exploratory data analysis (EDA)[1] includes a variety of approaches to explore data such as descriptive statistics, visualizations, or inspecting data samples [13]. While EDA generally occurs in the beginning of ML workflow to guide further steps, e.g., selection of techniques and tools [2], it is an iterative process that is highly intertwined with other steps [15, 17]. For example, analysts examined the intermediate results to discover heuristic rules to clean data during data wrangling or to identify underperforming subset of data during model development [10].

Unfortunately, existing EDA tools face several limitations in providing support for exploratory *text* analysis for data-centric NLP. First, these tools are separate from the rest of NLP pipelines, which results in workflow discontinuity due to frequent tool switching between iterations. Recent interview studies [2, 10, 15] report that analysts use a mix of spreadsheets, programming languages, visualization tools, database tools, and domain- or task-specific tools. For instance, to visually explore data, a user first exports the data to a tabular format and imports it in Tableau [2]. Switching between tools, users may experience increased logistic and cognitive overhead [5] due to migrating data or writing glue codes. Thus, there is a growing need for exploration tools that can be *seamlessly integrated with existing model development workflow*. Another issue in EDA for data-centric NLP is weak support for unstructured data such as text. Since most EDA methods and tools are designed for structured data (e.g., tables), analysts often have to transform their data into new forms (e.g., text $\longrightarrow$ word frequency, embeddings, topic modeling) or examine a sampled subset [15]. However, these extra transformation steps and subsequent EDA techniques can be highly arbitrary depending on analysis goals and downstream applications. To this end, a dedicated data model, operations, and interfaces with *built-in text support* can facilitate text analysis.

There have been several efforts to mitigate some of these challenges. To eliminate tedious tool switching, systems such as B2 [16] and DataPrep.EDA [9] or visualization libraries such as Altair [14]

[1]Following Wongsuphasawat et al. [15], we adopt a broad definition of EDA that aims at both profiling (understanding and assessing data) and discovery (new insights).
[2]https://www.tableau.com/

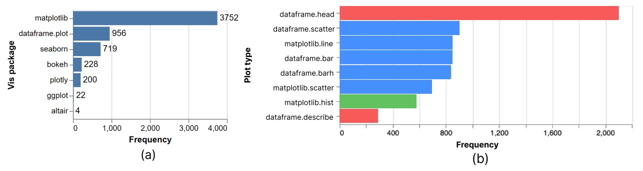


Figure 2: (a) Frequency of popular visualization packages for python from 5K NLP-related notebooks. (b) Frequency of functions from the top two packages from (a). Practitioners seek for the overview of data in a tabular format (red bars), bivariate analysis using bar, line, and scatter plots (blue bars), and univariate analysis using histogram (green bar).

and Matplotlib [6] enable transitions between code and visualizations in computational notebooks. Plotly Dash [3], Panel [4], and Symphony [3] allow users to compose dashboards in python environments. Leam [5] offers built-in text transformation operations based on a grammar for text analysis. However, these works either lack built-in text support, require environment switching, have task-specific designs without customizability and composablity, or do not offer rich interactive visualizations that enable local or subset-level analysis.

To address these issues, we present **`Weedle`**: **W**idget-**E**nabled **E**xploratory **D**ata analysis for N**L**P **E**xperts. It is a seamless, customizable EDA system with built-in text support and composable dashboard. **`Weedle`** is implemented as a python package containing a Jupyter widget, which can be seamlessly integrated with existing ML development environment. **`Weedle`**'s data model and interactive dashboard is tightly integrated with each other, grounded on our analysis of public NLP-related notebooks from GitHub and Kaggle.

2 NOTEBOOK ANALYSIS

To understand current data exploration practices for NLP, we analyzed public computational notebooks from GitHub and Kaggle. We compiled a list of common text transformation functions by exploring popular Kaggle notebooks that are tagged as NLP category. We manually examined the top 30 notebooks that received the most votes as of July 2022. We found that NLP practitioners used popular transformation methods (e.g., jaccard score, tf-idf, document clustering, distribution by labels) to convert text data to a structured format for further analyses. As Table 1 shows, commonly used transformation operations included general overview by printing data in a tabular format, frequent word distribution, ngram count, document length, distribution by labels, and embedding distribution. These transformed features are then explored using different forms of visualizations such as bar, line, scatter, violin plot, etc. We also noticed that there were lots of repetitive code snippets over multiple notebook cells, e.g., conducting the same analysis for different slices of data or creating similar charts with slightly different parameters, as the used visualization packages were mostly static.

We conducted an additional step of analysis on the public notebook dataset retrieved from github [12] to see if the various visualizations in Table 1 are actively used in real world settings by practitioners. Out of the 1M notebooks in total, we sampled the

[3]https://plotly.com/
[4]https://panel.holoviz.org/

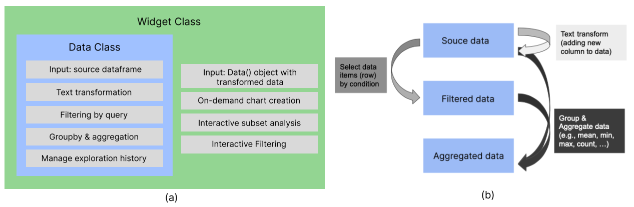

Widget Class
Data Class
Input: source dataframe
Text transformation
Filtering by query
Groupby & aggregation
Manage exploration history
Input: Data() object with transformed data
On-demand chart creation
Interactive subset analysis
Interactive Filtering
(a)
Souce data
Text transform (adding new column to data)
Select data items (row) by condition
Filtered data
Group & Aggregate data (e.g., mean, min, max, count, ...)
Aggregated data
(b)

Figure 3: (a) The high-level architecture of Weedle. Widget object contains an underlying Data object. (b) Data object keeps three stages of data—source, filtered, and aggregated—based on user inputs from the dashboard.

most recent 80k notebooks and retrieved 5k notebooks that have used NLP-specific packages such as nltk, gensim, and pyLDAvis. We then searched for the import statements from the notebooks to retrieve the frequently used visualization packages. As Figure 2a describes, the most popular package used for visualization in addressing NLP topics was matplotlib, followed by a built-in plot methods in pandas dataframe. Focusing on these two libraries, we investigated which chart types are frequently used. Figure 2b shows that users actively used the basic charts - bar, line, and scatter plots along with plain table view rather than various types of visualization methods including KDE, violin, and box plot. Thus, we choose bar charts and scatter plots as our basic visualization methods since it can support uni- and bi-variate analysis.

3 WEEDLE: EDA FOR DATA-CENTRIC NLP

Weedle is developed as a pip-installable python package. It contains two main classes, data and widget (Figure 3a). Their designs are influenced by our analysis on public notebooks in Section 2. Data class manages all data-related operations, including loading the source data, structuring data through text transformation functions, query-based filtering, and aggregations on groups. Having the instance of data class as an input, Widget class takes care of interactive features, such as on-demand chart creation, chart editing, and interactive filtering via brushing and selection, all of which triggers corresponding data operations.

The frontend is implemented using ipywidget[5] and Svelte[6]. The underlying data operations utilize various NLP packages including nltk, spacy, gensim, and transformers, along with data management/analysis libraries such as numpy and pandas.

3.1 Data Model for Text Support

Our data model keeps three stages of data: source data, filtered data, and aggregated data (Figure 3b) as DataFrames. Having one source data, text transformations update the source data itself by appending new columns based on the source column provided by users. For filtered data and aggregated data, **Weedle** keeps the history of explorations for provenance management.

Our system currently supports 6 data types: text, categorical, numerical, dict with categorical keys and numerical values (e.g., keywords and their frequency), vector (e.g., tf-idf, sentence embeddings), and SpanArray [11] (e.g., array of token and its POS tag pairs). The data type of each column can be given by the user or inferred automatically by our heuristic rules.

[5]https://ipywidgets.readthedocs.io/en/stable/
[6]https://svelte.dev/

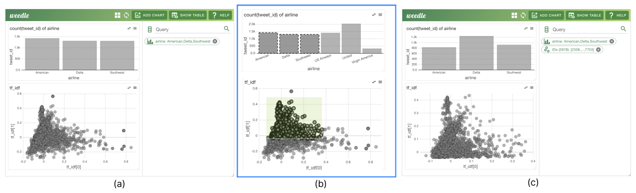


Figure 4: Interactive filtering for subset analysis. All charts are interconnected by shared filters. (b) Users can select visual components, i.e., bars and dots, via clicking (top) or brushing (bottom) to apply filters. (a) and (c) shows the widget status before and after the filtering interaction, correspondingly.

Our system supports the built-in text transformations that can be triggered only with the simple parameters as shown in Figure 1a. This includes document length, word count, word embedding, named entity recognition, tf-idf, topic modeling, sentiment analysis, and part-of-speech (POS) tagging. Users can pass on optional parameters, e.g., custom stopwords list for creating bag of words. Our system also supports a count operation that captures the number of occurrences per each data item given a set of keywords or regular expression patterns. Each transformation operation appends the output data as a new column (see Figure 3b). In addition, our system allows users to plug-in any custom transformation operation as a python function. For example, given a title text and a content text, generate an embedding of the combined text using a fine-tuned language model.

Grouping and aggregating operations should differ by the types of input data columns. For grouping a numerical feature, binning is performed by default. After grouping, aggregation operations calculate a representative value for each group. Built-in aggregation methods include count, average, min/max, etc for numerical and dict typed features and count, number of unique values for categorical features. Also, users can plug in custom aggregation functions.

3.2 Composable Dashboard with Multiple Coordinated Charts

Weedle contains a composable dashboard widget that facilitates EDA within computational notebooks. Users can build the dashboard on-demand by providing configurations programmatically (as parameters to Widget class) or interactively in a widget (see Figure 1b). The dashboard consists of the chart view, the filter view, the table view, and the menu bar. All components are linked via selection and filtering interactions. Users can resize the widget and its view components to allocate more space if needed.

The chart view shows multiple coordinated visualizations of underlying data. Users can flexibly change the size or layout of individual charts in the chart view via dragging. To add a visualization in the chart view, users can click 'Add chart' button in the menu bar. This will open a popup modal to set chart configurations such as chart type, which feature columns to use, grouping and aggregation options, etc. Currently **Weedle** supports two types of charts, bar and scatter. The system itself has the control on the inputs for attributes that only provides the right combinations of attributes and the available aggregations based on the cases regarding data types. Using a menu button on the top right of each chart, users can modify its configuration, remove the chart, or export the visualization as an image file. Users can interact with visual objects in the chart (e.g., bars and dots) to understand what it represents (e.g., a data item or a group of data items) and create a filter with selected data subset. As in Figure 4, users can either individually click visual objects one by one while holding a shift key, or use brushing to create a bounding box to select visual objects. After selecting visual objects, clicking right mouse button will automatically add a filter with the corresponding condition. The filters are shared in all views, so a filtering interaction updates the entire charts as Figure 4c illustrates.

The filter view contains the query input box and shows a list of currently applied filters. Filters can be based on feature values, data item ids, or query text, which are denoted by icons in the filter list. A filter can be removed by clicking the 'x' icon in the filter list. Using the query input box on the top of the filter view, users can add a query-based filter. The syntax of query filters simply follows the if statement format in Python. For example, if users would like to see the data where the document length (column name) is higher than 500, the query would be 'document length > 500'. In addition, filters can be added from the chart view as described above or from notebook cells programmatically. When a filter is added or removed, underlying data operations are triggered and then the chart view and the table view are updated correspondingly.

The table view displays raw data, allowing users to inspect individual items. Users can sort the table by columns. Each row in the table view is expandable to show long entries. Users can hide the table view using the button in the menu bar.

All widget configurations (e.g., chart types, grouping and aggregation attributes, and filters) are updated in a real-time manner through user interactions in frontend as well as in notebook cells. This enables not only the effective communication between the frontend and backend, but also allows the system to keep track of analysis history, so that the users can save or load the dashboard locally by importing/exporting the configurations. Also, individual charts can be saved as a PNG image for further use.

4 USE CASE: SENTIMENT ANALYSIS

We present a use case to demonstrate the workflow of **Weedle** in EDA for data-centric NLP.[7] We suppose that a speculative user who is working on the project that requires sentiment classification model for airline reviews. She is using Twitter US Airline review dataset [1], and she first loads the dataframe on the data instance. **Weedle** automatically recognizes data types for columns, and she can also manually assign the types. To examine data characteristics, she performs the built-in functions to generate columns for document length, tf-idf, bag-of-words, topic modeling, etc. She proceeds to the dashboard, and creates several charts, including: bar charts for (1) distribution of sentiment labels, (2) average document length for each sentiment label, (3) topic distribution for each sentiment label, (4) top keyword distribution, and (5) scatter plot for the 2D projection of tf-idf scores. She explores several variables by selecting visual objects and adds filters to update the charts. For example, she first wants to see if there is any difference on the average document lengths of positive and negative labeled data. She finds out that negative reviews tend to be longer than positive ones. She wonders if there is any correlation between tf-idf score range and sees that the labels are evenly distributed over entire tf-idf range. She generates the predicted sentiment labels with a pre-trained model, then creates one more bar chart for predicted label for error rates and analysis. Finally she finds out that the accuracy on the 'positive' reviews is relatively low, and she suspects that the first viable issue might be the data imbalance.

[7]Demo video is available at https://youtu.be/vZSTkWKqVqU.

5 LIMITATION AND FUTURE WORK

Computational overhead due to frequent filtering and aggregation operations may hinder users' interaction experience. Potential solutions can be caching, efficient indexing using databases, incremental computations [7]. Our current evaluation is limited to a specific use case. To evaluate the applicability of our tool for various NLP applications, we plan to recruit NLP experts and conduct an experimental study with their own datasets. Other lines of future work include extending data model, mixed-initiative data diagnosis, etc. Also, we are currently adding more text transformation operations and augmenting interactions between components in the widget to facilitate deeper exploration. We plan to make **Weedle** public.

REFERENCES

[1] 2019. Twitter US Airline Sentiment. https://www.kaggle.com/datasets/crowdflower/twitter-airline-sentiment.

[2] Sara Alspaugh, Nava Zokaei, Andrea Liu, Cindy Jin, and Marti A. Hearst. 2019. Futzing and Moseying: Interviews with Professional Data Analysts on Exploration Practices. *IEEE TVCG* 25, 1 (2019), 22–31.

[3] Alex Bäuerle, Ángel Alexander Cabrera, Fred Hohman, Megan Maher, David Koski, Xavier Suau, Titus Barik, and Dominik Moritz. 2022. Symphony: Composing Interactive Interfaces for Machine Learning. In *Proc. CHI 2022.* Article 210, 14 pages.

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proc. NAACL 2019.* 4171–4186.

[5] Peter Griggs, Cagatay Demiralp, and Sajjadur Rahman. 2021. Towards integrated, interactive, and extensible text data analytics with Leam. In *Proc. DaSH 2021.* 52–58.

[6] John D. Hunter. 2007. Matplotlib: A 2D Graphics Environment. *Computing in Science & Engineering* 9, 3 (2007), 90–95.

[7] Stratos Idreos, Olga Papaemmanouil, and Surajit Chaudhuri. 2015. Overview of Data Exploration Techniques. In *Proc. SIGMOD 2015.* 277–281.

[8] Andrew Ng. 2021. MLOps: from model-centric to data-centric AI. https://www.deeplearning.ai/wp-content/uploads/2021/06/MLOps-From-Model-centric-to-Data-centricAI.pdf.

[9] Jinglin Peng, Weiyuan Wu, Brandon Lockhart, Song Bian, Jing Nathan Yan, Linghao Xu, Zhixuan Chi, Jeffrey M. Rzeszotarski, and Jiannan Wang. 2021. DataPrep.EDA: Task-Centric Exploratory Data Analysis for Statistical Modeling in Python. In *Proc. SIGMOD 2021.* 2271–2280.

[10] Sajjadur Rahman and Eser Kandogan. 2022. Characterizing Practices, Limitations, and Opportunities Related to Text Information Extraction Workflows: A Human-in-the-Loop Perspective. In *Proc. CHI 2022.* Article 628, 15 pages.

[11] Frederick Reiss, Hong Xu, Bryan Cutler, Karthik Muthuraman, and Zachary Eichenberger. 2020. Identifying Incorrect Labels in the CoNLL-2003 Corpus. In *Proc. CoNLL 2020.* 215–226.

[12] Adam Rule, Aurélien Tabard, and James D. Hollan. 2018. Exploration and Explanation in Computational Notebooks. In *Proc. CHI 2018.* 1–12.

[13] John W Tukey. 1977. *Exploratory Data Analysis.* Vol. 2. Reading, MA.

[14] Jacob VanderPlas, Brian Granger, Jeffrey Heer, Dominik Moritz, Kanit Wongsuphasawat, Arvind Satyanarayan, Eitan Lees, Ilia Timofeev, Ben Welsh, and Scott Sievert. 2018. Altair: Interactive Statistical Visualizations for Python. *Journal of Open Source Software* 3, 32 (2018), 1057.

[15] Kanit Wongsuphasawat, Yang Liu, and Jeffrey Heer. 2019. Goals, Process, and Challenges of Exploratory Data Analysis: An Interview Study. *arXiv* 1911.00568 (2019).

[16] Yifan Wu, Joseph M Hellerstein, and Arvind Satyanarayan. 2020. B2: Bridging Code and Interactive Visualization in Computational Notebooks. In *Proc. UIST 2020.* 152–165.

[17] Ge Zhang, Mike A Merrill, Yang Liu, Jeffrey Heer, and Tim Althoff. 2022. CORAL: COde RepresentAtion learning with weakly-supervised transformers for analyzing data analysis. *EPJ Data Science* 11, 1 (2022), 14.