DeepEye: Creating Good Data Visualizations by Keyword Search

Yuyu Luo[†], Xuedi Qin[†], Nan Tang[‡], Guoliang Li[†], Xinran Wang[†]

[†]Department of Computer Science, Tsinghua University [‡]Qatar Computing Research Institute, HBKU
{luoyuyu@mail., qxd17@mails., liguoliang@, wangxr14@mails.}tsinghua.edu.cn, ntang@hbku.edu.qa

ABSTRACT

Creating good visualizations for ordinary users is hard, even with the help of the state-of-the-art interactive data visualization tools, such as Tableau, Qlik, because they require the users to understand the data and visualizations very well. DeepEye is an innovative visualization system that aims at helping everyone create good visualizations simply like a Google search. Given a dataset and a keyword query, DeepEye understands the query intent, generates and ranks good visualizations. The user can pick the one she likes and do a further faceted navigation to easily navigate the candidate visualizations. In this demonstration, the attendees will have the opportunity to experience the following features: (1) *visualization recommendation* – Our system can automatically recommends meaningful visualizations by learning from existing known datasets and good visualizations; (2) *keyword search* – The attendee can pose text queries for specifying what visualizations she wants (*e.g.,* trends) without specifying how to generate them; (3) *faceted navigation* – One can further refine the results by a click-based faceted navigation to find other relevant and interesting visualizations.

ACM Reference Format:
Yuyu Luo[†], Xuedi Qin[†], Nan Tang[‡], Guoliang Li[†], Xinran Wang[†] . 2018. DeepEye: Creating Good Data Visualizations by Keyword Search. In *SIGMOD'18: 2018 International Conference on Management of Data, June 10–15, 2018, Houston, TX, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3183713.3193545

1 INTRODUCTION

Nowadays, the ability to create good visualizations has shifted from a nice-to-have skill to a must-have skill for all data analysts. However, the overwhelming choices of interactive data visualization tools (e.g., Tableau, Qlik and D3 [2]) only allow *experts* to create good visualizations, assuming that the experts know many details: the meaning and the distribution of the data, the right combination of attributes, and the right type of charts – these requirements are apparently not easy, even for experts.

Challenges. Not surprisedly, creating good data visualization is hard in practice. From the user perspective, there are many possible ways of visualizations for a given dataset (for example, different attribute combinations and visualization types), and many ways of transforming data (for example, grouping, binning, sorting, and a combination thereof) – these make it infeasible for the user to enumerate all possible visualizations and select the ones she needs. From the system perspective, among numerous problems, no consensus has emerged to quantify the "goodness" of a visualization. What makes it much harder is when the system does not even know what the user wants.

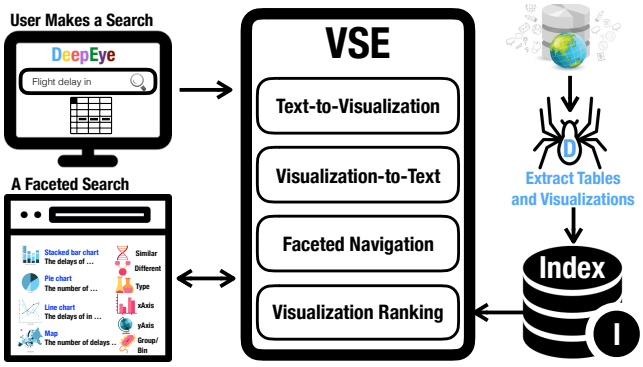


Figure 1: The architecture of DeepEye

DeepEye belongs to the line of new research, namely visualization recommendation systems [8–10]. Informally speaking, given a dataset, these systems automatically recommend visualizations to the user under different criteria, such as relevance, surprise, non-obviousness, diversity and coverage. However, as pointed out by [1], despite the many attempts and efforts, these systems may still mislead the user, by generating visualizations that might be worse than nothing, since it is basically impossible to guess a user's query intent from nothing.

Our Methodology. DeepEye allows the user to specify her query intent by keyword (or text) search (Figure 1), which will be converted to our internal visualization language for generating candidate visualizations. Note that the user's text query is typically underspecified and ambiguous. DeepEye solves the fundamental cognitive science problem for quantifying and ranking good visualizations with a novel idea *visualization by examples*. More specifically, based on the plenty of generic priors to showcase great visualizations, DeepEye trains binary classifiers to determine whether to visualize a given dataset with a specific visualization type (*e.g.,* bar charts or line charts) is meaningful, and it uses a supervised learning-to-rank model to rank good visualizations. It also provides the explanation of the visualizations with plain text, for easy understanding. In addition, DeepEye also allows the user to navigate the other visualizations by faceted navigation. That is, based on a seed visualization that the user picks, instead of guessing what

SIGMOD'18, June 10–15, 2018, Houston, TX, USA

ACM ISBN 978-1-4503-4703-7/18/06…$15.00
https://doi.org/10.1145/3183713.3193545

[1]Guoliang Li is the corresponding author.

other visualizations she might be interested in, DeepEye lets the user specify her preference via a click-based faceted navigation.

This demo implements the basic ideas of our vision paper [6], which extends our work of automatic data visualization [5] to support keyword search and faceted search. As mentioned above, DeepEye differs from other visualization recommendation systems in three main aspects: (1) instead of guessing a user's intent, it accepts an underspecified text search; (2) DeepEye ranks visualizations of a new dataset by learning from existing known datasets and good visualizations; and (3) it supports faceted navigation that can help users easily find more good visualizations of interest.

2 SYSTEM OVERVIEW

Overview. The architecture of DeepEye is given in Figure 1. Bootstrapped with a user posed text query, DeepEye translates this text query to multiple candidate visualizations, discovers and ranks good (*i.e.,* candidate) visualizations, and returns the top ranked ones. When the user selects a visualization and further explores by clicking a facet navigation, DeepEye discovers more visualizations and helps the user easily explore her desired visualizations. The user may iterate over the above processes until she finds all visualizations of her interest.

Datasets. We focus on relational tables. For simplicity, we consider a single table, denoted by D, and our techniques can be easily extended to support multiple tables by joins (see SeeDB [10]).

Visualization Queries. We focus on the popular visualization types (*e.g.,* pie/line/bar/scatter charts) for a given dataset D. For *declarative visualization language*, we use a high-level grammar that enables rapid specification of interactive data visualizations. To facilitate our discussion, we define a simple language that can capture all possible visualizations. (Note that our system can support any declarative visualization query language, *e.g.,* Vega-Lite [7].) Figure 2 shows our language for specifying visualization queries for two columns (please find more details in [5]).

VISUALIZE	TYPE ($\in$ {bar, pie, line, scatter})
SELECT	X', Y' ($X' \in \{X, \text{BIN}(X)\}$, $Y' \in \{Y, \text{AGG}(Y)\}$)
FROM	D
TRANSFORM	X (using an operator $\in$ {BIN, GROUP})
ORDER BY	X', Y'
WHERE	X' OP v

Figure 2: Visualization language (two columns)

▷ **VISUALIZE**: specifies the visualization type

▷ **SELECT**: extracts the selected columns

- X'/Y' relates to X/Y: X' is either X or binning values, *e.g.,* binning by hour; Y' is either Y or the aggregation values (*e.g.,* AGG = {SUM, AVG, CNT}) after transforming X

▷ **FROM**: the source table

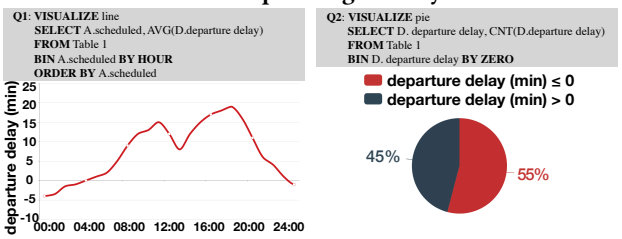
▷ *TRANSFORM*: transforms the selected columns

- Binning: Binning X into several buckets, *e.g.,* by month.
- Grouping: GROUP BY X

▷ *ORDER BY*: sorts the selected column, *i.e.,* ORDER BY X'/Y'

A. scheduled	B. carrier	C. destination city name	D. departure delay (min)	E. arrival delay (min)	F. passengers	...
01-Jan 00:04	AA	New York	-5	2	173	...
01-Jan 06:43	MQ	Atlanta	9	-2	132	...
01-Jan 09:30	EV	Minneapolis	13	17	127	...
01-Jan 11:30	AA	Boston	22	10	141	...
01-Jan 17:59	MQ	New York	19	13	232	...
01-Jan 23:26	UA	Los Angeles	0	-2	119	...
...	...	...	...	...	...	...

Table 1: An excerpt of flight delay statistics



(a) Flight delay *w.r.t.* scheduled time (b) Proportions of departure delays

Figure 3: Examples of query-to-visualization

▷ *WHERE*: selects the value satisfying a condition, *i.e.,* > 100.

Considering a sample table in Table 1 about flight delay statistics, the queries Q_1 and Q_2 will produce a line chart and a pie chart respectively, as shown in Figure 3.

We will use chart and visualization interchangeably in the paper, which corresponds to a query Q over D, denoted by $Q(D)$.

Visualization Crawler. The DeepEye *crawler* extracts tables and their associated visualizations from multiple sources, where *both* the table *and* the visualization specifications (or equivalently, queries) are explicitly given[2]. For example, there are hundreds of visualization examples in https://www.highcharts.com, with both datasets and visualization specifications.

3 DEMONSTRATION OVERVIEW

A picture is worth a thousand words. A good visualization is worth a terabyte of data. In this demonstration, we will show how easy it is to find interesting visualizations to tell stories of a dataset.

Datasets. We have prepared hundreds of real-world datasets whose semantics is easy to understand by general audience. Table 2 shows three datasets that we will use in our demonstration, together with their descriptions.

No.	Name	Description
D_1	*Flight Delay Statistics*	Flight delay data with scheduled, carrier name, destination, departure, departure delay, arrival delay, and number of passengers.
D_2	*Electricity Consumption Statistics*	This dataset reflects the electricity consumption of Chinese metropolis during 2016.
D_3	*Foreigners Visit of China*	This dataset describes each visitor's age, gender, nationality, continent, purpose of entry, entry methods, dwell time, and etc.

Table 2: Three real-world datasets

Next, we will use an example (see Figure 4) to explain how DeepEye works, using the *Flight Delay Statistics* table in Table 1. We will postpone the discussion for some technical details to Section 4.

[2]If the queries in the data sources are different from our queries, we can use rule-based methods to transform their queries to our queries.

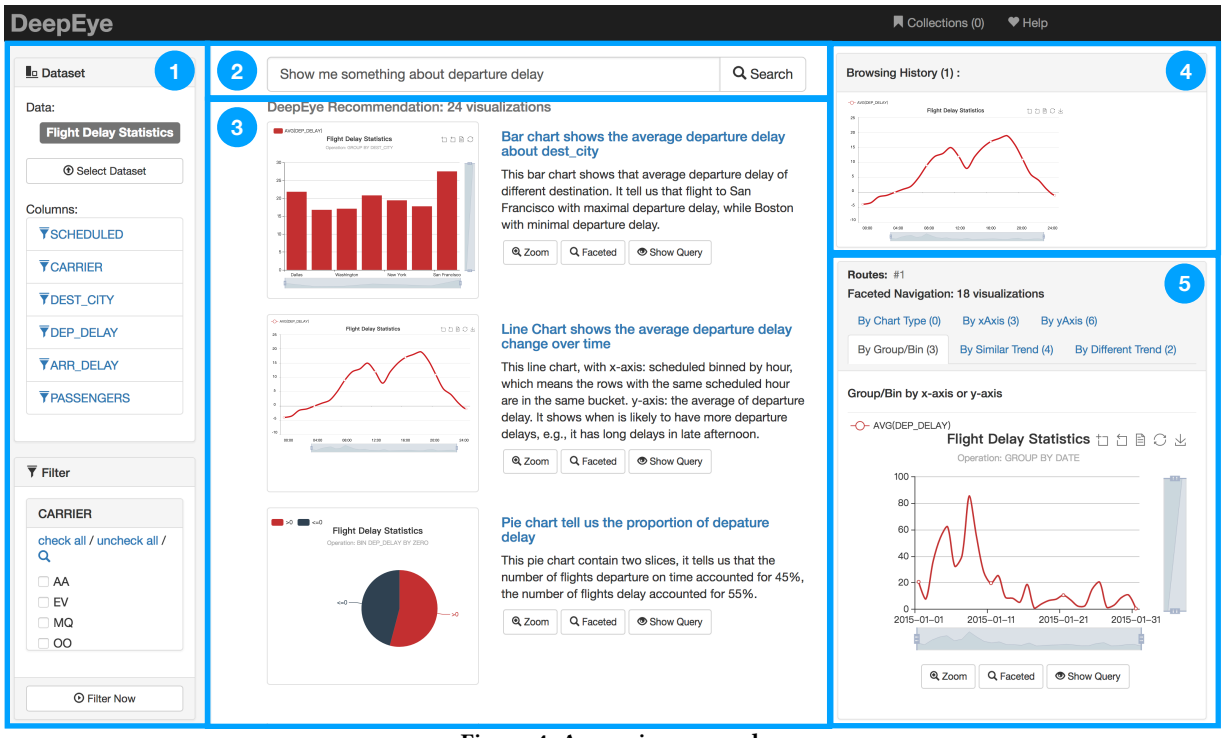


Figure 4: A running example

Dataset Specification. The user can upload a new table, or select/filter an existing table, which is to be visualized (see Figure 4–❶). For example, after the user selects the *Flight Delay Statistics* table, she can filter data by the *"Filter"* panel if necessary, *e.g.,* removing those *CARRIER*s that she does not care about, which can significantly reduce the search space for the back-end algorithms.

Creating Good Visualizations. A user can create good visualizations based on the following steps.

(i) **Text search.** The user can pose a text query and get some relevant visualizations she wants. For example, given a query "*Show me something about departure delay*" (see Figure 4–❷), DeepEye will recommend visualizations relevant to departure delay. That is, DeepEye will fix one attribute *departure delay* and discovers other attributes that when being combined with attribute *departure delay* using an appropriate type of chart, will produce good visualizations.

(ii) **Visualization recommendation.** DeepEye will rank the candidate (good) visualizations. It also provides the explanations in plain text for each recommended visualization, which will help the user to better understand the visualizations (see Figure 4–❸). The user can click the *"Zoom"* button for more visualization details.

(iii) **Faceted navigation.** The user may select one good visualization, and conduct a further faceted navigation to find other visualizations of her interest. Suppose that the user selects the *line chart* (Figure 4–❸) by clicking the *"Faceted"* button, DeepEye will suggest appropriate facets and return visualizations to the user. We can see from Figure 4–❺ that, the suggested facets for this selected visualization are chart type, x-axis, y-axis, group/bin, similar, and different. The first chart under the facet "By Group/Bin" is a line chart which bins x-axis into bucket by date. It shows the trend of average departure delay during Jan 2015. Note that, the user can do a further faceted navigation iteratively, which may get more meaningful visualizations. Figure 4–❹ keeps track of the visualizations that the user already selected.

(iv) **Interactive Refinement.** DeepEye also supports popular interactions such as zoom in, zoom out, and mark, by leveraging an interactive visualization library ECHARTS (http://echarts.baidu.com). The user can click the *"Show Query"* button (Figure 4–❸) to check the DeepEye visualization query and can also customize her visualization by modifying the DeepEye query, for expert users.

4 TECHNICAL DETAILS

In this section, we will discuss some technical details of different modules (Figure 1) for the visualization search engine of DeepEye.

Text-to-Visualization. Given a text query K and a dataset D, it generates all candidate visualizations that are possibly good, which has two steps: *text-to-queries* and *visualization recognition*.

(1) Text-to-Queries. Given a text query, we aim at generating a set $\mathbf{Q}$ of visualization queries that correspond to a set $\mathbf{V}$ of visualizations. As mentioned earlier, this problem is hard because text queries are always ambiguous and underspecified – there might have a large number of candidate visualizations due to different attribute combinations and multiple data transformation operations (*e.g.,* grouping, binning, sorting). To address this problem, for each text query, we first identify its matching type - reserved keywords in our query, table name, attribute name, values in attributes, or some constant values [4]. We then feed them into the visualization queries, *e.g.,* feeding attribute name to X or Y, table name to D, and get all possible visualization queries.

(2) Visualization Recognition. Some visualization queries may be meaningless, *e.g.,* it's not suitable for pie chart to visualize temporal

data. Hence, we want to find a set of good visualizations $\mathbf{V}' \subseteq \mathbf{V}$. This problem is also hard because there is no consensus to quantify that which visualization is good. Our solution to capture human perception for visualization recognition is by learning from examples. The hypothesis about what are learned from generic priors can be applied to different domains is that the explanatory factors behind the data for visualization is not domain specific, *e.g.,* pie charts are best used when making part-to-whole comparisons. Typically, what will decide whether a visualization of a dataset is good or not depends more on its features (or representations), rather than its data values. More specifically, we consider the following features of a dataset D: the data type of a column (*e.g.,* categorical, numerical, and temporal), the number of distinct values of a column, the number of tuples in a column, the ratio of unique values in a column, the max() and min() values of a column, and statistical correlation between two columns (*e.g.,* linear, polynomial, and log).

We have crawled many visualizations as training data, and we have tested decision trees, Bayes classifier, and SVM for visualization recognition. Our empirical result shows that decision trees perform well in practice (see [5] for more details.).

Rule-based pruning. The above two-step method is time-consuming and cannot meet the real-time search requirement of users, because it will first generate many bad visualizations and then remove them. To address this issue, we propose a rule-based pruning method. The basic idea is that we first learn some visualization decision rules and then embed the rules into the text-to-visualizations method to directly prune bad visualizations.

For example, there are observations about good/bad visualizations, such as pie charts are best to use when comparing parts of a whole, and they do not show changes over time; and bar charts are used to compare things between different groups or to track changes over time, and too many bars (*e.g.,* > 50) are hard for human to interpret. The traditional wisdom from visualization experts can be encoded into rules to prune many apparently bad charts (see *e.g.,* https://www.pinterest.com/pin/20125529565819990/ for a chart type cheat sheet that can be easily leveraged). Based on these rules, we can generate some visualization templates, where each query template represents a good visualization query but without the where clause. Then for each text query, we find the good visualization templates matching the keywords. Then if the text query contains some filtering condition, we add the where clause into the query. In this way, we can directly generate good visualizations $\mathbf{V}'$.

Visualization-to-Text. The visualizations may be hard to understand if the user is not familiar with the data. To address this issue, we can generate the natural language explanation of each visualization to help users better understand the visualization. We use a rule-based method to translate each visualization query into a natural language, *e.g.,* "This chart shows the trend of (*chart type*) average (*aggregate function*) departure delay (*y-axis*) during Jan 1 (*x-axis*). The minimum delay occurs at around 0 a.m., the maximum delay occurs at about 7 p.m, and flight delays occur mostly in the peak-hour (from 8 a.m. to 8 p.m.)". Note that it is rather challenging to automatically detect "meaningful" results from visualizations, because for different chart types we have different expectations (*e.g.,* the trend for the line chart and the max/min value for bar charts) and we need to learn the features from the crawled data.

Visualization Ranking. Given a set of visualizations $\mathbf{V}'$, the *Visualization Ranking* module will rank $\mathbf{V}'$ and return top ones to the user. We use the learning-to-rank [3] techniques to learn the features from known good visualizations, which is an ML technique for training the model in a ranking task, which has been widely employed in Information Retrieval (IR), Natural Language Processing (NLP), and Data Mining (DM). Roughly speaking, it is a supervised learning task that takes the input as lists of feature vectors, and outputs the grades (ranks) of visualizations. The goal is to learn a function $F(\cdot)$ from the training examples, such that given two input vectors $\mathbf{x_1}$ and $\mathbf{x_2}$, it can determine which one is better, $F(\mathbf{x_1})$ or $F(\mathbf{x_2})$. As learning-to-rank is hard for users to understand the ranking, we also propose a partial-order-based method to rank the visualizations [5]. We define a partial order between visualizations based on some rules, *e.g.,* the matching quality between data and chart, line chart is usually better than bar chart when visualizing temporal data. Then we build a graph based on the partial order, where nodes are visualizations and edges are partial orders between nodes. We then rank the visualizations based on the graph.

Faceted Navigation. When a user picks a visualization V, she can further explore other visualizations by facets for finding more interesting visualizations. The *Faceted Navigation* module will discover another set of visualizations based on V, which will also be ranked and returned to the user. Different from traditional faceted navigation, where the facets can be easily defined as the attributes, the facets are hard to define for visualizations. To address this issue, we define the facets by visualization type, X attribute, Y attribute, Group/Bin, similar trend, different trend. For example, based on a visualization query, we can do a faceted navigation on Group/Bin to find others visualizations by changing Group/Bin strategies. The user can select her interested visualizations and iteratively do faceted navigation.

ACKNOWLEDGEMENT

This work was supported by the 973 Program of China (2015CB358700), NSF of China (61632016, 61472198, 61521002, 61661166012), and TAL education.

REFERENCES

[1] C. Binnig, L. D. Stefani, T. Kraska, E. Upfal, E. Zgraggen, and Z. Zhao. Toward sustainable insights, or why polygamy is bad for you. In *CIDR*, 2017.
[2] M. Bostock, V. Ogievetsky, and J. Heer. D^3 data-driven documents. *IEEE Trans. Vis. Comput. Graph.*, 17(12):2301–2309, 2011.
[3] C. J. C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. N. Hullender. Learning to rank using gradient descent. In *ICML*, pages 89–96, 2005.
[4] J. Fan, G. Li, and L. Zhou. Interactive sql query suggestion: Making databases user-friendly. In *ICDE*, pages 351–362, 2011.
[5] Y. Luo, X. Qin, N. Tang, and G. Li. DeepEye: Towards Automatic Data Visualization. In *ICDE*, 2018.
[6] X. Qin, Y. Luo, N. Tang, and G. Li. DeepEye: Visualizing Your Data by Keyword Search [Visionary Paper]. In *EDBT*, 2018.
[7] A. Satyanarayan, D. Moritz, K. Wongsuphasawat, and J. Heer. Vega-lite: A grammar of interactive graphics. *IEEE Trans. Vis. Comput. Graph.*, 23(1):341–350, 2017.
[8] T. Siddiqui, A. Kim, J. Lee, K. Karahalios, and A. G. Parameswaran. Effortless data exploration with zenvisage: An expressive and interactive visual analytics system. *PVLDB*, 10(4):457–468, 2016.
[9] M. Vartak, S. Huang, T. Siddiqui, S. Madden, and A. G. Parameswaran. Towards visualization recommendation systems. *SIGMOD Record*, 45(4):34–39, 2016.
[10] M. Vartak, S. Rahman, S. Madden, A. G. Parameswaran, and N. Polyzotis. SEEDB: efficient data-driven visualization recommendations to support visual analytics. *PVLDB*, 8(13):2182–2193, 2015.