Data Formulator: AI-powered Concept-driven Visualization Authoring

Chenglong Wang, John Thompson, and Bongshin Lee

Abstract—With most modern visualization tools, authors need to transform their data into tidy formats to create visualizations they want. Because this requires experience with programming or separate data processing tools, data transformation remains a barrier in visualization authoring. To address this challenge, we present a new visualization paradigm, *concept binding*, that separates high-level visualization intents and low-level data transformation steps, leveraging an AI agent. We realize this paradigm in Data Formulator, an interactive visualization authoring tool. With Data Formulator, authors first define *data concepts* they plan to visualize using natural languages or examples, and then bind them to visual channels. Data Formulator then dispatches its AI-agent to automatically transform the input data to surface these concepts and generate desired visualizations. When presenting the results (transformed table and output visualizations) from the AI agent, Data Formulator provides feedback to help authors inspect and understand them. A user study with 10 participants shows that participants could learn and use Data Formulator to create visualizations that involve challenging data transformations, and presents interesting future research directions.

Index Terms—AI, visualization authoring, data transformation, programming by example, natural language, large language model

1 Introduction

Most modern visualization authoring tools (e.g., Charticulator [39], Data Illustrator [27], Lyra [42]) and libraries (e.g., ggplot2 [53], Vega-Lite [44]) expect tidy data [54], where every variable to be visualized is a column and each observation is a row. When the input data is in the tidy format, authors simply need to bind data columns to visual channels (e.g., Date $\mapsto$ x-axis, Temperature $\mapsto$ y-axis, City $\mapsto$ color in Fig. 1). Otherwise, they need to prepare the data, even if the original data is clean and contains all information needed [3]. Authors usually rely on data transformation libraries (e.g., tidyverse [55], pandas [33]) or separate interactive tools (e.g., Wrangler [17]) to transform data into the appropriate format. However, authors need either programming experience or tool expertise to transform data, and they have to withstand the overhead of switching between visualization and data transformation steps. The challenge of data transformation remains a barrier in visualization authoring.

To address the data transformation challenge, we explore a fundamentally different approach for visualization authoring, leveraging an AI agent. We separate the high-level visualization intent "*what to visualize*" from the low-level data transformation steps of "*how to format data to visualize*," and automate the latter to reduce the data transformation burden. Specifically, we support two key types of data transformations (and their combinations) needed for visualization authoring:

- **Reshaping**: A variable to be visualized is spread across multiple columns or one column includes multiple variables. For example, if authors want to create a different scatter plot from the table in Fig. 1 by mapping Seattle and Atlanta temperatures to x,y-axes (Fig. 2-①), they need to first "pivot" the table from long to wide format, because both variables of interest are stored in the Temperature column and are not readily available.

- **Derivation**: A variable needs to be extracted or derived from one or more existing columns. For example, if authors want to create a bar chart to show daily temperature differences between two cities (Fig. 2-②) and a histogram to count the number of days which city is warmer (Fig. 2-③), they need to derive the temperature difference and the name of the warmer city from the two cities' temperature columns, and map them to the y-axis and x-axis, respectively, and the city name to color channels of the corresponding charts. The derivation is also needed when the variable to be visualized requires analytical computation (e.g., aggregation, moving average, percentile) across multiple rows from a column in the table. For example, to plot a line chart to visualize the 7-day moving averages of Seattle temperatures (Fig. 2-④), the authors need to calculate the moving average using a window function and map it to y-axis with Date on x-axis.

Date	City	Temperature
2020-01-01	Seattle	51
2020-01-01	Atlanta	45
2020-01-02	Seattle	45
2020-01-02	Atlanta	47
...	...	...
2020-12-31	Seattle	48
2020-12-31	Atlanta	49

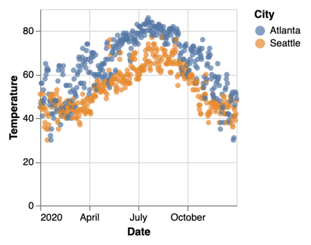


Fig. 1: A dataset of Seattle and Atlanta daily temperatures in 2020 (left) and a scatter plot that visualizes them by mapping Date to x-axis, Temperature to y-axis, and City to color (right).

In this paper, we introduce Data Formulator, an interactive visualization authoring tool that embodies a new paradigm, *concept binding*. To create a visualization with Data Formulator, authors provide their visualization intent by binding data concepts to visual channels. Upon loading of a data table, existing data columns are provided as known data concepts. When the required data concepts are not available to author a given chart, the authors can create the concepts: either using natural language prompts (for derivation) or by providing examples (for reshaping). Data Formulator handles these two cases differently, with different styles of input and feedback, and we provide a detailed description of how they are handled in Sec. 2. Once the necessary data concepts are available, the authors can select a chart type (e.g., scatter plot, histogram) and map data concepts to desired visual channels. If needed, Data Formulator dispatches the backend AI agent to infer necessary data transformations to instantiate these new concepts based on the input data and creates candidate visualizations. Because the authors' high-level specifications can be ambiguous and Data Formulator may generate multiple candidates, Data Formulator provides feedback to explain and compare the results. With this feedback, the authors can inspect, disambiguate, and refine the suggested visualizations. After that, they can reuse or create additional data concepts to continue their visualization authoring process.

We also report a chart reproduction study conducted with 10 participants to gather feedback on the new concept binding approach that

- *Chenglong Wang, John Thompson, and Bongshin Lee are with Microsoft Research. E-mail: {chenglong.wang, johnthompson, bongshin}@microsoft.com.*

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

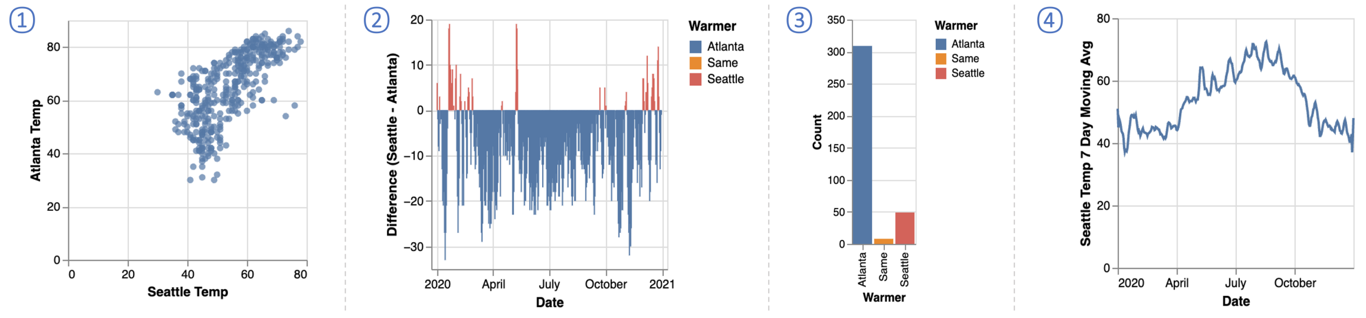


Fig. 2: Visualizations created from df in Fig. 1 that require data transformation: (1) a scatter plot with Seattle and Atlanta temperatures on x,y-axes, (2) a bar chart to visualize the temperature difference between the two cities, (3) a histogram to count the number of days each city being warmer, and (4) a smoothed line chart that shows the 7-day moving averages of Seattle temperature.

employs an AI agent, and to evaluate the usability of Data Formulator. After an hour-long tutorial and practice session, most participants could create desired charts by creating data concepts—both with derivation and reshaping transformations. We conclude with a discussion on the lessons learned from the design and evaluation of Data Formulator, as well as important future research directions.

2 Illustrative Scenarios

In this section, we illustrate users' experiences to create visualizations in Figs. 1 and 2 using programs and Data Formulator from the initial input data in Fig. 1. We refer to this dataset as df in this section.

2.1 Experience with Programming

We first illustrate how an experienced data scientist, Eunice, uses programming to create the desired visualizations with pandas and Altair libraries in Python.

Daily Temperature Trends. Eunice starts with the scatter plot in Fig. 1. Because df is in the tidy format with Date, City, and Temperature available, Eunice needs no data transformation and writes a simple Altair program to create the plot:

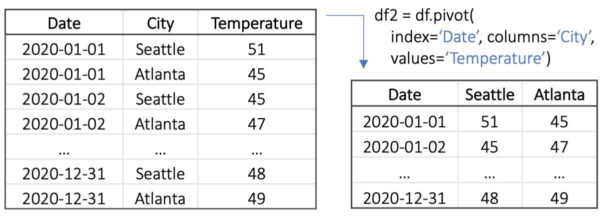
```
alt.Chart(df).mark_circle().encode(x='Date', y='Temperature', color='City')
```

This program calls the Altair library (alt), selects the input dataset df and the scatter plot function mark_circle, and maps columns to x,y and color channels. It renders the desired scatter plot in Fig. 1.

Seattle vs. Atlanta Temperatures. To make a more direct comparison of two cities' temperatures, Eunice wants to create a different scatter plot (Fig. 2-①) by mapping Seattle and Atlanta temperatures to x,y-axes. However, Seattle and Atlanta temperatures are not available as columns in df. She therefore needs to transform df to surface them. Because df is in the "long" format, where temperatures of both cities are stored in one column Temperature, she needs to pivot the table to the "wide" format. Eunice switches to the data transformation step and uses the pivot function from the pandas library to reshape df (Fig. 3). This program populates Seattle and Atlanta as new column names from the City column, and their corresponding Temperature values are moved to these new columns by Date. With df2, Eunice creates the desired visualization, which maps Seattle and Atlanta to x,y-axes of the scatter plot with the following program:

```
alt.Chart(df2).mark_circle().encode(x='Seattle', y='Atlanta')
```

Temperature Differences. Eunice wants to create two visualizations to show how much warmer is Atlanta compared to Seattle: a bar chart to visualize daily temperate differences (Fig. 2-②) and a histogram to show the number of days each city is warmer (Fig. 2-③). Again, because necessary fields Difference and Warmer are not in df2, Eunice needs to transform the data. This time, she writes a program to perform column-wise computation, which extends df2 with two new columns

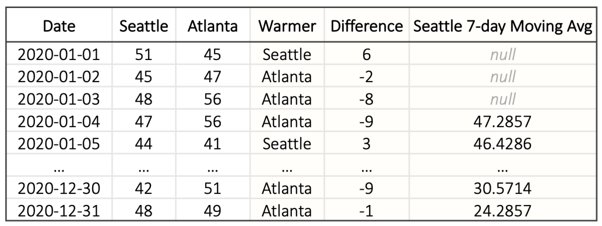


Date	City	Temperature
2020-01-01	Seattle	51
2020-01-01	Atlanta	45
2020-01-02	Seattle	45
2020-01-02	Atlanta	47
...	...	...
2020-12-31	Seattle	48
2020-12-31	Atlanta	49

```
df2 = df.pivot(
   index='Date', columns='City',
   values='Temperature')
```

Date	Seattle	Atlanta
2020-01-01	51	45
2020-01-02	45	47
...	...	...
2020-12-31	48	49

Fig. 3: Prepare the new data df2 with the pivot function to populate Seattle and Atlanta temperatures from City and Temperature columns.



Date	Seattle	Atlanta	Warmer	Difference	Seattle 7-day Moving Avg
2020-01-01	51	45	Seattle	6	null
2020-01-02	45	47	Atlanta	-2	null
2020-01-03	48	56	Atlanta	-8	null
2020-01-04	47	56	Atlanta	-9	47.2857
2020-01-05	44	41	Seattle	3	46.4286
...	...	...	...	...	...
2020-12-30	42	51	Atlanta	-9	30.5714
2020-12-31	48	49	Atlanta	-1	24.2857

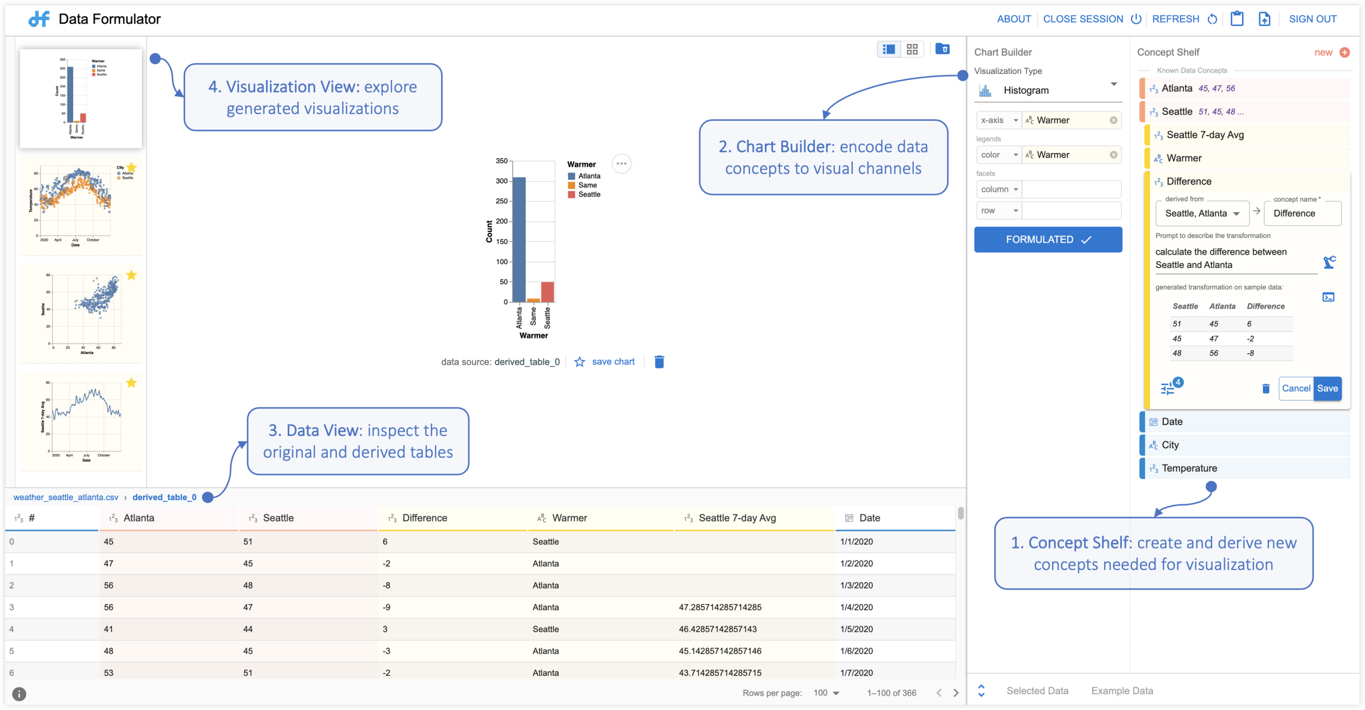
Fig. 4: Extend df2 in Fig. 3 to derive Warmer, Difference, and Seattle 7-day Moving Avg columns that are necessary for visualizations in Fig. 2.

Warmer and Difference (Fig. 4). Eunice then creates the daily temperature differences chart by mapping Date and Difference to x,y-axes and the histogram by mapping Warmer to x-axis and the aggregation function, count(), to y-axis to calculate the number of entries.

```
# extend df2 with new columns 'Difference' and 'Warmer'
df2['Difference'] = df2['Seattle'] - df2['Atlanta']
df2['Warmer'] = df2['Difference'].apply(
   lambda x: 'Seattle' if x > 0 else ('Atlanta' if x < 0 else 'Same'))

# create the bar chart
alt.Chart(df2).mark_bar().encode(x='Date', y='Difference',
↪   color='Warmer')
# create the histogram
alt.Chart(df2).mark_bar().encode(x='Warmer', y='count()',
↪   color='Warmer')
```

7-day Moving Average of Seattle's Temperature. Finally, Eunice wants to include a line chart for Seattle temperature trends in the report. Because daily temperatures fluctuate, she decides to create a smooth line chart based on 7-day moving average temperatures. Eunice needs an analytical function to calculate the moving average. Because the input data is sorted by Date, Eunice chooses the rolling function from pandas: she sets window=7 and center=True so that the moving average is calculated with a sliding window from day $d-3$ to day $d+3$ for each date d. This transformation adds the new column Seattle 7-day Moving Avg to df2 (Fig. 4; the first 3 days are null because of insufficient data),



calculate the difference between Seattle and Atlanta

Seattle	Atlanta	Difference
51	45	6
45	47	-2
48	56	-8

weather_seattle_atlanta.csv › derived_table_0

#	Atlanta	Seattle	Difference	Warmer	Seattle 7-day Avg	Date
0	45	51	6	Seattle		1/1/2020
1	47	45	-2	Atlanta		1/2/2020
2	56	48	-8	Atlanta		1/3/2020
3	56	47	-9	Atlanta	47.285714285714285	1/4/2020
4	41	44	3	Seattle	46.42857142857143	1/5/2020
5	48	45	-3	Atlanta	45.142857142857146	1/6/2020
6	53	51	-2	Atlanta	43.714285714285715	1/7/2020

Fig. 5: Data Formulator UI. After loading the input data, the authors interact with Data Formulator in four steps: (1) in the Concept Shelf, create (e.g., Seattle and Atlanta) or derive (e.g., Difference, Warmer) new data concepts they plan to visualize, (2) encode data concepts to visual channels of a chart using Chart Builder and formulate the chart, (3) inspect the derived data automatically generated by Data Formulator, and (4) examine and save generated visualizations. Throughout the process, Data Formulator provides feedback to help authors understand generated data and visualizations.

and Eunice maps Date and the new column to a line chart to create the desired visualization (Fig. 2-④).

```
df2['Seattle 7-day Moving Avg'] = df2['Seattle'].rolling(window=7, center=True)
alt.Chart(df2).mark_line().encode(x='Date', y='Seattle 7-day Moving Avg')
```

Remark. In all cases, Eunice can specify visualizations using simple Altair programs by mapping data columns to visual channels. However, data transformation steps make the visualization process challenging. Eunice needs to choose the right type of transformation based on the input data and desired visualization (e.g., creating the scatter plot in Fig. 1 from df2 would require unpivot instead). Furthermore, Eunice needs knowledge about pandas to choose the right function and parameters per task (e.g., rolling will not fit if Eunice wants to calculate moving average for each city in df). Eunice's programming experience and data analysis expertise allowed her to successfully complete all tasks. But a less experienced data scientist, Megan, finds this process challenging. Megan decides to use Data Formulator to reduce the data transformation overhead.

2.2 Experience with Data Formulator

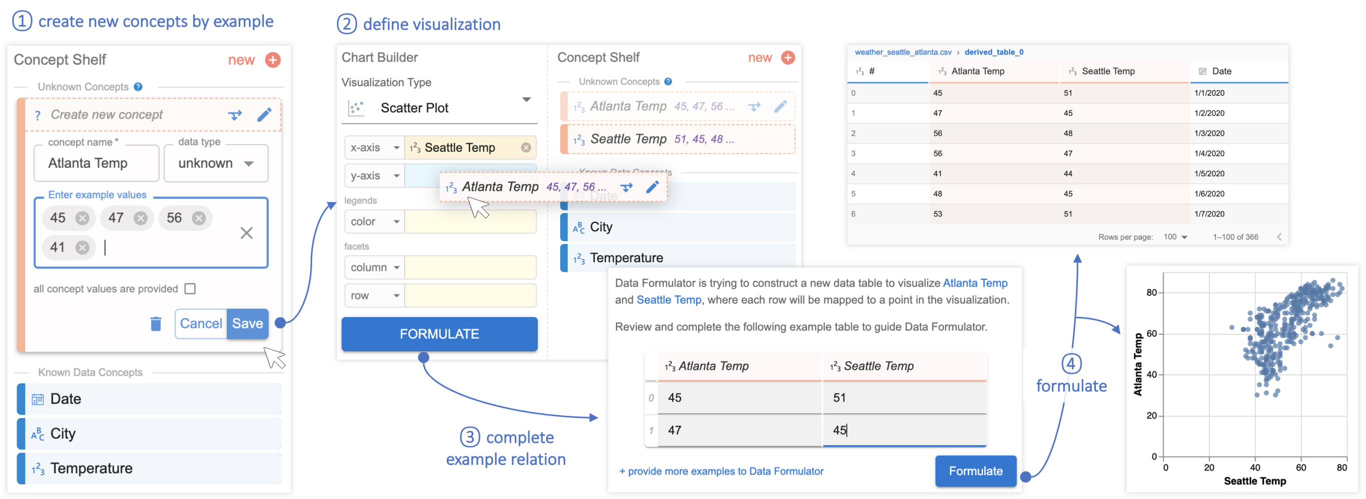
Data Formulator (Fig. 5) has a similar interface as "shelf-configuration"-style visualization tools like Tableau or Power BI. But unlike these tools that support only mappings from input data columns to visual channels, Data Formulator enables authors to create and derive new data concepts and map them to visual channels to create visualizations *without requiring manual data transformation*.

Daily Temperature Trends. Once Megan loads the input data (Fig. 1), Data Formulator populates existing data columns (Date, City, and Temperature) as known *data concepts* in the Concept Shelf. Because all three data concepts are already available, no data transformation is needed. Megan selects the visualization type "Scatter Plot" and maps these data concepts to x, y and color channels in Chart Builder through drag-and-drop interaction. Data Formulator then generates the desired scatter plot.

Seattle vs. Atlanta Temperatures. To create the second scatter plot (Fig. 4-①), Megan needs to map Seattle and Atlanta temperatures to x, y-axes of a scatter plot. Because Seattle and Atlanta temperatures are not available as concepts yet, Megan starts out by creating a new data concept Atlanta Temp (Fig. 6-①): she clicks the new ⊕ button in the Concept Shelf, which opens a concept card that asks her to name the new concept and provide some examples values; Megan provides four Atlanta temperatures (45, 47, 56, 41) from the input data as examples and saves it. Similarly, Megan creates another new concept Seattle Temp. Because Data Formulator's current knowledge to them is limited to their names and example values, both concepts are listed as an unknown concept for now. (They will be resolved later when more information is provided.)

With these new concepts and the Scatter Plot selected, Megan maps new data concepts Seattle Temp and Atlanta Temp to x, y-axes (Fig. 6-②), and then clicks the FORMULATE button to let Data Formulator formulate the data and instantiate the chart. Based on the visualization spec, Data Formulator realizes that the two unknown concepts are related to each other but not yet certain how they relate to the input data. Thus, Data Formulator prompts Megan with an example table to complete: each row in the example table will be a data point in the desired scatter plot. Megan needs to provide at least two data points from the input data to guide Data Formulator on how to generate this transform (Fig. 6-③). Here, Megan provides the temperatures of Atlanta and Seattle on 01/01/2020 and 01/02/2020 from the table Fig. 1. When Megan submits the example, Data Formulator infers a program that can transform the input data to generate a new table with fields Atlanta Temp and Seattle Temp that subsumes the example table provided by Megan. Data Formulator generates the new table and renders the desired scatter plot (Fig. 6-④). Megan inspects the derived table and visualization and accepts them as correct.

Temperature Differences. To create a bar chart and a histogram to visualize temperature differences between the two cities, Megan needs two new concepts, Difference and Warmer. This time, Megan notices that both concepts can be *derived* from existing fields based



Data Formulator is trying to construct a new data table to visualize Atlanta Temp and Seattle Temp, where each row will be mapped to a point in the visualization.

Review and complete the following example table to guide Data Formulator.

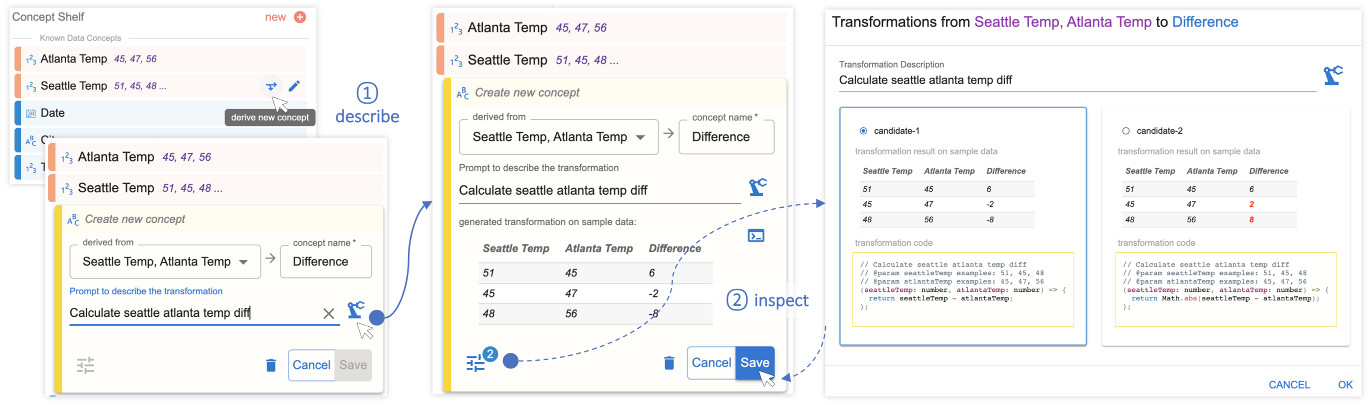
	Atlanta Temp	Seattle Temp
0	45	51
1	47	45

weather_seattle_atlanta.csv › derived_table_0

#	Atlanta Temp	Seattle Temp	Date
0	45	51	1/1/2020
1	47	45	1/2/2020
2	56	48	1/3/2020
3	56	47	1/4/2020
4	41	44	1/5/2020
5	48	45	1/6/2020
6	53	51	1/7/2020

Rows per page: 100 1–100 of 366

Fig. 6: Megan (1) creates new data concepts, Seattle Temp and Atlanta Temp, by providing examples and (2) maps them to x,y-axes of a scatter plot to specify the visualization intent. (3) Data Formulator asks Megan to provide a small example to illustrate how these two concepts are related, and Megan confirms the example. (4) Based on the example, Data Formulator generates the data transformation and creates the desired visualization.



Seattle Temp	Atlanta Temp	Difference
51	45	6
45	47	-2
48	56	-8

Transformations from Seattle Temp, Atlanta Temp to Difference

Transformation Description
Calculate seattle atlanta temp diff

candidate-1

transformation result on sample data

Seattle Temp	Atlanta Temp	Difference
51	45	6
45	47	-2
48	56	-8

transformation code

```
// Calculate seattle atlanta temp diff
// @param seattleTemp examples: 51, 45, 48
// @param atlantaTemp examples: 45, 47, 56
(seattleTemp: number, atlantaTemp: number) => {
  return seattleTemp - atlantaTemp;
};
```

candidate-2

transformation result on sample data

Seattle Temp	Atlanta Temp	Difference
51	45	6
45	47	2
48	56	8

transformation code

```
// Calculate seattle atlanta temp diff
// @param seattleTemp examples: 51, 45, 48
// @param atlantaTemp examples: 45, 47, 56
(seattleTemp: number, atlantaTemp: number) => {
  return Math.abs(seattleTemp - atlantaTemp);
};
```

Fig. 7: (1) Megan derives the new concept Difference from Atlanta Temp and Seattle Temp using natural language. Data Formulator generates two candidates and displays the first one in the concept card. (2) Megan opens the dialog to inspect both, confirms the first one, and saves the concept.

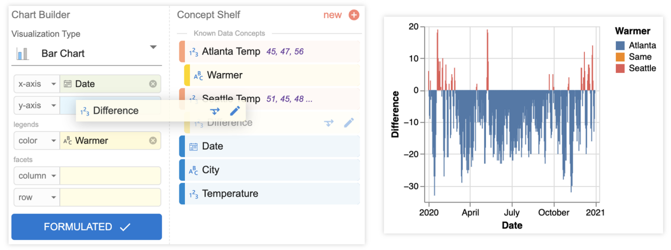
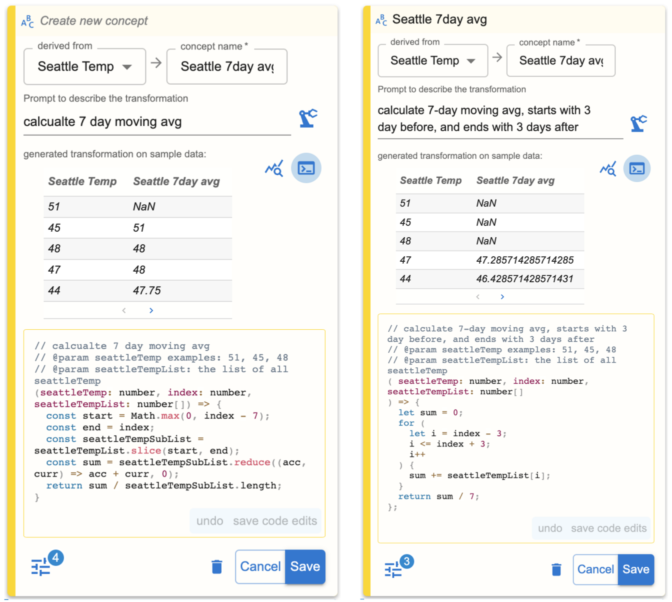


Fig. 8: Megan creates the bar chart using derived concepts, Difference and Warmer, as well as an original concept Date.

on column-wise mappings, and thus she uses the “derive” function of Data Formulator (Fig. 7). Megan first clicks the “derive new concept” option on the existing concept Seattle Temp, which opens up a concept card that lets her describe the transformation she wants using natural language. Megan selects Seattle Temp and Atlanta Temp as the “derived from” concepts, provides a name Difference for the new concept, and describes the transform using natural language, “Calculate seattle atlanta temp diff.” Megan then clicks the generate button and Data Formulator dispatches its backend AI agent to generate code. Data Formulator returns two code candidates and presents the first one in the concept card. Megan opens up the dialog to inspect both candidates and learns that because her description did not clearly specify whether she wants the difference or its absolute value, Data Formulator returns both options as candidates. After inspecting the example table and the transformation code provided by Data Formulator, Megan confirms the first candidate and saves the concept Difference. Similarly, Megan creates a concept, Warmer, from Seattle Temp and Atlanta Temp with the description “check which city is warmer, Atlanta, Seattle, or same.” Data Formulator applies the data transformation on top of the derived table from the last task and displays the extended table in Data View (Fig. 5). Because both concepts are now ready to use, Megan maps them to Chart Builder to create the desired visualizations (Fig. 8).

7-day Moving Average of Seattle’s Temperature. Last, Megan needs to create a line chart with 7-day moving average temperatures. Because the moving average can be derived from the Seattle Temp column, Megan again chooses to use the derive function. Megan starts with a brief description “calculate 7-day moving avg” and calls Data Formulator to generate the desired transformation. Upon inspection, Megan notices that the generated transformation is close but does not quite match her intent: the 7-day moving average starts from $d-6$ to d for each day d as opposed to $d-3$ to $d+3$ (Fig. 9). Based on this observation, Megan changes the description into “calculate 7-day moving avg, starts with 3 days before, and ends with 3 days after” and re-runs Data Formulator. This time, Data Formulator generates the correct transformation and presents the extended data table in Fig. 5. Megan then maps Date and Seattle 7-day Moving Avg to x,y-axes of a line chart.



calcualte 7 day moving avg

Seattle Temp	Seattle 7day avg
51	NaN
45	51
48	48
47	48
44	47.75

```
// calcualte 7 day moving avg
// @param seattleTemp examples: 51, 45, 48
// @param seattleTempList: the list of all
seattleTemp
(seattleTemp: number, index: number,
seattleTempList: number[]) => {
  const start = Math.max(0, index - 7);
  const end = index;
  const seattleTempSubList =
seattleTempList.slice(start, end);
  const sum = seattleTempSubList.reduce((acc,
curr) => acc + curr, 0);
  return sum / seattleTempSubList.length;
}
```

calculate 7-day moving avg, starts with 3 day before, and ends with 3 days after

Seattle Temp	Seattle 7day avg
51	NaN
45	NaN
48	NaN
47	47.285714285714285
44	46.428571428571431

```
// calculate 7-day moving avg, starts with 3
day before, and ends with 3 days after
// @param seattleTemp examples: 51, 45, 48
// @param seattleTempList: the list of all
seattleTemp
( seattleTemp: number, index: number,
seattleTempList: number[]
) => {
  let sum = 0;
  for (
    let i = index - 3;
    i <= index + 3;
    i++
  ) {
    sum += seattleTempList[i];
  }
  return sum / 7;
};
```

Fig. 9: Megan derives the 7-day moving averages from Seattle Temp. After inspecting the results, she edits the description to be more precise.

Remark. With the help of Data Formulator, Megan creates visualizations without manually transforming data. Instead, Megan specifies the data concepts she wants to visualize by:

- building new concepts using examples (when the new concept is spread among multiple columns or multiple concepts are stored in the same column, e.g., Seattle Temp and Atlanta Temp are both stored in the Temperature column); and
- deriving new concepts using natural language (when the new concept can be computed from existing ones using column-wise operators, e.g., Difference from Seattle Temp and Atlanta Temp).

Megan then drags-and-drops data concepts to visual channels of a chart. In this process, for derived concepts, Data Formulator displays generated candidate code and example table to help Megan inspect and select the transformation; for concepts created by example, Data Formulator prompts Megan to elaborate their relations by completing an example table. Data Formulator then transforms the data and generates the desired visualizations. Data Formulator reduces Megan's visualization overhead by shifting the task of specifying data transformation into the task of inspecting generated data. Because Data Formulator's interaction model centers around data concepts, Megan does not need to directly work with table-level operators, such as pivot, map/reduce and partitioning, which are challenging to master.

3 The Data Formulator Design

In this section, we describe our design principles, explain Data Formulator's interaction model, and how Data Formulator derives data concepts and formulates visualizations from the author's inputs.

3.1 Design Principles

Data Formulator introduces *data concepts*, an abstraction of the columns needed for an author to specify their target visualization. To eliminate the author's burden to manually transform the data table before plotting, we designed Data Formulator based on the following guiding design principles.

Treat design concepts as first-class objects. The notion of data concepts is a generalization of table columns: it is a reference to columns both from a current table and from a future transformed table. They offer two benefits. First, concept-level transformations are easier to describe and understand than table-level operators. Table-level transformations require either advanced operators like pivot and unpivot, or high-order functions like map and window, while concept-level operators are first-order functions over primitive elements (e.g., arithmetic) or lists (e.g., percentile). This makes it easier for the author to communicate with the AI agent and verify the results. Second, we can build the interaction experience on top of existing designs people are already familiar with: data concepts resemble data columns existing shelf-configuration tools commonly use.

Leverage benefits from multiple interaction approaches. Data Formulator employs both natural language interaction (for deriving concepts) and programming-by-example approach (for building custom concepts). Natural language descriptions have a superior ability to translate high-level intent into executable code and large language models (LLMs) can reason about natural concepts (e.g., academic grades are A, B, C, D, and F; months are from January to December). However, it can be difficult for the author to provide proper descriptions if they do not understand notions like pivoting, and natural language descriptions can be imprecise and ambiguous. In contrast, while program synthesizers cannot reason about natural concepts, they are less ambiguous, and it is easier for the author to convey reshaping operations by demonstrating the output relation. By incorporating multiple approaches and feedback for different transformation types (derivation vs. reshaping), Data Formulator takes advantage of both, reducing the specification barrier and improving the likelihood for the AI agent to generate correct and interpretable codes.

Ensure correct data transformation and promote trust. While LLM and program synthesizers can automatically generate code to eliminate the author's manual data transformation burden, they can incorrectly generalize the author's specification. Therefore, it is crucial for the author to view and verify the results. Our design employs mechanisms to ensure such inspection by the author: (1) display multiple candidates for the author to review, if available, (2) display both the code (process) and the sample output values (results) to help the author understand the transformation, and (3) allow the author to edit the generated transformation code to correct or refine it.

Improve the system expressiveness. Data Formulator's expressiveness is defined by the combination of transformation function and visualization language. Data Formulator's visualization spec builds on top of Vega-Lite specifications. While Data Formulator's UI does not provide options to layer marks, the author can import their custom Vega-Lite specs of layered visualizations to achieve the same design. For data transformation, Data Formulator supports reshaping options from tidyverse as described in Sec. 2, and it supports both column-wise derivation and analytical computation that can be generated by the LLM. Note that while our transformation language does not include aggregation, the author can achieve the same visualization by setting aggregation options on the desired axes (e.g., map Month to x-axis and avg(Seattle Temp) to y-axis to create a bar chart with average temperature). However, with the current design, the author cannot derive or reshape data that first require aggregation without re-importing the aggregated data.

3.2 Interaction Model

Figure 10 shows Data Formulator's high-level interaction model. Data Formulator first loads data columns from the input table as original (and known) concepts (e.g., Date, City, and Temperature concepts in Fig. 5). The author uses the Concept Shelf to create new data concepts, if needed, in two ways (Sec. 3.3): (1) derive a concept from existing ones by interacting with an AI agent using natural language or (2) build a custom concept by providing example values. If the new concept is derived from known concepts, Data Formulator immediately extends the current data table and registers it as a known concept.

With necessary data concepts known, the author uses the Chart Builder to map data concepts to visual channels of a chart. If unknown custom concepts are used to specify a visualization, Data Formulator asks the author to provide an example relation among the encoded concepts to transform the input table by using a programming-by-example approach. With the necessary data formulations applied, Data Formulator generates a Vega-Lite spec and renders the visualization.

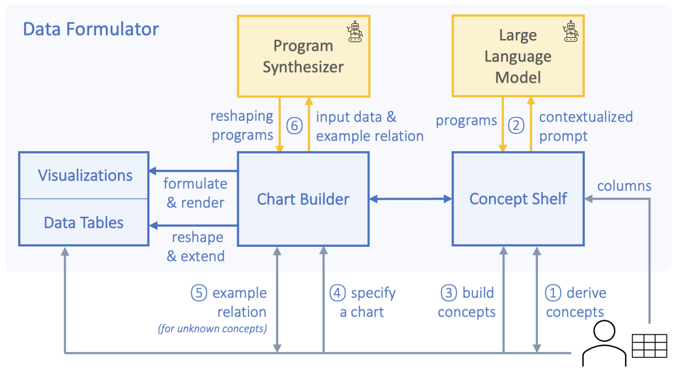


Fig. 10: Data Formulator's interaction model.

3.3 Creating New Data Concepts

The author can **derive** a concept from one or more data concepts by interacting with Data Formulator's AI agent (Fig. 10-①). In addition to a concept name, the author provides both a list of source concepts from which the new concept is derived and a natural language description of the transformation (Fig. 7-①). Data Formulator then generates a contextualized prompt that grounds the description in the context of source concepts. This prompt combines the author's description and the descriptions of input parameters for all source concepts (with example values sampled from their domains) as comments, and joins it with the function prefix to instruct the AI agent to complete a Typescript function (as opposed to generate non-code text or uncontrolled code snippets). Data Formulator prepares two types of prompts for each query to cover simple derivation (Example 1) and analytical computation (Example 2) because it does not know if analytical computation is needed beforehand.

Example 1: The prompt for "Calculate seattle atlanta temp diff" with source concepts Seattle Temp and Atlanta Temp (Fig. 7).

```
// Calculate seattle atlanta temp diff
// @param seattleTemp examples: 51, 45, 48
// @param atlantaTemp examples: 45, 47, 56
(seattleTemp: number, atlantaTemp: number) => {
```

Example 2: The prompt for "calculate 7-day moving avg" with source concept Seattle Temp (Fig. 9). It provides index and seattleTempList so that the function can access to other values of the seattleTemp when analytical computation is needed (e.g., calculate the moving average for current index, derive percentile of the seatteTemp among all values).

```
// calculate 7-day moving avg
// @param seattleTemp examples: 51, 45, 48
// @param seattleTempList: the list of all seattleTemp
(seattleTemp: number, index: number, seattleTempList: number[]) => {
```

Data Formulator sends both prompts to LLM (we use Codex Davinci 2 [7]) to generate the transformation code (Fig. 10-②), asking for five candidate completions. When candidate programs are returned from LLM, Data Formulator filters out programs that are not executable or contain error outputs by executing them on sample values from source domains. Data Formulator then presents the programs along with their example execution results for the author to inspect (Fig. 7). Once confirmed, a new derived concept is created and shown in the Concept Shelf. If all source fields are known concepts, Data Formulator derives a new column by applying the transformation function to every tuple from the source columns and appends the column in the current table for the author to review (e.g., Fig. 5).

The author can also **build** a custom concept by providing its name and a set of example values that belong to its domain (Fig. 10-③). Custom concepts are designed to support data reshaping: the author creates custom concepts when (1) the concept is spread across multiple columns; the author wants to combine multiple columns in a wide table to create one new concept in a long table, (2) multiple concepts are stored in one column; they want to surface fields from a long table, and (3) multiple values for a concept are collapsed in a column as a list (e.g., the value for an "actors" column is a list of actors for each movie); the author wants to split the list into multiple rows (i.e., one actor per row). These custom concepts are *not* known yet upon creation because Data Formulator needs additional information from the author to resolve their relation with the input data. As we will describe in the next section, the resolution is achieved by inferring the reshaping program based on the example relations provided by the user.

With data concepts (including newly crated ones) ready, the author is ready to interact with the Chart Builder to create visualizations.

3.4 Specifying and Formulating the Visualization

Chart Builder employs a shelf-configuration interface: authors drag-and-drop data concepts to visual channels of the selected visualization to specify visual encoding. Based on the encoding, Data Formulator generates a Vega-Lite specification (e.g., Fig. 11) to render the visualization. Data Formulator adopts a chart-based specification: each chart type corresponds to a Vega-Lite template with placeholder encodings to be filled from the author specification. Data Formulator currently supports scatter plots (circle-based, bubble chart, ranged dot plots), bar charts (single-column, stacked, layered, grouped, histogram), line charts (with and without dots), heatmap, and custom charts (with all compatible visual channels).

```
{ "mark": "circle", "encoding" : { "x": {"field": "Date", "type": "temporal"}, "y":
↪  {"field": "Temperature", "type": "quantitative"},  "color": {"field": "City"} } }

{ "mark": "circle",  "encoding" : { "x": {"field": "Seattle Temp", "type":
↪  "quantitative"}, "y": {"field": "Atlanta Temp", "type": "quantitative"} } }
```

Fig. 11: Vega-Lite specs for the scatter plots in Fig. 2-1 and Fig. 6.

When all fields used in the visual encoding are available, Data Formulator combines the Vega-Lite spec with the input data to render the visualization (e.g., Fig. 1). Otherwise, when some concepts are unknown (unresolved custom concepts or concepts derived from unknown ones), Data Formulator first interacts with the author and then calls the program synthesis engine to create the transformed table.

Once the author specifies the visual encoding, Data Formulator first checks if any unknown concepts are used. If so, it asks the author to illustrate the relation of unknown concepts with other concepts used in the visual encoding by filling out an example relation in a sample table (e.g., Fig. 10-⑤). Data Formulator needs such example relation to disambiguate the visualization intent because unknown concepts contain example values only from their own domains, missing information on how they will be related row-wise in the transformed table. For example, Data Formulator generates the example relation with Seattle Temp and Atlanta Temp fields as shown in Fig. 6-③ for the author to complete. To reduce the author's efforts, Data Formulator pre-fills two example values of Atlanta Temp based on its sample domain and asks the author to complete their corresponding Seattle Temp values (e.g., what's Seattle Temp when Atlanta Temp is 45). Each row in the example relation will be a row in the transformed data, which will then be mapped to a point in the scatter plot.

Once the author submits the example relation, Data Formulator calls the program synthesizer to solve the data reshaping problem (Fig. 10-⑥). Given an example relation E, with input data T, the program synthesizer solves the programming-by-example problem to find a reshaping program p such that $E \subseteq p(T)$ (i.e., the transformed data should generalize the example E). The reshaping program p is defined by the grammar in Figure 12, where p is recursively defined over four core reshaping operators from the R tidyverse library. We include only reshaping operators because other operators like unite and summarise are already supported by Data Formulator's ability to derive concepts from natural language. With this grammar, the program synthesizer performs an enumerative search in the program space for candidate programs. To speed up this combinatorial search process, we leverage abstract interpretation to prune the search space: the program synthesis

$$
\begin{array}{lll}
p \leftarrow T & & \\
\quad \mid \ \mathsf{pivot\_longer}(p, \bar{c}) & & \textit{(pivot from wide to long)} \\
\quad \mid \ \mathsf{pivot\_wider}(p, c_{name}, c_{vals}) & & \textit{(pivot from long to wide)} \\
\quad \mid \ \mathsf{separate}(p, c) & & \textit{(split a column into two)} \\
\quad \mid \ \mathsf{separate\_rows}(p, c) & & \textit{(separate a collapsed column into rows)}
\end{array}
$$

Fig. 12: Reshaping operators supported by Data Formulator. T refers to input data, and c refers to column names.

engine returns candidate programs that satisfy the example relation to Chart Builder. Note that multiple candidates could be generated since the example relation is small and potentially ambiguous. In practice, unlike other programming-by-example tools, the small example relation is precise enough to constrain the program space that only the correct candidate is returned, because the program synthesizer only needs to solve the reshaping problem.

With generated reshaping programs, Chart Builder prepares the input data: it first generates a reshaped table from each reshaping program and then for every derived concept used in the encoding, it extends the reshaped table with a new column by applying the transformation function on every tuple in the table. This way, Data Formulator generates a new table with all necessary fields to instantiate the visualization.

Data Formulator presents the prepared table and candidate visualizations for the author to inspect (Fig. 5-③④). When the author confirms and saves a desired visualization, the transformed data is used to resolve unknown concepts: these concepts are now available as known concepts to be used to create visualizations.

3.5 Implementation

Data Formulator is built as a React web application in Typescript; its backend is a Python server that runs on a Dv2-series CPU with 3.5 GiB RAM on Azure. Data Formulator's backend queries the OpenAI Codex API for concept derivation and runs the synthesis algorithm locally. Data Formulator's scalability to larger data relates to (1) the frontend's visualization rendering capability and (2) the backend's efficiency to execute data transformation scripts. To scale up Data Formulator for large datasets, we envision a sampling-based approach [29], where Data Formulator presents results on a representative subset of data to enhance interactivity and returns full results asynchronously.

4 Evaluation: Chart Reproduction Study

We conducted a chart reproduction study [40] to gather feedback on the new concept binding approach that employs an AI agent, and to evaluate the usability of Data Formulator.

4.1 Study Design

Participants. We recruited 10 participants (3 female, 7 male) from a large technology company. All participants had experience creating (simple) charts and identified themselves as a person with normal or corrected-to-normal vision, without color vision deficiency. Six participants are data scientists, two are applied scientists, and the remaining two are data & applied scientists, and they are all located in the United States. Four participants are in their 30's, three are in 20's, and one participant is in each of the 40's, 50's, and 18-19 age group. They had varying levels of self-identified expertise in terms of chart authoring, computer programming, and experience with LLMs.

Tasks and Datasets. We prepared six chart reproduction tasks with two datasets (3 tasks for each dataset): daily COVID-19 cases from Jan 21, 2020 to Feb 28, 2023 (3 columns; 1,134 rows) for the first task set (Tasks 1-3) and daily temperatures in 2020 for Seattle and Atlanta (4 columns; 732 rows; Fig. 1) for the second set (Tasks 4-6). In both task sets, each subsequent task is built upon the previous one. One task (Task 4) required building two new concepts for reshaping and the other five tasks required the use of derived concepts. We also prepared three tutorial tasks, using students' exam scores dataset (5 columns; 1,000 rows): in addition to the scores for three subjects (math, reading, and writing), the data table included a student's id and major. The first tutorial task was about creating a chart with known/available concepts, while the second and third tutorial tasks were for creating charts using derived concepts and unknown concepts, respectively. Finally, we produced two practice tasks (one for reshaping and another for derivation). For these, the exam scores dataset was transformed into a long format, including math and reading scores under the subject and score column, resulting in 4 columns and 2,000 rows.

Setup and Procedure. We conducted sessions remotely via the Microsoft Teams. Each session consisted of four segments: (1) a brief explanation of the study goals and procedure, (2) training with tutorial and practice, (3) chart reproduction tasks, and (4) debrief.

The training segment started with a quick introduction of Data Formulator's basic interactions using a simple task that does not require data transformation. Then, with their screen shared and recorded with audio, participants went through a tutorial and created three visualizations following step-by-step instructions provided in slides. They next created two visualizations on their own as practice. After an optional break, the participants performed six reproduction tasks using the two datasets mentioned above. Each task included a description (e.g., "Create a Scatter Plot to compare Atlanta Temperature against Seattle Temperature."), the labels for axes and color legend (if necessary), and an image of the target visualization. (Study materials are included in the supplemental material.) We encouraged the participants to think aloud, describing their strategies, whether any feature of Data Formulator works or makes sense, if the system behaves as they expect, etc. We recorded if the participants required a hint (and which hint) and how long it took for them to complete the task. The recorded completion time is not intended to indicate performance, as we wanted to gain insights about our approach using the think aloud method. Instead, we wanted to see if and how the participants completed, faltered, or recovered for each task, within a reasonable amount of time. The session ended with a debriefing after the participants filled out a questionnaire with 5 questions about their experience with Data Formulator. The entire session took about two hours to complete, while the training segment took about an hour. We compensated each participant with a $100 Amazon Gift card.

4.2 Results

After an hour-long tutorial and practice session, most participants could use Data Formulator to create different types of charts that involve advanced data transformations. Furthermore, they were generally positive about their experience with Data Formulator in chart authoring.

Tasks Completion and Usability Issues. Participants completed all tasks on average within 20 minutes, with a deviation of about four and a half minutes. Table 1 shows the average and standard deviation of task completion time in minutes, along with the total number of hints provided for each chart reproduction task (for all 10 participants). The participants spent most of their time (on average less than five minutes) on Task 6 because it was not trivial to inspect the code to generate 7-day moving average. For Tasks 5 and 6, we had to give one hint (to two different participants) to guide them to use a different type of concept (they needed to derive a concept but initially tried to build a concept). There were a few cases that we had to provide a hint to a single participant: how to select multiple sources for derivation (Task 4), what are the correct source concepts for derivation (Tasks 2 & 5), and the example values should be from the original table (Task 4). We had to provide the highest number of hints for Task 1. This was because when participants derived the year from the date value, its data type was set to number and the participants did not know or remember how to change its data type to string. (As detailed below, some participants tried to fix it by providing a different natural language prompt).

For derived concepts, once the participants identified the correct interaction approach and input fields, they are able to describe and refine the transformation in natural language to solve the tasks. We recorded all participants' prompts (see supplementary material). On average, participants made 1.62 prompt attempts per derived concept, and the length of those prompts averaged 7.28 words. The system generated an average of 1.94 candidates per prompt attempt.

Table 1: The average and standard deviation of task time (in minutes) and the total number of hints provided for chart reproduction tasks.

Task	Average Time	Standard Deviation	Total Number of Hints
Task 1	2:21	0:45	7
Task 2	3:19	2:09	2
Task 3	3:45	1:33	2
Task 4	2:43	1:33	2
Task 5	2:22	1:55	3
Task 6	4:29	1:39	2

Participants rated Data Formulator on five criteria using a 5-point Likert scale (5 being the most positive) as follows: easy to learn (M = 3.90, SD = 0.88), easier than other tools to transform data (M = 3.80, SD = 1.23), AI-agent's usefulness (M = 4.4, SD = 0.70), helpful to verify generated data (M = 4.1, SD = 0.74), and the trustworthiness of generated data (M = 4.7, SD = 0.48).

Participants provide feedback to improve the user interface. Four participants expected a way to multi-select on concept cards and click "derive" for deriving a concept from multiple existing ones. The current method of clicking "derive" on one concept and then multi-selecting is not intuitive. Two other participants expected the AI to select or identify which concepts to derive from based on their prompts. A few participants expected to change data type using the prompt (e.g., "year as a string" when the year is extracted from date). Five participants wanted the derived examples table to show more values, or unique derived values. Reshaping data was at times a point of confusion: two participants found it difficult to understand how the AI formulated candidate datasets, while two others did not intuit or remember the task sequence to formulate data for unknown concepts. When required to reshape data, three participants entered plausible, but not exact values in the example table during the training: they misunderstood the rigid connection to the original dataset. To strengthen that connection participants recommended including additional columns (especially a column that is unique for a pivot transform) or to filter or highlight rows of the data table view that correspond to the values used in the example table. We also observed users' attempts to re-use a derived concept as a commutative function on other concepts: two participants tried to drag a derived concept and drop it on other concepts.

Overall Reaction and Experience. To understand participants' reaction to the new concept-drive approach employing an AI agent, we analyzed the debrief interview, during which participants stated something or confirmed an observation made by the experimenter. Using the transcription from the recorded sessions, one researcher applied an open coding method to surface all unique feedback, issues and ideas from the participants. He expanded the codes to generalize for semantically similar participant statements. While quantities of qualitative data does not provide a metric for importance, we counted how many participants mentioned each code, providing how frequently our participants had shared experiences or ideas with Data Formulator.

Overall, participants were positive about their experience with Data Formulator. All 10 participants said that natural language prompts work well for generating data transforms and eight mentioned that AI is a helpful tool for the study tasks. Participants more frequently praised the derived concept than the unknown concept method for transforming data. Specifically, when it comes to verifying candidate derived concepts: all except one participant commented that displaying code was helpful and seven found the example derived values table to be useful. While only half of the participants commented that pivoting with unknown concepts is easier than with other tools, only three affirmed the example data table being helpful.

Five participants mentioned that they were impressed by the power of the AI agent to generate data transforms. Five participants found having candidates (for both derived and formulated data) to be helpful because the candidates provided an opportunity to choose a correct answer, or at the least to select a promising direction to refine. Participants also explained that generating candidates increases trust in a collaborative experience. On the other hand, three participants mentioned they are reluctant to give much trust to the AI generative features of the tool.

5 Related Work

Data Formulator builds on top of prior research in visualization authoring tools, data transformation tools, and code generation techniques.

Visualization Grammars and Tools. The grammar of graphics [56] first introduces the representation of visualizations based on chart types and encodings of data columns to their visual channels. Many high-level grammars are designed to realize this idea. For example, ggplot2 [53] is a charting library in R based on visual encodings. Vega-Lite [44] and its Python-wrapper Altair [49] extend the traditional grammar of graphics design with rules for layered and multi-view displays, as well as interactions, and Animated Vega-Lite [63] further extends it to support animations. These grammars hide low-level implementation details and are concise. Therefore, they are generally preferred for the rapid creation of visualization in exploratory settings over toolkits and libraries like Protovis [4], Atlas [26], and D3 [5] that are designed for more expressive and novel visualization authoring. High-level grammars inspire interactive visualization tools like Tableau [47], Power BI, Lyra [42], Charticulator [39], and Data Illustrator [27]. These tools adopt a shelf-configuration design: authors map data columns to visual encoding "shelves" often using the drag-and-drop interaction, and enerate specifications in high-level grammars to render visualizations. These grammars and tools require that the input data is in a tidy format, where all variables to be visualized are columns of input data. Because this means authors often need to transform the data first to create any visualizations, Satyanarayan et al. recognized the automatic inferring or suggestions of appropriate transformations when necessary, as an important research problem [43].

To reduce authors' efforts, visualization by demonstration [41,45,62] and by example [52] tools are introduced. Lyra 2 [62] generates interaction rules after authors perform an interaction on the visualization. VbD [41] lets users demonstrate transformations between different types of visualizations to produce new specifications. Although these approaches reduce the chart specification efforts, they require tidy input data. Falx [52], on the other hand, addresses the data transformation challenge with a visualization-by-example design. Falx lets authors specify visualizations via low-level example mappings from data points to primitive chart elements. However, Falx does not support derivation types of transformation because of its underlying programming-by-example algorithm limitations; its requirement to focus on low-level elements also introduces a challenging paradigm shift for users who are more familiar with tools that focus on high-level specifications [44,47].

Natural language interfaces [8, 11, 19, 25, 28, 35] enhance users' ability to author and reason about visualizations. NCNet [28] uses a Seq-to-Seq model to translate chart description texts into Vega-Lite specs. VisQA [19] is a pipeline that leverages semantic parsing techniques [34] to provide atomic data-related answers based on its visualizations. NL4DV [31] and Advisor [25] generate visualizations based on user questions. To manage ambiguity in natural language inputs [46], DataTone [11] ranks solutions based on user preference history, and Pumice [23] introduces a multi-modal approach that leverages examples to refine the initial ambiguous specification. Data Formulator's concept derivation interface is based on natural language. Data Formulator benefits from large language models' expressiveness [7], and manages ambiguity by restricting the target function type to columns-to-column mapping functions (as opposed to arbitrary data transformation scripts). In the future, more powerful language models can be plugged into Data Formulator to improve code generation quality.

Data Formulator adopts the shelf-configuration approach like Tableau and Power BI, but it supports encoding from *data concepts* to visual channels to address the data transformation burden. Because Data Formulator can automatically transform the input data based on the concepts used in the visualization specification, authors do not need to manually transform data. Furthermore, because Data Formulator's Chart Builder resembles tools like Power BI and Tableau, it lets the authors focus on high-level designs. Data Formulator's multi-modal interaction approach supports both derivation and reshaping tables. While Data Formulator currently focuses on standard visualization supported by Vega-Lite, its AI-powered concept-driven approach can also work with expressive and creative visualization design tools like StructGraph [48] and Data Illustrator [27] to automate data transformations.

Data Transformation Tools. Libraries and tools like tidyverse [55], pandas [33], Potter's Wheel [38], Wrangler [17], Tableau Prep, and Power Query are developed to support data transformation. They introduce operators to reshape, compute, and manipulate tabular data needed in data analysis. Automated data transformation tools, including programming-by-example tools [16, 36, 50] and initiative tools [13, 17, 18, 60], are developed to reduce authors' specification effort. Data Formulator tailors key transformation operators from the tidyverse library (reshaping and derivation) for visualization authoring. Because the desired data shape changes with visualization goals, even with these tools, authors still need the knowledge and effort to first identify the desired data shape, and then switch tools to transform the data. Data Formulator bridges visual encoding and data transformation with data concepts to reduce this overhead.

Code Generation. Code generation models [7, 9, 10] and program synthesis techniques [6, 12, 50, 61] enable users to complete tasks without programming by using easier specifications, including natural language, examples, and demonstrations. Code generation models like Codex [7], PaLM [9], and InCoder [10] are transformer-based causal language models (commonly referred to as LLMs) that complete texts from natural language prompts. These LLMs can generate expressive programs to solve competitive programming [14, 24], data science [20], and software engineering tasks [1] from high-level descriptions. Programming-by-example [52] and programming-by-demonstration [2, 37] tools can synthesize programs based on users' output examples or demonstrations that illustrate the computation process. Natural language approaches are highly expressive, but some tasks can be challenging to phrase. On the other hand, while programming-by-example techniques are precise, they are less expressive and do not scale to large programs as they require well-defined program spaces. Therefore, Data Formulator adopts a mixed-modality approach to solve the data transformation task. It leverages the Codex model [7] for concept derivations and the example-based synthesis algorithm [51] for reshaping, which takes advantage of both approaches to reduce authors' specification overhead.

Because code generation techniques generalize programs from incomplete user specifications, generated programs are inherently ambiguous, and thus require disambiguation to identify a correct solution among candidates. Prior work proposes techniques to visualize the search process [61], visualize code candidates [52, 59], and present distinguishing examples for authors to inspect [15]. Data Formulator provides feedback to the authors by presenting the generated code together with its execution results for them to inspect, select, and edit.

6 Discussion and Future Work

Unified Interaction with Multiple Modalities Data Formulator employs two different modalities for authors to specify different types of data transformation: natural language for concept derivation and examples for table reshaping (Sec. 3). This design combines strengths of both modalities so that the authors can better communicate their intent with the AI agent, and the AI agent can provide precise solutions from a more expressive program space. However, choosing the right input modality when creating a new concept can be challenging for inexperienced authors. To address this challenge, we envision a stratified approach where the authors just initiate the interaction in natural language, and the AI agent will decide whether to ask the authors, for example relations for clarification or to directly generate derivation codes. This design will shift the effort of deciding which approach to start with from the authors to the AI agent, and "by-example" specification will become a followup interaction step to help the authors clarify their intent. We envision this mixed-initiative unified interaction will further reduce the authors' efforts in visualization authoring.

Conversational Visualization Authoring with AI Agents. Conversational AI agents [32] have the strength of leveraging the interaction contexts to better interpret user intent. They also provide opportunities for users to refine the intent when the task is complex or ambiguous. However, conversation with only natural language is often insufficient for visualization authoring because (1) it does not provide users with precise control over the authoring details (e.g., exploring different encoding options, changing design styles) and (2) the results can be challenging to inspect and verify without concrete artifacts (e.g., programs, transformed data). It would be useful to research how conversational AI can be integrated with Data Formulator's concept-driven approach to improve the overall visualization experiences. First, with a conversational AI agent, the authors can incrementally specify and refine their intent for tasks that are difficult to solve in one shot. Second, a conversational agent complements Data Formulator by helping the authors explore and configure chart options. Because Data Formulator focuses on data transformation, it does not expose many chart options (e.g., axis labeling, legend style, visual mark styles) in its interface. A conversational AI agent can help the authors access and control these options without overwhelming them with complex menus. For example, when the authors describe chart styles they would like to change, Data Formulator can apply the options directly or dynamically generate editing panels for them to control. We envision the effective combination of conversational AI experiences, and the Data Formulator approach will let the authors confidently specify richer designs with less effort.

Concept-driven Visual Data Exploration. Visual data exploration tools [21, 22, 30, 57, 58] help data scientists understand data and gain meaningful insights in their analysis process. These tools support a rich visual visualization space, yet still require datasets to be in the appropriate shape and schema. While Data Formulator is designed for visualization authoring, its concept-driven approach can be used in visual data exploration to expand the design space. Beyond the current concept-driven features of Data Formulator, the AI agent could be enhanced to recommend data concepts of interest based on the data context or author interaction history. Building on this idea, the tool could recommend charts based on all potentially relevant data concepts. This expansive leap could overcome one of the limitations of chart recommendation systems: by enabling the authors to view charts beyond their input data columns without additional user intervention.

Study Limitations. While our participants had varying levels of expertise in chart authoring, computer programming, and experience with LLMs, many of them had considerable knowledge about data transformation methodology and programming. It would be useful to investigate if and how people with limited expertise could learn and use Data Formulator. The main goal of Data Formulator was to reduce manual data transformation in visualization authoring efforts. As such, in our study, we focused on derivation and reshaping types of data transformations with simple datasets. While they are key types of transformation and our tasks covered multiple styles of derivations, the transformations we studied are by no means comprehensive. It would be valuable to evaluate the broader combinations and complexities of data transformations. Our study adopted a chart reproduction study [40], which is commonly used for evaluating chart authoring systems (e.g., [27, 39, 42]). Therefore, our study shares its inherent limitations: because we prepared datasets and tasks, and provided target visualizations as a reference, we do not know if and how people would use Data Formulator to create visualizations with their own data.

7 Conclusion

This paper introduces Data Formulator, a concept-driven visualization authoring tool that leverages an AI agent to address the data transformation challenge in visualization authoring. With Data Formulator, authors work with the AI agent to create new data concepts and then map data concepts to visual channels to specify visualizations. Data Formulator then automatically transforms the input data and instantiates the visualizations. Throughout the authoring process, Data Formulator provides feedback to the authors to inspect and refine the generated code to promote confidence. As discovered in the chart reproduction study, participants can learn and use Data Formulator to create visualizations that require advanced data transformations. In the future, the concept-driven visualization approach can potentially benefit the design of new visual data exploration tools and expressive visualization authoring tools to overcome the data transformation barrier.

A Supplemental Material

We include three zip files in the supplemental material: (1) a 6-minute video that walks through user experiences of creating visualizations about Seattle and Atlanta temperatures, described in Sec. 2 with Data Formulator, (2) a set of short videos that demonstrate additional Data Formulator scenarios, and (3) our user study materials including: study script, tutorials, study tasks, and prompts created by participants.

References

[1] S. Barke, M. B. James, and N. Polikarpova. Grounded copilot: How programmers interact with code-generating models. *Proc. ACM Program. Lang.*, 7(OOPSLA1):85–111, 2023. doi: 10.1145/3586030 9

[2] S. Barman, S. E. Chasins, R. Bodík, and S. Gulwani. Ringer: web automation by demonstration. In E. Visser and Y. Smaragdakis, eds., *OOPSLA 2016, part of SPLASH 2016, Amsterdam, The Netherlands, October 30 - November 4, 2016*, pp. 748–764. ACM, 2016. doi: 10.1145/2983990.2984020 9

[3] L. Bartram, M. Correll, and M. Tory. Untidy data: The unreasonable effectiveness of tables. *IEEE Trans. Vis. Comput. Graph.*, 28(1):686–696, 2022. doi: 10.1109/TVCG.2021.3114830 1

[4] M. Bostock and J. Heer. Protovis: A graphical toolkit for visualization. *IEEE Trans. Vis. Comput. Graph.*, 15(6):1121–1128, 2009. doi: 10.1109/TVCG.2009.174 8

[5] M. Bostock, V. Ogievetsky, and J. Heer. D^3 data-driven documents. *IEEE Trans. Vis. Comput. Graph.*, 17(12):2301–2309, 2011. doi: 10.1109/TVCG.2011.185 8

[6] S. Chaudhuri, K. Ellis, O. Polozov, R. Singh, A. Solar-Lezama, and Y. Yue. Neurosymbolic programming. *Found. Trends Program. Lang.*, 7(3):158–243, 2021. doi: 10.1561/2500000049 9

[7] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. de Oliveira Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, et al. Evaluating large language models trained on code. *CoRR*, abs/2107.03374, 2021. 6, 8, 9

[8] Q. Chen, S. Pailoor, C. Barnaby, A. Criswell, C. Wang, G. Durrett, and I. Dillig. Type-directed synthesis of visualizations from natural language queries. *Proc. ACM Program. Lang.*, 6(OOPSLA2):532–559, 2022. doi: 10.1145/3563307 8

[9] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, and more. Palm: Scaling language modeling with pathways. *CoRR*, abs/2204.02311, 2022. doi: 10.48550/arXiv.2204.02311 9

[10] D. Fried, A. Aghajanyan, J. Lin, S. Wang, E. Wallace, F. Shi, R. Zhong, W. Yih, L. Zettlemoyer, and M. Lewis. Incoder: A generative model for code infilling and synthesis. *CoRR*, abs/2204.05999, 2022. doi: 10.48550/arXiv.2204.05999 9

[11] T. Gao, M. Dontcheva, E. Adar, Z. Liu, and K. G. Karahalios. Datatone: Managing ambiguity in natural language interfaces for data visualization. In C. Latulipe, B. Hartmann, and T. Grossman, eds., *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology, UIST 2015, Charlotte, NC, USA, November 8-11, 2015*, pp. 489–500. ACM, 2015. doi: 10.1145/2807442.2807478 8

[12] S. Gulwani, O. Polozov, and R. Singh. Program synthesis. *Found. Trends Program. Lang.*, 4(1-2):1–119, 2017. doi: 10.1561/2500000010 9

[13] Y. He, Z. Jin, and S. Chaudhuri. Auto-transform: Learning-to-transform by patterns. *Proc. VLDB Endow.*, 13(11):2368–2381, 2020. 9

[14] D. Hendrycks, S. Basart, S. Kadavath, M. Mazeika, A. Arora, E. Guo, C. Burns, S. Puranik, H. He, D. Song, and J. Steinhardt. Measuring coding challenge competence with APPS. In J. Vanschoren and S. Yeung, eds., *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*, 2021. 9

[15] R. Ji, J. Liang, Y. Xiong, L. Zhang, and Z. Hu. Question selection for interactive program synthesis. In A. F. Donaldson and E. Torlak, eds., *Proc. of the 41st ACM SIGPLAN International Conference on Programming Language Design and Implementation, PLDI 2020, London, UK, June 15-20, 2020*, pp. 1143–1158. ACM. doi: 10.1145/3385412.3386025 9

[16] Z. Jin, M. R. Anderson, M. J. Cafarella, and H. V. Jagadish. Foofah: Transforming data by example. In S. Salihoglu, W. Zhou, R. Chirkova, J. Yang, and D. Suciu, eds., *SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017*, pp. 683–698. ACM, 2017. doi: 10.1145/3035918.3064034 9

[17] 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 (CHI)*, pp. 3363–3372, 2011. doi: 10.1145/1978942.1979444 1, 9

[18] 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. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*, UIST '20, p. 140–151. Association for Computing Machinery, New York, NY, USA, 2020. doi: 10.1145/3379337.3415842 9

[19] D. H. Kim, E. Hoque, and M. Agrawala. Answering questions about charts and generating visual explanations. In R. Bernhaupt, F. F. Mueller, D. Verweij, J. Andres, J. McGrenere, A. Cockburn, I. Avellino, A. Goguey, P. Bjøn, S. Zhao, B. P. Samson, and R. Kocielnik, eds., *CHI '20: Honolulu, HI, USA, April 25-30, 2020*, pp. 1–13. ACM. doi: 10.1145/3313831.3376467 8

[20] Y. Lai, C. Li, Y. Wang, T. Zhang, R. Zhong, L. Zettlemoyer, S. W. Yih, D. Fried, S. I. Wang, and T. Yu. DS-1000: A natural and reliable benchmark for data science code generation. *CoRR*, abs/2211.11501, 2022. doi: 10.48550/arXiv.2211.11501 9

[21] D. J. L. Lee, V. Setlur, M. Tory, K. Karahalios, and A. G. Parameswaran. Deconstructing categorization in visualization recommendation: A taxonomy and comparative study. *IEEE Trans. Vis. Comput. Graph.*, 28(12):4225–4239, 2022. doi: 10.1109/TVCG.2021.3085751 9

[22] D. J. L. Lee, D. Tang, K. Agarwal, T. Boonmark, C. Chen, J. Kang, U. Mukhopadhyay, J. Song, M. Yong, M. A. Hearst, and A. G. Parameswaran. Lux: Always-on visualization recommendations for exploratory dataframe workflows. *Proc. VLDB Endow.*, 15(3):727–738, 2021. doi: 10.14778/3494124.3494151 9

[23] T. J. Li, M. Radensky, J. Jia, K. Singarajah, T. M. Mitchell, and B. A. Myers. PUMICE: A multi-modal agent that learns concepts and conditionals from natural language and demonstrations. In F. Guimbretière, M. S. Bernstein, and K. Reinecke, eds., *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology, UIST 2019, New Orleans, LA, USA, October 20-23, 2019*, pp. 577–589. ACM, 2019. doi: 10.1145/3332165.3347899 8

[24] Y. Li, D. Choi, J. Chung, N. Kushman, J. Schrittwieser, R. Leblond, T. Eccles, J. Keeling, F. Gimeno, A. Dal Lago, et al. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097, 2022. 9

[25] C. Liu, Y. Han, R. Jiang, and X. Yuan. Advisor: Automatic visualization answer for natural-language question on tabular data. In *14th IEEE Pacific Visualization Symposium, PacificVis 2021, Tianjin, China, April 19-21, 2021*, pp. 11–20. IEEE. doi: 10.1109/PacificVis52677.2021.00010 8

[26] Z. Liu, C. Chen, F. Morales, and Y. Zhao. Atlas: Grammar-based procedural generation of data visualizations. In *2021 IEEE Visualization Conference, 2021 - Short Papers, New Orleans, LA, USA, October 24-29, 2021*, pp. 171–175. IEEE, 2021. doi: 10.1109/VIS49827.2021.9623315 8

[27] 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 (CHI)*, pp. 123:1–13, 2018. doi: 10.1145/3173574.3173697 1, 8, 9

[28] Y. Luo, N. Tang, G. Li, J. Tang, C. Chai, and X. Qin. Natural language to visualization by neural machine translation. *IEEE Trans. Vis. Comput. Graph.*, 28(1):217–226, 2022. doi: 10.1109/TVCG.2021.3114848 8

[29] D. Moritz, B. Howe, and J. Heer. Falcon: Balancing interactive latency and resolution sensitivity for scalable linked visualizations. In S. A. Brewster, G. Fitzpatrick, A. L. Cox, and V. Kostakos, eds., *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems, CHI 2019, Glasgow, Scotland, UK, May 04-09, 2019*, p. 694. ACM, 2019. doi: 10.1145/3290605.3300924 7

[30] 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 Trans. Vis. Comput. Graph.*, 25(1):438–448, 2019. doi: 10.1109/TVCG.2018.2865240 9

[31] A. Narechania, A. Srinivasan, and J. T. Stasko. NL4DV: A toolkit for generating analytic specifications for data visualization from natural language queries. *IEEE Trans. Vis. Comput. Graph.*, 27(2):369–379, 2021. doi: 10.1109/TVCG.2020.3030378 8

[32] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744, 2022. 9

[33] T. pandas development team. pandas-dev/pandas: Pandas, Mar. 2023. doi: 10.5281/zenodo.7741580 1, 9

[34] P. Pasupat and P. Liang. Compositional semantic parsing on semi-structured tables. In *ACL 2015, July 26-31, 2015, Beijing, China, Volume 1: Long Papers*, pp. 1470–1480. The Association for Computer Linguistics, 2015. doi: 10.3115/v1/p15-1142 8

[35] G. Poesia, A. Polozov, V. Le, A. Tiwari, G. Soares, C. Meek, and S. Gulwani. Synchromesh: Reliable code generation from pre-trained language models. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*, 2022. 8

[36] O. Polozov and S. Gulwani. Flashmeta: a framework for inductive program synthesis. In J. Aldrich and P. Eugster, eds., *Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2015, Pittsburgh, PA, USA, October 25-30, 2015*, pp. 107–126. ACM, 2015. doi: 10.1145/2814270.2814310 9

[37] K. Pu, R. Fu, R. Dong, X. Wang, Y. Chen, and T. Grossman. Semanticon: Specifying content-based semantic conditions for web automation programs. In M. Agrawala, J. O. Wobbrock, E. Adar, and V. Setlur, eds., *The 35th Annual ACM Symposium on User Interface Software and Technology, UIST 2022, Bend, OR, USA, 29 October 2022 - 2 November 2022*, pp. 63:1–63:16. ACM, 2022. doi: 10.1145/3526113.3545691 9

[38] V. Raman and J. M. Hellerstein. Potter's wheel: An interactive data cleaning system. In P. M. G. Apers, P. Atzeni, S. Ceri, S. Paraboschi, K. Ramamohanarao, and R. T. Snodgrass, eds., *VLDB 2001, Proceedings of 27th International Conference on Very Large Data Bases, September 11-14, 2001, Roma, Italy*, pp. 381–390. Morgan Kaufmann, 2001. 9

[39] D. Ren, B. Lee, and M. Brehmer. Charticulator: Interactive construction of bespoke chart layouts. *IEEE Trans. Vis. Comput. Graph. (Proceedings of InfoVis)*, 25(1), 2019. doi: 10.1109/TVCG.2018.2865158 1, 8, 9

[40] D. Ren, B. Lee, M. Brehmer, and N. H. Riche. Reflecting on the evaluation of visualization authoring systems : Position paper. In M. Sedlmair, P. Isenberg, M. Meyer, and T. Isenberg, eds., *2018 IEEE Evaluation and Beyond - Methodological Approaches for Visualization, BELIV 2018, Berlin, Germany, October 21, 2018*, pp. 86–92. IEEE Computer Society, 2018. doi: 10.1109/BELIV.2018.8634297 7, 9

[41] B. Saket, H. Kim, E. T. Brown, and A. Endert. Visualization by demonstration: An interaction paradigm for visual data exploration. *IEEE Trans. Vis. Comput. Graph.*, 23(1):331–340, 2017. doi: 10.1109/TVCG.2016.2598839 8

[42] A. Satyanarayan and J. Heer. Lyra: An interactive visualization design environment. *Computer Graphics Forum (Proceedings of EuroVis)*, 33(3), 2014. doi: 10.1111/cgf.12391 1, 8, 9

[43] A. Satyanarayan, B. Lee, D. Ren, J. Heer, J. T. Stasko, J. Thompson, M. Brehmer, and Z. Liu. Critical reflections on visualization authoring systems. *IEEE Trans. Vis. Comput. Graph.*, 26(1):461–471, 2020. doi: 10.1109/TVCG.2019.2934281 8

[44] A. Satyanarayan, D. Moritz, K. Wongsuphasawat, and J. Heer. Vega-Lite: A grammar of interactive graphics. *IEEE Transactions on Visualization and Computer Graphics (Proceedings of InfoVis)*, 23(1):341–350, 2017. doi: 10.1109/TVCG.2016.2599030 1, 8

[45] L. Shen, E. Shen, Z. Tai, Y. Wang, Y. Luo, and J. Wang. GALVIS: visualization construction through example-powered declarative programming. In M. A. Hasan and L. Xiong, eds., *Proceedings of the 31st ACM International Conference on Information & Knowledge Management, Atlanta, GA, USA, October 17-21, 2022*, pp. 4975–4979. ACM, 2022. doi: 10.1145/3511808.3557159 8

[46] A. Srinivasan, N. Nyapathy, B. Lee, S. M. Drucker, and J. T. Stasko. Collecting and characterizing natural language utterances for specifying data visualizations. In Y. Kitamura, A. Quigley, K. Isbister, T. Igarashi, P. Bjørn, and S. M. Drucker, eds., *CHI '21: CHI Conference on Human Factors in Computing Systems, Virtual Event / Yokohama, Japan, May 8-13, 2021*, pp. 464:1–10. ACM, 2021. doi: 10.1145/3411764.3445400 8

[47] C. Stolte, D. Tang, and P. Hanrahan. Query, analysis, and visualization of hierarchically structured data using polaris. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, July 23-26, 2002, Edmonton, Alberta, Canada*, pp. 112–122. ACM, 2002. doi: 10.1145/775047.775064 8

[48] T. Tsandilas. Structgraphics: Flexible visualization design through data-agnostic and reusable graphical structures. vol. 27, pp. 315–325, 2021. doi: 10.1109/TVCG.2020.3030476 9

[49] 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. vol. 3, p. 1057, 2018. doi: 10.21105/joss.01057 8

[50] C. Wang, A. Cheung, and R. Bodík. Synthesizing highly expressive SQL queries from input-output examples. In A. Cohen and M. T. Vechev, eds., *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2017, Barcelona, Spain, June 18-23, 2017*, pp. 452–466. ACM, 2017. doi: 10.1145/3062341.3062365 9

[51] C. Wang, Y. Feng, R. Bodík, A. Cheung, and I. Dillig. Visualization by example. *Proc. ACM Program. Lang.*, 4(POPL):49:1–49:28, 2020. doi: 10.1145/3371117 9

[52] C. Wang, Y. Feng, R. Bodík, I. Dillig, A. Cheung, and A. J. Ko. Falx: Synthesis-powered visualization authoring. In Y. Kitamura, A. Quigley, K. Isbister, T. Igarashi, P. Bjørn, and S. M. Drucker, eds., *CHI '21: CHI Conference on Human Factors in Computing Systems, Virtual Event / Yokohama, Japan, May 8-13, 2021*, pp. 106:1–106:15. ACM, 2021. doi: 10.1145/3411764.3445249 8, 9

[53] H. Wickham. *ggplot2 - Elegant Graphics for Data Analysis*. Use R. Springer, 2009. doi: 10.1007/978-0-387-98141-3 1, 8

[54] H. Wickham. Tidy data. *The Journal of Statistical Software*, 59, 2014. 1

[55] H. Wickham, M. Averick, J. Bryan, W. Chang, L. McGowan, R. François, G. Grolemund, A. Hayes, L. Henry, J. Hester, M. Kuhn, T. Pedersen, E. Miller, S. Bache, K. Müller, J. Ooms, D. Robinson, D. Seidel, V. Spinu, K. Takahashi, D. Vaughan, C. Wilke, K. Woo, and H. Yutani. Welcome to the tidyverse. *J. Open Source Softw.*, 4(43):1686, Nov. 2019. doi: 10.21105/joss.01686 1, 9

[56] L. Wilkinson. *The Grammar of Graphics, Second Edition*. Statistics and computing. Springer, 2005. 8

[57] K. Wongsuphasawat, D. Moritz, A. Anand, J. D. Mackinlay, B. Howe, and J. Heer. Voyager: Exploratory analysis via faceted browsing of visualization recommendations. *IEEE Trans. Vis. Comput. Graph.*, 22(1):649–658, 2016. doi: 10.1109/TVCG.2015.2467191 9

[58] 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 2017 CHI Conference on Human Factors in Computing Systems*, CHI '17, p. 2648–2659. Association for Computing Machinery, New York, NY, USA, 2017. doi: 10.1145/3025453.3025768 9

[59] K. Xiong, Z. Luo, S. Fu, Y. Wang, M. Xu, and Y. Wu. Revealing the semantics of data wrangling scripts with comantics. *IEEE Trans. Vis. Comput. Graph.*, 29(1), 2023. doi: 10.1109/TVCG.2022.3209470 9

[60] C. Yan and Y. He. Auto-suggest: Learning-to-recommend data preparation steps using data science notebooks. In D. Maier, R. Pottinger, A. Doan, W. Tan, A. Alawini, and H. Q. Ngo, eds., *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference Portland, OR, USA, June 14-19, 2020*, pp. 1539–1554. ACM. doi: 10.1145/3318464.3389738 9

[61] T. Zhang, Z. Chen, Y. Zhu, P. Vaithilingam, X. Wang, and E. L. Glassman. Interpretable program synthesis. In Y. Kitamura, A. Quigley, K. Isbister, T. Igarashi, P. Bjørn, and S. M. Drucker, eds., *CHI '21: CHI Conference on Human Factors in Computing Systems, Virtual Event / Yokohama, Japan, May 8-13, 2021*, pp. 105:1–16. ACM. doi: 10.1145/3411764.3445646 9

[62] J. Zong, D. Barnwal, R. Neogy, and A. Satyanarayan. Lyra 2: Designing interactive visualizations by demonstration. *IEEE Trans. Vis. Comput. Graph.*, 27(2):304–314, 2021. doi: 10.1109/TVCG.2020.3030367 8

[63] J. Zong, J. Pollock, D. Wootton, and A. Satyanarayan. Animated vega-lite: Unifying animation with a grammar of interactive graphics. *IEEE Trans. Vis. Comput. Graph.*, 29(1):149–159, 2023. doi: 10.1109/TVCG.2022.3209369 8