Kanit Wongsuphasawat (Research Statement)

Data visualization is a critical tool for data science. Analysts use plots to explore and understand distributions and relationships in their data. Machine learning developers also use diagrams to understand and communicate complex model structures. Yet visualization authoring requires a lot of manual efforts and non-trivial decisions, demanding that the authors have a lot of expertise, discipline, and time in order to effectively visualize and analyze the data.

My research in human-computer interaction focuses on **augmenting visualization tools with automated design and recommendation**. By *automating repetitive parts* of authoring while *preserving user control* to guide the automation, people can leverage their domain knowledge and creativity to achieve their goals more effectively with less efforts and human errors. In my thesis, I have developed new formal languages for chart specification and recommendation [3,4], and used these languages to develop interactive systems that enable new forms of recommendation-powered visual data exploration [1,2]. I also built a tool that combines automatic layout techniques with user interaction to help developers visualize and inspect the structure of deep learning models in TensorFlow [5]. These systems have been open sourced and adopted in data science communities.

Augmenting Exploratory Data Analysis with Visualization Recommendation

Exploratory data analysis generally involves both specific question answering and open exploration. While traditional visualization tools [9,10] support a variety of charts to answer specific questions, they typically require *manual chart authoring*, involving many decisions including choosing data fields, applying data transformations, and designing visual encodings. This manual process can be tedious and non-trivial, demanding familiarity with the data domain as well as analysis and design expertise. While analysts should achieve systematic data coverage in their exploration, in practice they may overlook important insights, such as potentially confounding factors and data quality issues, or prematurely fixate on specific questions due to the lack of expertise or discipline.

To facilitate rapid and systematic data exploration, my thesis contributes the design and evaluation of interactive systems that *complement manual chart authoring with recommendation browsing* (Figure 1). Using the iterative design process, I developed and studied new *recommendation-powered graphical interfaces*. The **Voyager** visualization browser [1] enables data exploration via browsing of recommended charts. Our user study, which compared Voyager with a chart authoring tool, indicated the complementary benefits of manual authoring and recommendation browsing. Inspired by the study result, the **Voyager 2** system [2] blends manual and automated chart authoring to facilitate both question answering and exploration in a single tool. To provide *infrastructure for chart authoring and recommendation*, I have built new languages and recommender engine. The **Vega-Lite** grammar [3] provides a formal language for specifying interactive visualizations. The **CompassQL** recommender engine [2,4] uses Vega-Lite to reason about the space of chart designs. CompassQL represents a user input in the Voyager interfaces in the form of a *visualization query*, or a partial chart specification that omits some properties to be suggested by the engine. Given a query, CompassQL enumerates and ranks candidate specifications to suggest charts. Besides Voyager, Vega-Lite and CompassQL have also facilitated the development of other applications and research projects.

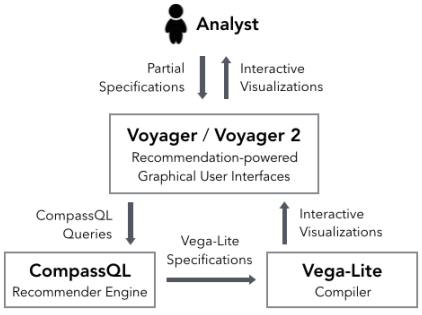

Figure 1: The Voyager and Voyager 2 recommendation-powered interfaces are enabled by the Vega-Lite grammar and CompassQL query language.

Voyager: Exploratory Analysis via Faceted Browsing of Visualization Recommendations

As manual chart authoring can impede rapid and systematic exploration, I developed Voyager [1], a system that enables interactive navigation of recommended charts for exploration data analysis (Figure 2).

To facilitate rapid and systematic exploration, Voyager embeds analysis and visualization design practices to *guide exploration while preserving user control*. To encourage analysts to thoroughly examine the data and avoid premature fixation, Voyager shows univariate summaries of all fields upon loading a new dataset. As exploration proceeds, users can focus on specific aspects of the data and steer the recommendations by selecting data fields and transformations (Figure 2, left). To promote broad exploration, Voyager prioritizes showing *data variation* (different fields and transformations) over *design variation* (different visual encodings of the same data). Besides charts showing selected fields (Figure 2, top right), Voyager also suggests charts with one extra field to help analysts consider other potentially relevant fields (Figure 2, lower right). To suggest the best visual encodings, Voyager applies empirically-derived models of perceptual effectiveness to rank visualizations.

Figure 2: Voyager facilitates systematic and rapid data exploration by presenting recommended charts based on analysis and visualization design practices for users to browse (right). To focus on specific aspects of the data and steer the recommendations, users can select data fields and transformation functions (left).

We evaluated Voyager with a user study on exploratory data analysis of previously unseen data. To provide a baseline system, we built *PoleStar*, a chart authoring tool modeled on Tableau [10]. To assess if Voyager helps users systematically explore more data, we analyze data field coverage. We found that subjects interacted with 1.5 times more unique field sets in Voyager. For user ratings (Figure 4), Voyager was preferred for open exploration as it gave users *"options that [they] wouldn't have thought about"*. However, PoleStar was preferred for question answering as users could build plots specific for their questions. Users also desired to *"start exploration with Voyager and switch to PoleStar to dive into questions"*. This result indicated the value of chart recommendation for open exploration, but also called for a unified tool that support both manual authoring and recommendation browsing.

Voyager 2: Blending Manual and Automated Chart Authoring

Motivated by the complementary value of manual chart authoring and recommendation browsing shown in the Voyager study, I designed **Voyager 2**, a tool that blends manual and automated chart authoring to facilitate *both* open exploration and question answering in one tool.

With Voyager 2 (Figure 3), users can *pivot among multiple interaction methods* within one system. As in traditional visualization tools like PoleStar and Tableau [11], users can *manually create arbitrary views* (Figure 3, top right). In addition, Voyager 2 presents *two new partial view specification interfaces*. **Related views** suggests charts based on the current specified view, allowing users to browse charts with *relevant data fields or alternative ways to summarize or encode the data* (Figure 3, lower right). The **wildcard** interfaces enable analysts to *specify multiple charts in parallel* by varying chart properties, giving them control over sets of views aligned with their analysis goals. To transition from browsing to follow-on analysis, users can make any suggested views the new specified view, and modify the view or browse other relevant views.

We compared Voyager 2 with PoleStar. Akin to Voyager, Voyager 2 helped users systematically explore more data (2.4 more unique field sets interacted with), and received higher ratings for open-exploration tasks (Figure 4) compared to PoleStar. For question answering, while Voyager was less preferred than PoleStar, subjects rated Voyager 2 comparably to PoleStar. Despite having more features than PoleStar, users remarked that Voyager 2 was *"easier to use"* and *"more learner-friendly"*. Comparing across studies, Voyager 2 improved upon both Voyager and PoleStar in terms of supporting *both* tasks in exploratory analysis including open exploration and question answering.

Vega-Lite: a Grammar of Interactive Visualizations

To provide a formal language to represent visualizations in the Voyager systems, I developed Vega-Lite [1,3], a high-level visualization grammar on top of Vega [6]. As a grammar, Vega-Lite provides primitive building blocks for composing a variety of charts. A *single-view specification* in Vega-Lite (Figure 5), which defines a Cartesian plot, describes `data` sources, `mark`, and `encoding` mappings from (optionally transformed) data `fields` to visual channels such as `x`, `y`, `color`, or `shape`. Inspired by prior visualization grammars for analysis including ggplot2 [9] and Tableau's VizQL [10], Vega-Lite provides a concise syntax by automatically generating low-level chart components such as scales, axes, and legends. To customize the plots, users can override default properties of these generated components. With a concise JSON syntax, Vega-Lite enables both rapid manual authoring and programmatic generation of charts.

Figure 3 Voyager 2 blends manual and automated chart authoring. Users can use the shelf interfaces (left) to specify arbitrary views (top right). From the specified view, Voyager 2 presents related views, allow users to browse and discover relevant data fields and alternative ways to summarize or encode the data (lower right). Users can also use the *wildcard* interfaces (in teal) to specify multiple charts in parallel by varying certain chart properties.

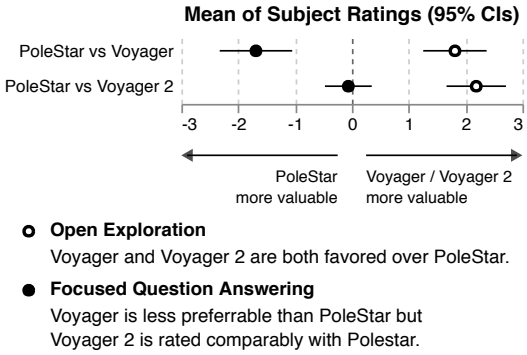

Figure 4: Task-based subject preference from studies comparing Voyager and Voyager 2 with the PoleStar specification tool. Voyager 2 is *overall* favored over Voyager and PoleStar for supporting *both* open exploration and focused question answering.

```
{
  "data": {"url": "flights.json"},
  "mark": "bar",
  "encoding": {
    "x": {
      "bin": true,
      "field": "distance",
      "type": "quantitative"
    },
    "y": {
      "aggregate": "count",
      "type": "quantitative"
    }
  }
}
```

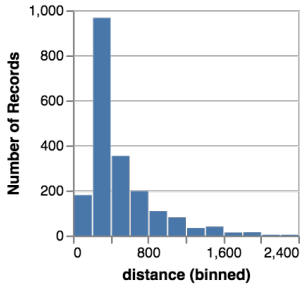

Figure 5: A histogram in Vega-Lite: bars that map a binned field and aggregated count to their position and length.

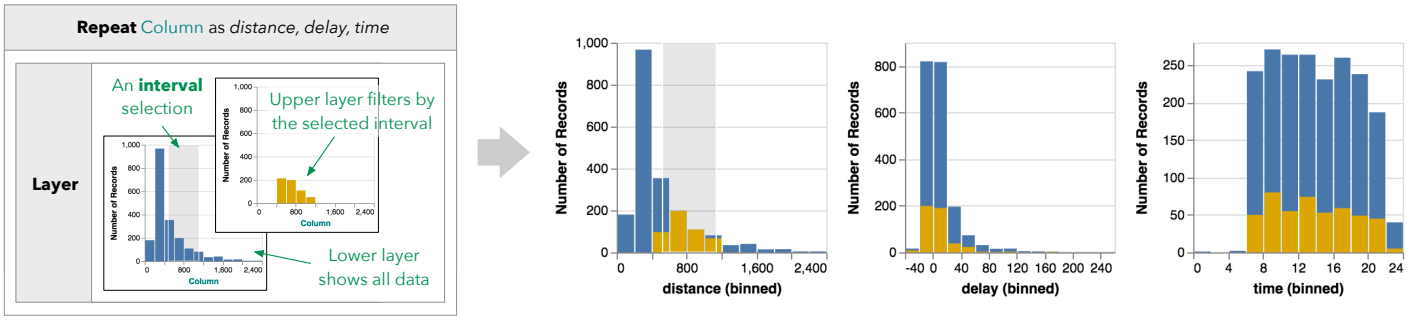

Figure 6: Cross-filtering histograms can be expressed as a column-based repetition of layered histograms. For each subplot, the upper layer (gold) shows the filtered data while the lower layer (blue) shows the original data with an interval selection on x-axis.

Beyond existing grammars, we have extended Vega-Lite to support *interactive, multi-view graphics* [3]. A novel *view algebra* enables hierarchical composition of layered and multi-view plots via operators including *facet*, *layer*, *concatenation*, and *repeat*. Interactions can be defined by specifying and applying *selections*, an abstraction that defines input event processing, points of interest, and a predicate function for inclusion testing. With these building blocks, Vega-Lite enables rapid authoring of interactive multi-view plots. For example, the cross-filtered histograms (Figure 6) can be defined within a few dozen lines of JSON in an accretive fashion, compared to several hundred lines of code in other libraries like Vega and D3 [6, 17].

Besides supporting Voyager, Vega-Lite has served as *a platform for developing other applications and research projects* (Figure 7). For research, we built PoleStar as a baseline system for our user studies using Vega-Lite. We also used Vega-Lite to develop an automatic model to reason about visualization similarity and sequencing [7]; our colleagues at Stanford [14], Georgia Tech, and Princeton are using Vega-Lite to build natural language interfaces for data visualization and analysis. As a declarative format, Vega-Lite also enables *sharing across applications and platforms.* JupyterLab, the latest version of the Jupyter/iPython notebook, supports Vega-Lite and Vega as its official plotting formats. In an ongoing work with the Jupyter team, we are integrating Voyager 2 as a JupyterLab extension, so users can easily share and re-use Vega-Lite outputs from Voyager 2 in JupyterLab. Vega-Lite is also wrapped as native visualization libraries in many languages. For example, our colleagues has built a Python wrapper called *Altair [12]* and noted that *"Vega-lite (and Vega) are perhaps the best existing candidate for a principled lingua franca of visualization"*. With an easy-to-use yet flexible design, Vega-Lite is also used for *teaching* in book [16] and classes at leading institutions including Stanford, CMU, and the Universities of Maryland and Washington. Vega-Lite has over **1,000 GitHub stars** and **30,000 downloads per month** on NPM.

CompassQL: Visualization Recommender Engine & Query Language

To support chart recommendation in Voyager, I developed the *CompassQL* recommender engine [2,4]. CompassQL represents a user input in the Voyager interfaces as *a visualization query*, or a partial chart specification that defines an organized collection of charts. A specification in a CompassQL query (Figure 9) has a structure like a single-view specification in Vega-Lite, but can contain **wildcards** to indicate properties that are omitted to be determined by the engine. To organize outputs, a query may contain **recommendation directives** for grouping redundant plots (such as plots with similar data fields and transformations) and choosing or ordering plots (e.g., by a perceptual effectiveness ranking of visual encodings).

Given a query, CompassQL enumerates candidate Vega-Lite specifications by replacing each wildcard with concrete values. Each candidate must satisfy both user-defined constraints in the query and the engine's built-in constraints, which applies visualization design knowledge to prune misleading charts. The engine then groups and ranks qualified charts based on the recommendation directives.

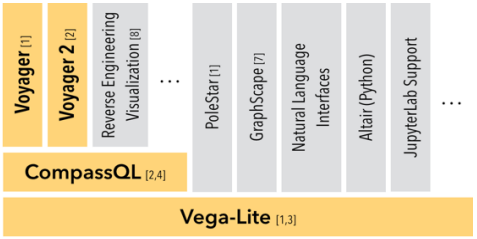

Figure 7. Beyond Voyager, Vega-Lite and CompassQL have served as a platform for developing other applications and research

```
{
  "spec": {
    "data": {"url": "cars.json"},
    "mark": "?",
    "encodings": [{
      "channel": "x",
      "field": "Horsepower",
      "type": "quantitative"
    },{
      "channel": "y",
      "field": "MilesPerGallon",
      "type": "quantitative"
    }]
  },
  "chooseBy": "effectiveness"
}
```

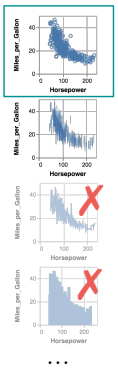

Figure 9: A CompassQL query for the shelf interface in Figure 3. The "auto" mark in the shelves produces a wildcard (?) to enumerate marks. The CompassQL engine applies built-in constraints to prune misleading marks and chooses the best mark based on a perceptual effectiveness ranking.

```
{
  "spec": {
    "data": {"url": "cars.json"},
    "mark": "?",
    "encodings": [{
      "channel": "x",
      "fn": ["bin", "mean"],
      "field": "Horsepower",
      "type": "quantitative"
    },{
      "channel": "y",
      "fn": ["bin", "mean"],
      "field": "MilesPerGallon",
      "type": "quantitative"
    }]
  },
  "groupBy": "transformedFields",
  "chooseBy": "effectiveness",
  "orderBy": "functionOrder"
}
```

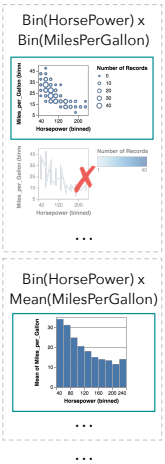

Figure 8: To suggest related summary views in Figure 3, Voyager 2 augments the query in the specified view (Figure 9) with constrained wildcard functions (fn) and groups charts with similar data fields and functions.

The Voyager systems use the CompassQL query language to *represent both specification and recommendation*. To generate a query, Voyager maps selected fields to wildcard channels and marks. Voyager 2 maps all wildcards in the interface directly to wildcards (?) in the queries (Figure 9). To suggest related views, Voyager 2 augments the specified view with additional wildcards (Figure 8). To organize recommendations, both systems group charts with the same data and transformations by default to show data variations. They then apply the perceptual effectiveness ranking to choose the top visual encoding for each group.

Beyond supporting Voyager, I also demonstrated that CompassQL can express a variety of existing recommendation approaches [4], including techniques used within Tableau [11] and recent work from the database community [15]. CompassQL was also used to generate training data for an analysis pipeline that reverse-engineers visual encodings from bitmap chart images [8].

Visualizing Dataflow Graphs of Deep Learning Models in TensorFlow

Besides data exploration, automatic visualization design can assist other data science activities such as machine learning model authoring. To help developers understand deep learning architecture, I led the development of TensorFlow graph visualizer [5], a tool that combines automatic layout techniques with user interaction to visualize complex dataflow graphs of TensorFlow models.

To simplify creation and deployment of deep learning models, Google's TensorFlow library generates low-level dataflow graphs to represent computations in the models. As developers often draw high-level diagrams to build a mental map and communicate their model structures, they desire a way to automatically generate diagrams from their code. However, these graphs are low-level and typically contain thousands of nodes. Some of these nodes also have high degree but are unimportant for understanding model structure. As a result, standard graph layout tools produce tangled diagrams (Figure 10, top).

Combining automatic layout techniques and interactions that give users control, the visualizer enables developers to create legible interactive diagrams that match their mental maps. The tool applies hierarchical graph clustering to build a high-level diagram of the model (Figure 10-11), akin to what developers typically draw, and enables users to explore its nested structure by expanding clusters. To help users create graph clusters that match their mental maps, our strategy is to let users annotate the source code with hierarchical information. To declutter the layout, the tool applies heuristics to extract nodes that developers normally omit from their hand-drawn diagrams and allows users to customize the layout by extracting and un-extracting more nodes.

The graph visualizer has been released as a part of TensorFlow and widely used in the community for debugging and sharing their deep learning model structures. Online reviews of deep learning libraries mentioned that the visualization "*helps differentiate TensorFlow from other libraries*" and *"is a great step in the right direction"*. Screenshots from the visualizer regularly appear on the official and 3rd-party tutorials as well as on StackOverFlow questions for explaining models. From internal mailing lists at Google and external blogs, we have found that many users repeatedly modified their annotations in the code to make the diagrams match their mental map, indicating that the visualizer is indeed valuable for model developers.

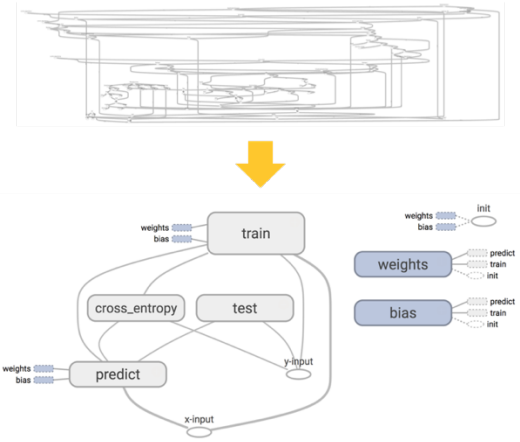

Figure 10. TensorFlow Graph Visualizer converts the low-level computation graph of a linear model (TensorFlow's "Hello World" example) into a high-level interactive diagrams.

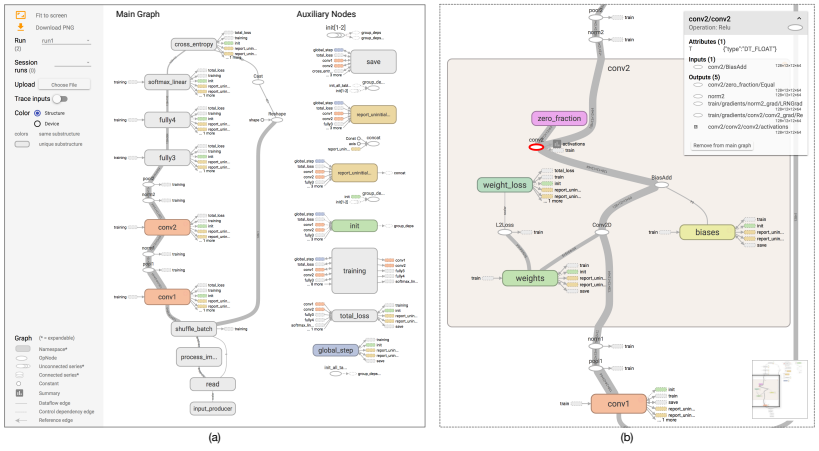

Figure 11. A diagram of a convolutional network for classifying images. (Left) An overview shows a dataflow between operations that are grouped based on user annotation. Less important nodes are extracted to the side. (Right) Expanding a group shows its nested structure.

Future Research

I plan to continue applying my expertise in human-computer interaction to study and design tools that facilitate people, both experts and novices, to work with and benefit more from data visualization and artificial intelligence.

More Expressive Chart Recommendation & New Applications. My thesis has developed novel systems for chart specification and recommendation, and used them to facilitate data exploration. However, this work is only an initial investigation of visualization recommendation research. While CompassQL currently only suggests single-view plots, multi-view composition and interaction

support in Vega-Lite can enable a richer set of recommendations. Layered charts such as box plots can provide more statistical information like variability and outliers in addition to the central tendency of a distribution. Suggesting interaction techniques for different types of plots can also facilitate interactive analysis. Besides data exploration, automated visualization design can assist other tasks. Suggesting annotations to indicate trends may enhance visual communication. Warning about misleading plots and proposing alternatives with explanations for why certain designs are better can educate novices and improve visualization literacy.

To support a richer set of recommendations, we need to study and extend enumeration strategies, design constraints, and rankings in CompassQL beyond single-view plots. For example, suggesting multi-view graphics requires enumeration of composition structures (e.g., by layering or concatenating views) in addition to enumerating wildcards in a fixed specification structure. Design constraints such as enforcing consistencies of visual encodings between sub-views [13] can facilitate reading of multi-view graphics. Moreover, manually designing features for the perceptual effectiveness ranking may not scale for this large combinatorial design space, I plan to explore probabilistic models that learn and improve the rankings based on user interactions. Such work will also open up possibilities for personalization and adapting recommendation for different data domains.

Machine Learning Development Tools. Helping developers build and understand models is vital to the success of machine learning systems. Visualization can be a critical tool for this mission. The dataflow graph visualization in TensorFlow has eased inspection and understanding of deep learning architecture. However, beyond just visualizing the models from their source code, the visualization could support direct editing and exploration of the models. Moreover, other visualization techniques could facilitate model analysis. To provide better support for model exploration and analysis, many important questions remain. For example, what are variations of the models that developers expect to vary in their experiments? How might we design interfaces to help people better explore and evaluate these variations? Could we learn common model composition patterns and suggest appropriate building blocks for developers? How could we apply visualization techniques to help developers analyze and understand their model performance?

Interaction Design for Intelligent Systems. I am also interested in the general design issues for intelligent systems, namely how to help end-users benefit from automation while preserving their control and creativity. I believe that insights from my work on visualization tools are applicable for other domains. One important design consideration is to find the right balance between automation and user control. For trivial decisions such as generating low-level chart components in Vega-Lite, simply automating the process while providing a way to override or undo can be sufficient. For more complex activities like data exploration, it might be more appropriate to suggest multiple options for users to browse and give users fine-grained controls over the suggestions like in Voyager 2. Partial specifications of domain-specific representations like CompassQL queries can facilitate such fine-grained control over the suggestions. A related consideration is to identify parts of the work that need user input. Without user annotation, layout techniques in the TensorFlow Graph Visualizer would not produce diagrams that match users' mental maps. Despite generally positive results for Voyager 2, a study participant remarked that *"[the related views] are so spoiling that I start thinking less."* Going forward, the bias and complacency due to automation is another important issue. Besides visualization, I am keen to apply these considerations to design systems in other domains. By studying and designing systems in diverse domains, I hope to generalize observed design patterns and develop guidelines for building systems that empower people with machine intelligence.

References

1. Voyager: Exploratory Analysis via Faceted Browsing of Visualization Recommendations. *Kanit Wongsuphasawat, Dominik Moritz, Anushka Anand, Jock Mackinlay, Bill Howe, Jeffrey Heer.* IEEE TVCG (InfoVis) 2015. **Invited to SIGGRAPH 2016 as 1 of 4 top TVCG papers**.
2. Voyager 2: Augmenting Visual Analysis with Partial View Specifications. *Kanit Wongsuphasawat, Zening Qu, Dominik Moritz, Riley Chang, Felix Ouk, Anushka Anand, Jock Mackinlay, Bill Howe, Jeffrey Heer.* ACM SIGCHI 2017.
3. Vega-Lite: a Grammar of Interactive Graphics. *Arvind Satyanarayan, Dominik Moritz, Kanit Wongsuphasawat, Jeffrey Heer.* IEEE TVCG (InfoVis) 2016. **Best Paper Award.**
4. Towards A General-Purpose Query Language for Visualization Recommendation. *Kanit Wongsuphasawat, Dominik Moritz, Anushka Anand, Jock Mackinlay, Bill Howe, Jeffrey Heer.* ACM SIGMOD Human-in-the-Loop Data Analysis (HILDA) 2016
5. Visualizing Dataflow Graphs of Deep Learning Models in TensorFlow. *Kanit Wongsuphasawat, Daniel Smilkov, James Wexler, Jimbo Wilson, Dandelion Mané, Doug Fritz, Fernanda Viégas, Martin Wattenberg.* IEEE TVCG (VAST) 2017. **Best Paper Award.**
6. Declarative Interaction Design for Data Visualization. *Arvind Satyanarayan, Kanit Wongsuphasawat, Jeffrey Heer.* ACM UIST 2014
7. GraphScape: A Model for Automated Reasoning about Visualization Similarity and Sequencing. *Younghoon Kim, Kanit Wongsuphasawat, Jessica Hullman, Jeffrey Heer.* ACM SIGCHI 2017. **Best Paper Honorable Mention.**
8. Reverse-Engineering Visualizations: Recovering Visual Encodings from Chart Images. *Jorge Poco, Jeffrey Heer.* EuroVis 2017
9. Ggplot2: Elegant Graphics for Data Analysis. *Hadley Wickham.* Springer 2009.
10. Polaris: A System for Query, Analysis, and Visualization of Multidimensional Relational Databases. *Chris Stolte, et al.. 2002.* IEEE TVCG (InfoVis) 2002.
11. Show me: Automatic presentation for visual analysis. *Jock Mackinlay, et al.* IEEE TVCG (Proc. Infovis) 2007
12. Altair: Declarative statistical visualization library for Python. https://github.com/altair-viz/altair
13. Keeping Multiple Views Consistent: Constraints, Validations, and Exceptions in Visualization Authoring. *Zening Qu et al.* IEEE TVCG (InfoVis) 2017
14. A Conversational Agent for Data Science. *Ethan Fast.* https://hackernoon.com/4ae300cdc220
15. SeeDB: Efficient Data-Driven Visualization Recommendations to Support Visual Analytics. *Manasi Vartak, et al.* VLDB 2016
16. Making Data Visual: A Practical Guide to Using Visualization for Insight. *Miriah Meyer, Danyel Fisher.* O'Reilly Media 2018
17. D^3 data-driven documents. *Mike Bostock, Vadim Ogievetsky, Jeffrey Heer.* IEEE TVCG (Infovis) 2011