Northstar: An Interactive Data Science System

Tim Kraska
Massachusetts Institute of Technology
kraska@mit.edu

ABSTRACT

In order to democratize data science, we need to fundamentally rethink the current analytics stack, from the user interface to the "guts." Most importantly, enabling a broader range of users to unfold the potential of (their) data requires a change in the interface and the "protection" we offer them. On the one hand, visual interfaces for data science have to be intuitive, easy, and interactive to reach users without a strong background in computer science or statistics. On the other hand, we need to protect users from making false discoveries. Furthermore, it requires that technically involved (and often boring) tasks have to be automatically done by the system so that the user can focus on contributing their domain expertise to the problem. In this paper, we present Northstar, the Interactive Data Science System, which we have developed over the last 4 years to explore designs that make advanced analytics and model building more accessible.

PVLDB Reference Format:
Tim Kraska. Northstar: An Interactive Data Science System. *PVLDB*, 11 (12): 2150-2164, 2018.
DOI: https://doi.org/10.14778/3229863.3240493

1. INTRODUCTION

To truly democratize Data Science, we need to fundamentally change the way people interact with data. Astonishingly, the interfaces people use to analyze data have not changed since the 1990s, and most analytical tasks are still performed using scripting languages and/or SQL. Of course, there have been fashion trends in the choice of programming language (e.g., from PERL to Python), algorithms (e.g., from neural nets to statistical learning and back to neural nets), and database technology (SQL to NoSQL to Not Only SQL). Yet, people still interact with data primarily through writing scripts and SQL-like languages, with up to hour-long wait times for results.

Proceedings of the VLDB Endowment, Vol. 11, No. 12
Copyright 2018 VLDB Endowment 2150-8097/18/8.
DOI: https://doi.org/10.14778/3229863.3240493

We argue that we should stop holding onto the past; rather, we should start designing systems for how Data Science **should be done** 10 years from now. With Northstar[1], a system for interactive Data Science, we've tried to do exactly that for the last four years. Perhaps surprisingly, some aspects of our vision for the system have been inspired by movies such as "Minority Report" and the newer James Bond films. All of these movies feature highly collaborative visual environments with touch (and pen) interfaces for analyzing data; you see nobody coding in Python. With Northstar, we had a very similar goal: to provide a highly collaborative visual Data Science environment based on a touch and pen interface, and simplify its use so much that domain experts who are not trained in statistics or computer science can use it without any help.

At the same time, we wanted to use hardware that is already available and not wait until holograms actually become reality (though, we were really excited about HoloLens [76]). We therefore settled on interactive whiteboards – specifically, the Microsoft Surface Hub – as our core target platform. Interactive whiteboards are essentially large multi-touch TVs, but often with highly reduced lag time and better resolution, which provide a real alternative to whiteboards. Consequently, we were not shocked that Microsoft struggled to fulfill the demand for the Surface Hub [111], nor did it surprise us that other tech companies, such as Google, followed suit and now also offer their own interactive whiteboard solutions.

So far, interactive whiteboards are just better conferencing systems, but they have the potential to be much more. We want to put them at the center of every meeting that involves numbers, from discussing sales figures to better understanding the customer base, and even to building predictive models. We envision a collaborative environment where domain experts and data scientists can work together to arrive at initial solutions during a single meeting – solutions which can then, if necessary, be refined offline. This is in stark contrast to the current dreadful way that data scientists and domain experts interact: meetings after meetings to find a common base before real progress is first made. Consequently, to foster collaboration and results during a meeting, the system has to provide a visual interface, because co-programming Python with a CEO is simply not an option. Furthermore, we want to enable domain

[1]Previously, it was named Interactive Data Exploration System (IDES), but this name no longer seemed adequate since we added significant support for model building.

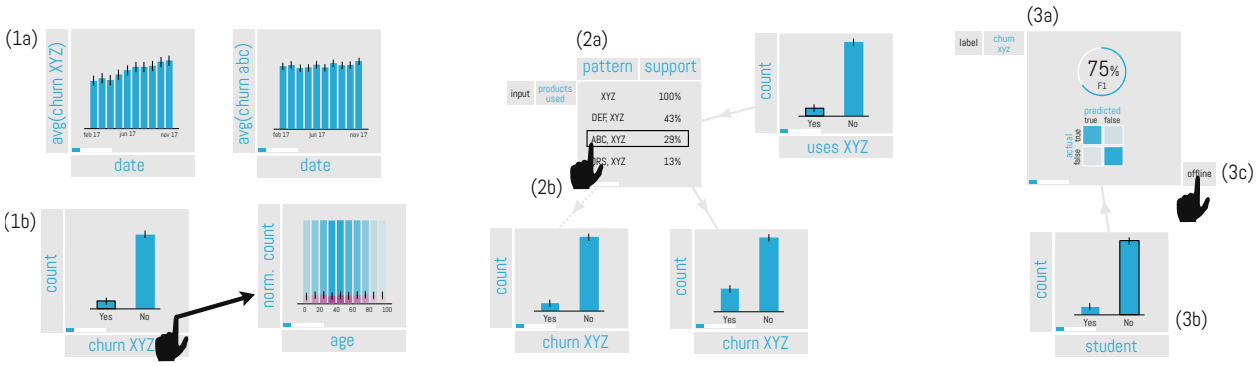

Figure 1: Illustration of Vizdom accompanying the use case described in Section 2. (1a) Example of progressive visualizations. (1b) Example of a brushing operation initiated by putting two visualizations close to each other. (2a) Example of a frequent itemset operator that is filtered by a selection in the visualization in the top right area. (2b) Operators can also serve as filters for downstream visualizations. Dashed connection lines represent a NOT operator. The bottom left visualization is filtered to everything BUT the item selected in the operator. (3a) Example of a prediction operator that uses Northstar's automated ML capabilities. (3b) Visualizations can be used as filters to subset the data a prediction operator works on. (3c) Users can choose to transfer long-running prediction operators to an offline mode if more precision is desired.

experts to build models on their own without the help of a data scientist or within the meeting room setting. Thus, the user experience on a domain expert's laptop should be similar to that on an interactive whiteboard and feature a virtual data scientist, who watches over the process and prevents any major mistakes. Interestingly, by putting the user experience first, we not only found that existing systems do not work in this setup, but we also ended up designing a system very different from one we would have created using a systems-first approach. Today, we already have several pilot deployments of Northstar in industry and academia, among them Adobe and IGT.

The main goal of this system paper is to provide an overview of Northstar and explain the rationale behind its design, as well as outline interesting challenges, solutions, and future work for designing the next generation of Data Science systems, with the goal of eventually truly democratizing Data Science.

The remainder of this paper is organized as follows: After a motivating use case (Section 2), we provide an overview of Northstar (Section 3), and afterward discuss the different components of Northstar in more detail (Section 4 - 8). Finally, we discuss related work (Section 9) and future directions (Section 10), before concluding the paper (Section 11 & 12).

2. A MOTIVATING USE CASE

To motivate Northstar and illustrate its power, we present an introductory use case of how we envision Northstar being used. Throughout this description, we refer to parts of Figure 1.

Pete, a product manager, and Dana, a data scientist, both work at a large software company that offers various productivity tools through a subscription-based model. The company has recently released a new product, XYZ, and Pete has noticed that it's not meeting their expectations. More specifically, it seems like the churn rate (i.e., the numbers of customers who stop their subscription) for XYZ has been increasing steadily.

Pete asks Dana to meet with him to investigate the issue collaboratively and explore how customers are behaving. They start up Vizdom, the front end of Northstar, on an interactive whiteboard (see also Figure 2) in one of the meeting rooms and start out with a few data exploration queries. (1a) For example, they plot the churn rates of other products and find, that the overall trend for XYZ indeed looks worse. (1b) Using Vizdom's brush and normalization features, they follow up by looking at XYZ's user demographics to see if particular subpopulations (different age groups, in the example) of users are more likely to stop their subscriptions. They conclude that no particular group of users shows any interesting trends. All of these visualizations are computed and refined progressively. Despite their large data set, Dana and Pete see near-instantaneous results for all their interactions, allowing them to explore many different questions in a short amount of time and without losing focus on their current train of thought.

Dana suggests investigating interactions between different tools. (2a) She brings up a frequent itemset operator in Vizdom, inputs "products used," and filters it to include only users who have used XYZ; they observe that users commonly use XYZ in combination with other company tools. (2b) Pete has a hunch and selects all the users using XYZ together with ABC. He has long suggested that the overlap in functionality between the two tools is fairly significant and that they might compete with each other. Looking at the churn rate for this subpopulation and comparing it to the overall population of users who use XYZ, they see that there is a noticeable difference: users who use both tools are more likely to end their subscription than users who do not. Vizdom's filter functionality supports the full power of boolean queries. In the example, Pete uses a gesture to produce a NOT operator, represented by the dashed connection line. Pete writes a note: he wants to follow up with his team and discuss ideas for making the two tools more distinct.

After exploring the data, our two protagonists decide to be more proactive and install measures to reduce the number of future subscription cancellations. In the past, they had good success with sending coupons to customers. They think adding a new customer relation workflow that automatically sends this coupons to users they are about to loose could help.

(3a) To check how feasible this is, they use Vizdom to build a model that predicts, given all available data, if a particular user is likely to cancel their subscription in the future. The system goes off and initiates an automatic model search that validates its results by splitting the data into various training and testing folds. Since this operation is again done progressively, Pete and Dana can see after just a few seconds that the system found a model that does fairly well on this task (i.e., F1-Score = 75%). Pete is not an expert in machine learning, so Dana toggles the view to a confusion matrix that shows the detailed performance of the best model the system has found so far. Pete wants some more details, so they inspect the model in more depth. Among other things, they can inspect different training testing splits, see the predictions for different user sub-populations, or look at attribute-based decision boundaries. In doing so, Pete notices that one of the top decision criteria is whether a customer is a student. That seems fishy to Pete. Given his domain background and experience, he knows there is high fluctuation among students. They often cancel their subscriptions at the end of a semester then sign up again at the beginning of the next. (3b) He asks Dana to exclude this population of users from their model. The system restarts its model search and Pete and Dana see that the overall performance is a bit worse now, but Pete thinks it's better to exclude those users to avoid sending unnecessary promotions.

They decide that the workflow is reasonable and plan to deploy it into the production environment. (3c) Dana puts the ongoing model search into offline mode. The system will keep searching for and improving models that solve this problem and will notify Dana of the results after a specified time. At that point, Dana can export the best model the system found in Python and hand it off to one of her team members to set up this new customer-relations workflow with some A/B testing. After a few weeks, Dana and Pete reconvene and use Vizdom again to analyze if this prediction model worked well and if sending these promotions affected the churn rate.

3. Northstar OVERVIEW

Putting the targeted user experience first, as sketched out in the previous section, led us to a very different system design than a systems-first approach would have. In this section, we first discuss some of the key realizations that influenced the system design before presenting the overall system architecture.

3.1 Key Requirements for Designing an Interactive Data Science System

Over the course of the project, we conducted several user studies [113, 115, 114], which guided our system design to address the aforementioned use case. While some of the realizations appear to be trivial in hindsight, they were not when we started. In the following, we outline some of our key findings during the course of the project.

(1) Results have to be approximate: Results for any operation, including machine learning, have to return in seconds, if not milliseconds. Anything else disturbs the user's experience, leading to fewer discoveries [69, 113] and causing domain experts to walk away, saying, "better to do that offline." Unsurprisingly, that implies that results have to be approximated, as it is otherwise often impossible to return answers fast enough.

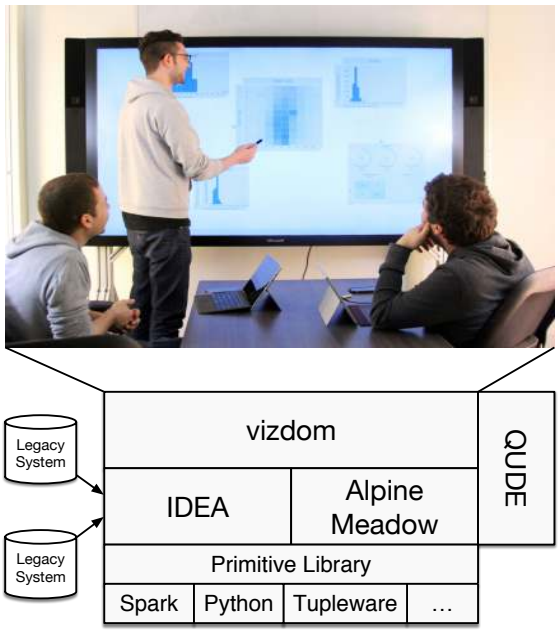

Figure 2: Northstar architecture overview

(2) Outliers matter: While a lot of work in approximate query processing disregards the importance of outliers, we found them extremely important. Users have a tendency to explore anything that stands out – the most valuable customers, records with data errors, the top-k sold products, etc.

(3) Progressive results are often better than error bars: While the first version of our interface featured error bars, we now support them only as an option for advanced users. As other studies have already shown [23], error bars are often misinterpreted by users or simply ignored in the first place. We found that progressive results, which are continuously updated in the background until they converge on the complete answer, provide the user with a much better experience. Most importantly, we found that fluctuations in the visualization help the user to better understand how reliable the approximation is, and the guarantee that the result will eventually converge provides additional confidence.

(4) Connect & explore over legacy systems: Companies often have a large landscape of legacy systems, from various database installations (Oracle, DB2, Postgres, etc.) to distributed filesystems (Hadoop, EMC, etc.), which hold most of their data and are extremely hard to change. Therefore, a key requirement is that a new analytics system has to seamlessly integrate with the existing infrastructure and that a user can simply connect to a data source and start exploring without any time-consuming pre-processing steps.

These requirements influenced one of the most important design decisions for Northstar; instead of creating a new database system which replaces existing DWHs (e.g., as proposed in [106, 107]) to better support Interactive Data Science (IDS), we decided to design an accelerator, which sits on top of DWHs, filesystems, etc., and essentially functions as a giant intelligent cache for the data source.

(5) Visualizations are not like SQL queries : The workload created by visualization systems is very different from what TPC-H and TPC-DS make us believe analytical workloads look like. A single visualization often requires more than one query, and even simple visualization are often extremely hard to express as SQL queries [35]. For example, histograms require binning the data into buckets and then performing an aggregation per bucket, which is surprisingly complicated to express in SQL and not well supported by existing DBMSs.

(6) Visualization can hide information: For months, we demonstrated Northstar's unsupervised learning feature using MIMIC II [77], a data set about critical care patients. Using frequent itemset clustering, we showed that we can quickly find a large difference between the common diseases of young vs. old people. However, just by chance, one of us inspected the data more closely and found that the age 0 was used for patients whose age was unknown. When those records were removed, the difference in the clustering wasn't that pronounced anymore, indicating that patients with missing patient information also have different diseases. This simple discovery led us down a rabbit hole of research topics to investigate techniques that can automatically point out potential problems in an analysis, from the Simpson Paradox to Multiple-Hypothesis Error.

(7) Taking a holistic view of Data Science is important: Users frequently want to switch between looking at data (exploring it), transforming and analyzing it (building models, running stats). For example, with the MIMIC 2 data set, we wanted to test if patients with a fever also have a higher heart rate. We therefore quickly created a statistical test in our interface, but then noticed that the temperature of some patients was reported in Celcius and not Fahrenheit. Thus, a user should be able to quickly write a UDF to convert Celsius to Fahrenheit and rerun the whole analysis with the corrected input without ever switching to different tools.

3.2 Overall System Architecture

We designed Northstar specifically to address the IDS requirements, and also kept redesigning it as we came across new ones. For example, we first designed Vizdom with the goal of taking advantage of our existing analytics framework, Tupleware [24, 25]. Tupleware is in many aspects similar to Spark, but was designed to run on small high-performance clusters and was able to achieve one to three orders of magnitude better performance. However, while our initial belief was that a fast execution engine is key to achieving interactivity, it turned out that query approximation and progressive results and the ability to quickly change the analytical workflow based on user interaction are far more important and often contradict the goal of code compilation. Interestingly, the same observation was recently made by the HyperDB team. Because of its excellent query execution performance, HyperDB was sold to Tableau and now is primarily used as a backend for Tableau's visual front end. However, in a recent publication [61], the authors primarily describe how to avoid code generation for the often short running queries generated by the visual front end. In contrast, we decided to build IDEA, the interactive data exploration accelerator, and now only use Tupleware to pre-compile complex operations. Furthermore, during the turn of the project, Python became increasingly popular as the main language of choice for data scientist. Thus, it became pretty clear that we can not rely on a single execution engine, but rather need to be compatible with other frameworks.

The resulting system architecture is shown in Figure 2. The Vizdom front end provides a visual data exploration environment specifically designed for pen and touch interfaces, such as the Microsoft Surface Hub. Figure 2 includes an actual picture of the Microsoft Surface Hub in our lab running Vizdom to explore a medical data set. A demo video of Vizdom can be found here `vimeo.com/139165014`. Currently, Vizdom connects to IDEA using a standard REST interface, which in turn connects to the data sources using the appropriate protocols (e.g., ODBC). These data sources can include anything from legacy data warehouses to raw files to advanced analytics platforms (e.g., Spark, Hadoop).

In turn, IDEA acts as an intelligent cache and streaming approximation engine that uses Tupleware [25, 24], Python, Spark or other engines as runtimes for more complex analytics tasks. To inform IDEA about which operations are available in which runtime, the primitive library provides a standardized API and metadata information about them. For the ML auto-tuning, we built Alpine Meadow, a "query" optimizer for machine learning. Finally, QUDE, the component to Quantify the Uncertainty in Data Exploration, monitors every interaction the user does and tries to warn about common mistakes and problems and, if possible, even prevents them from happening in the first place.

In the following, we describe each component in more detail and how they address the aforementioned challenges.

4. VIZDOM: AN NOVEL INTERFACE FOR DATA SCIENCE

As Fisher et al. [42] argued, a common way to perform data analytics at the turn of the 21st century was to use spreadsheet applications and data sets that would fit completely in memory. Computations were therefore fast and results were visible within seconds. Users could perform multiple analyses simultaneously, explore different aspects of the data, and iteratively and interactively refine findings at a fast pace.

Today, these conveniences are gone. Increasing data complexity that requires specialized query languages and transformations, modern analytical scenarios that rely on advanced algorithms (e.g., machine learning), and the sheer size of today's data all force users to interact with data through custom jobs written in scripting or programming languages. These jobs run for minutes or hours in the cloud, without providing insights on what goes on behind the scenes.

This mainframe-like interaction paradigm is an inherently poor fit for Data Science. The work is exploratory by nature and demands rapid iterations, and all but the simplest analysis tasks require domain experts, who often do not have programming skills, to be in the loop to effectively steer the process.

While systems like Tableau are a step in the right direction, offering a visual interface for data exploration, they lack support for creating sophisticated models. In our work to make Data Science more accessible, we saw user experience as a crucial component. We consciously designed Northstar using a top-down approach, where user needs drive the requirements for the

rest of our system. We therefore closely collaborated with Andy van Dam's group at Brown University to develop Vizdom [26], a novel pen-and-touch interface for interactive Data Science (Figure 2 shows Vizdom running on a Microsoft Surface Hub).

Vizdom exhibits a *fluid* [36] and novel interaction style that is designed to promote "flow"—staying immersed in the current activity and not being distracted by the user interface—and relies on prompt feedback and fast response times. Interactively analyzing multidimensional data sets requires frequent switching between a range of distinct but interrelated tasks (e.g., producing different visuals based on different column sets, calculating new variables, observing interactions between subsets of the data, creating statistical models, etc.). Vizdom addresses this challenge by unifying a comprehensive set of tools for visual data analysis into a hybrid pen-and-touch system designed to exploit the visualization advantages of large interactive displays. Tools are either data views, placeholders for visualizations, or operators that perform transformations or computations on data. User can interact with these elements through direct manipulation, and elements will act not only as result-viewers but also, more importantly, also as controls for adjusting or steering ongoing computations. Leveraging an unbounded whiteboard metaphor, users can combine these tools like building blocks to create interactive and visual workflows.

In designing Vizdom, we put heavy emphasis on fast responses to each and every user interaction regardless of the size of the data being analyzed. Human-computer interaction literature [80, 93] often states that a delay of one second is the upper bound for system responses, after which users lose focus on their current train of thought. To ensure a highly interactive environment for data analysis, Vizdom makes use of progressive visualizations and approximate answers as computed by IDEA (see Section 6).

We first demonstrated Vizdom at VLDB 2015, where it won the Best Demo Award. Since then, we have worked with various academic and industry partners to get Vizdom with its backend deployed and learn more about the real needs of domain experts and data scientists across various domains. We worked with one of Adobe's Data Science teams, who used Vizdom to analyze their product subscription data, and started a collaboration with IGT, among others. Similarly, we continuously use Vizdom for focused user studies to better understand user behavior. For example, we studied the impact of approximate visual results on exploratory analysis [113] and examined the effect of the multiple comparison problem in visual analysis [115].

5. TUPLEWARE: BARE METAL SPEED FOR UDFS

Todaya's analytics frameworks are ill-suited to support interactive visual frontends, even for simple operations on small data sets. Current frameworks (e.g., Hadoop, Spark) are designed to process massive data sets distributed across huge clusters, which addresses the problems faced by giant Internet companies. With these frameworks, just scheduling a single job can often take longer than any reasonable interactivity latency threshold. With Tupleware, we explored the design of a new analytical framework for interactive latencies and "normal" users—not the Googles of the world. Two key contributions of Tupleware are (1) the close-to-zero execution and scheduling overhead and (2) new query compilation and optimization techniques. The latter fundamentally bridges the gap between query optimizers, which usually make high-level optimization decision (e.g., join ordering), and compilers, which make low-level optimization decisions (e.g., loop-unrolling).

While Tupleware was very important in the early stages of Northstar, it lost its importance in the interactive Data Science stack over time. First, it turned out that even the fastest execution engine can be too slow to provide interactive response times for complex operations. Thus, we started to implement more and more algorithms as approximate and progressive alternatives directly in IDEA (see next section). Second, code compilation, as also noted by others [61], often has an up-front cost, which can quickly add up in cases where many small, short-running queries dominate the workload. Hence, it became more efficient to use Tupleware mainly to pre-compile complex analytical operations (similar to stored procedures), which are then combined into complex workflows based on the user interactions within IDEA using an iterator-based execution model. However, Tupleware was one of the very first systems to compile complex analytical workflows, and many systems built upon its results [1, 78, 83, 37]. Furthermore, we believe that Tupleware's compilation strategies might play an important role in the next generation of Northstar as part of synthesizing better access methods [63].

6. IDEA: AN INTERACTIVE DATA EXPLORATION ACCELERATOR

As mentioned in the previous section, even the fastest runtime can be too slow to guarantee interactive response times over very large data sets. Approximate query processing (AQP) techniques can help in these situations, but existing techniques fall short in providing good approximations over rare events. Yet, users commonly explore those events, as they often contain interesting insights (e.g., the habits of the few highest-valued customers, the suspicious outliers, etc.). Furthermore, existing AQP engines integrate poorly with legacy systems. We therefore started to develop the first Interactive Data Exploration Accelerator (IDEA) [27] with the goal of building the first approximation engine for interactive Data Science, which seamlessly integrates with existing IT infrastructures.

6.1 Neither a DB nor a Streaming Engine

Interestingly, IDEA required a fundamental rethinking of the query execution model; it is neither a system for one-shot queries, nor traditional AQP engine, nor a streaming engine. Rather, IDEA has entirely unique semantics. Fundamentally, IDEA is a database as a middle-tier and acts as an intelligent, in-memory caching layer that sits in front of the much slower data sources, managing both progressive results and the samples used to compute them.

But even over cached samples, the query execution model is different. Unlike DBMSs, queries are not one-shot operations that return batch results; rather, workflows are constructed incrementally, requiring fast response times and progressive results that refine over time. It is also not a traditional AQP engine, as users often incrementally compose operations into

complex workflows with one output being the input for one or more other operations, like in a streaming system. Yet, in contrast to streaming engines, IDEA is meant to enable free-form exploration of data sampled from a deterministic system (e.g., a finite data source), whereas traditional streaming engines typically assume a predefined set of queries over infinite data streams and do not have samples as inputs.

Also in contrast to a traditional DBMS, IDEA can offload prefiltering and pre-aggregation operations to an underlying data source (e.g., perform a predicate pushdown to a DBMS) or even transform the base data by executing custom UDFs in the source. Most importantly, though, in contrast to traditional DBMSs, AQP engines, or streaming engines, IDEA users compose queries incrementally, therefore resulting in simultaneous visualizations of many component results with varying degrees of error. Maintaining different component partial results rather than single, exact answers imposes a completely new set of challenges for both expressing and optimizing these types of queries. Currently, our IDEA prototype uses a preliminary interaction algebra to define a user's visual queries.

6.2 Visual Indexes

Similar to the algebra and optimizer, we also found that traditional indexes are suboptimal for interactive data exploration tasks. Existing incremental techniques either sort the data (e.g., database cracking [51]) or do not naturally support summary visualizations. However, sorting can destroy data randomness and, consequently, the ability to provide good estimates. Similarly, existing techniques generally index every tuple without considering any properties of the frontend (e.g., human perception limitations, visualization characteristics) and can require a lot of storage space, especially for highly dimensional data.

For example, some visualizations (e.g., histograms) require the system to scan all leaf pages in a traditional B-tree, since this index is designed for single-range requests rather than providing visual data summaries. We therefore proposed VisTrees [35], a new dynamic index structure that can efficiently provide approximate results specifically to answer visualization requests. The core idea of VisTrees is that the nodes within the index are "visually-balanced" to better serve visual user interactions and then compressed based on perception limitations.

6.3 Sample Management

As previously mentioned, IDEA caches as much data as possible from the underlying data sources in order to provide faster approximate results, since most data sources are significantly slower. For example, the memory bandwidth of modern hardware ranges from $40-50$GB/s per socket [10, 110], whereas we recently measured that PostgreSQL and a commercial DBMS can only export $40-120$MB/s, even with a warm cache holding all data in memory. Although DBMS export rates may improve in the future, IDEA's cache will still remain crucial for providing approximate answers to visual queries and supporting more complex analytics tasks (e.g., ML algorithms).

Conceptually, IDEA can be best referred to as a **sample management system** and roughly divides the memory into three parts: the *Result Cache*, the *Sample Store*, and space for *Indexes*. When triggered by an initial user interaction, IDEA begins ingesting data from the various data sources, speculatively performing operations and caching the results in the *Result Cache* to support possible future interactions. At the same time, IDEA also caches all incoming data in the *Sample Store* using a compressed row format. When the available memory for the *Sample Store* is depleted, IDEA starts to update the cache using a reservoir sampling strategy to eventually create a representative sample over the whole data set even if the data stream is biased. To further mitigate the impact of bias in the data stream, IDEA takes advantage of sampling operators most database systems provide as well as reads from random offsets of the data (e.g., when connected to a file). Furthermore, IDEA might decide to split up the reservoir sample into several stratified subsamples to overrepresent the tails of the distribution, or to create specialized indexes for potential future queries. This is done based on the current visualizations on the screen as they, for example, determine what filter chains the user is able to create. All these decisions are constantly optimized based on both past and current user interactions. For example, if the user drags a new attribute onto the canvas, the system will allocate more resources to the new attribute in preparation for potential follow-up queries. At the same time, IDEA constantly streams increasingly precise results to the frontend as the computation progresses over the data, along with indications about both the completeness and error estimates.

6.4 Approximating Black Boxes

In contrast to most other AQP engines, IDEA has the goal of approximating complex analytics and machine-learning pipelines. This is particularly challenging as many operations are black boxes to IDEA. For example, IDEA is fully compatible with scikit-learn [52] and even allows users to add new Python operations. In order to approximate results for these operations, IDEA uses a relatively simple idea: it executes the operation first over a small sample and then re-executes the operation over progressively increasing sample sizes. However, this creates a whole new set of challenges, e.g., what is a good sample size to start with and in what increments should it be made larger? We address some of these questions in more detail when we discuss Northstar ML auto-tuning capabilities (Section 7).

6.5 Result Reuse

As outlined earlier, query approximation for visual interactive Data Science has its own set of challenges but also provides a vast array of opportunities, one of them being reuse. Visual tools have encouraged a more conversational interaction paradigm [43], whereby users incrementally compose and iteratively refine queries throughout the data exploration process. Moreover, this style of interaction also results in several seconds (or even minutes) of user "think time" where the system is completely idle. Thus, these two key features provide an AQP system with ample opportunities to (1) reuse previously computed (approximate) results across queries during a session and (2) take actions to prepare for potential future queries.

However, it turned out that there existed close to no work to efficiently reuse approximate results. To that end, we made an interesting and, in hindsight, conceivably trivial observation: almost all visualizations convey simple statistics about the data. Based on that observation, we developed a new AQP formulation that treats aggregate query answers as random variables

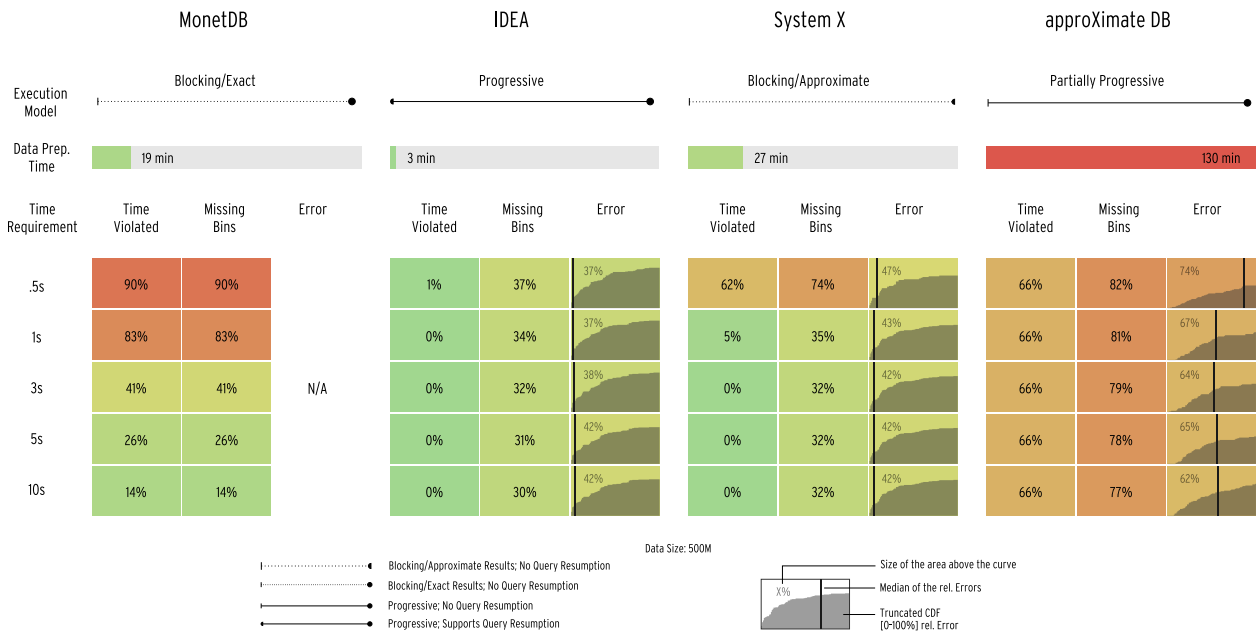

Figure 3: IDEBench results for four systems as a summary report. Results show the mean percentage of time violations and missing histogram bins, as well as the mean relative errors (MREs) and MRE CDF for the approximated results; the greater the proportion of small errors, the smaller the area above the curve.

to enable reuse of approximate results with formal reasoning about error propagation across overlapping queries [43].

For example, consider a simple bar chart showing the count per category of the following SQL query:

```
SELECT sex, COUNT(*)
FROM census
GROUP BY sex
```

One way to model a group-by attribute is to treat it as a categorical random variable X, where X can assume any one of the possible outcomes from the sample space Ω_X. For example, a random variable modeling the *sex* attribute can take on an outcome in the sample space $\Omega_{sex} = \{Male, Female\}$.

More interestingly, filter chains as shown in Figure 1 can be expressed as conditional variables. For example, assume that a *sex* bar chart was linked to a *salary* bar chart as the downstream operation, and only 'Female' was selected. Let us further assume that Y is the random variable for the salary distribution. Then, we can determine the proportional height of the bar for the salary range $0 - 10k$ as $P(Y \leq 10k|X = \text{Female})$.

Surprisingly, this view of query results for visualizations not only made it easier to estimate the quality of results for each operation in an incrementally composed workflow, but also opened up many new opportunities for result reuse. For example, it allows "query rewrites" by means of Bayes' Theorem or the Law of Total Probability.

6.6 Results

We recently evaluated IDEA against other systems to create a benchmark for interactive data exploration, called IDEBench [33]. The key result of this study is shown in Figure 3 and compares MonetDB, IDEA, approXimateDB, and a commercial in-memory AQP system (referred to as System X) with respect to the data loading time and data quality after X seconds over 500MB of flight data [81] for 10 different interactive exploration workflows (see [33] for more details). As the figure shows, *IDEA* overall is able to return the fastest and even meets a targeted return time of 500ms around 99% of the time with significantly less data prep time. In this case, the data was stored on file for all systems and we gave idea 3 minutes from "connecting to" the file to exploring. IDEA also has fewer missing bins in approximated histograms, and the mean relative error over all returned results is significantly less than *approXimateDB*'s and marginally less than *System X*'s'. One of the key reaons is, that the benchmark simulates a user who explores the data set by incrementally building queries and that IDEA can explore the incremental nature of requests. Finally, it should be noted, that in contrast to the next best system, the commercial AQP engine System X, IDEA features progressive results, and in contrast to all other systems, it can also approximate the execution of machine-learning algorithms.

7. Alpine Meadow: A QUERY OPTIMIZER FOR MACHINE LEARNING

One key promise of our work is to help users quickly arrive at an initial solution, often a model, in a collaborative meeting. Unfortunately, selecting the right ML algorithm and hyper-parameter tuning is often a time-consuming and boring process. In 2013, with MLbase [64], we took the first step toward unlocking the power of ML for end users. MLbase provided (1) a simple declarative way to specify ML tasks and (2) a novel optimizer to select and dynamically adapt the choice of learning algorithm. One of the key contributions of this work

was TuPAQ [95], a novel bandit-based hyper-parameter tuning strategy that was the predecessor of Hyperband [67]. However, MLbase was not designed for interactive environments and the process of hyper-parameter tuning could take hours. Therefore, we started the Alpine Meadow project based on our experience with MLbase. Alpine Meadow and MLbase have several commonalities: they both use a query-optimizer-based approach to ML auto-tuning and, for example, use bandits to efficiently explore the search space. But there are also significant differences—most importantly, the focus on interactivity. In the remainder of this section, we outline in greater detail what makes Alpine Meadow unique.

7.1 Focus on End-to-End Learning

Alpine Meadow was developed as part of the DARPA D3M [92] project, which aims to entirely automate machine learning. Every six months, DARPA evaluates all systems on how well they can autonomously solve a set of problems. The problems range from text classification tasks to building models that automatically measure wrist length from images. Each task comes in the form of a JSON description containing the goal and a description of the available data for the task. The system then needs to automatically find the best pipeline in a given time frame (e.g., 10 minutes) and is evaluated using some quality metric (e.g., F1 score). Therefore, in contrast to MLbase and many other ML auto-tuning systems, Alpine Meadow's goal is to automatically create the entire end-to-end workflow from data cleaning operations to the model, including deep-learning models. Also in contrast to MLbase, Alpine Meadow is designed to take user feedback into account. Meaning, a user can steer the exploration, for example, by proposing new features or presetting certain operations, such as cleaning operations.

7.2 Interactivity

The most notable difference between Alpine Meadow and other ML auto-tuning systems is the focus on interactivity. Alpine Meadow aims to provide a first answer in seconds, which is then refined in the background. Consequently, it usually tries simpler pipelines over small samples first before increasing the sample size and model complexity.

Furthermore, we developed a cost-quality model to measure the "promisingness" of a pipeline over a given sample. By filtering out pipelines with high cost, we can prune the search space, saving resources and reducing the overall search latency. For now, our cost-quality model estimates three factors for a given pipeline over a sample of size m: (1) the time for training and testing the pipeline; (2) the expected quality gain of a pipeline over the last best solution; (3) the risk of a pipeline, that is, the variance of quality. These factors are then weighted differently over time. For example, at first, the training time is given more weight as we want to return a good solution as quickly as possible to the user, potentially sacrificing quality. Later we assume that the quality of a pipeline matters more and we allow proportionally more expensive pipelines to run to eventually find the best performing model. . To actually build the cost model we use (1) learned rules (see next section) as well as (2) the history of past and ongoing training steps (see [9] for more details).

7.3 Learned Rule-Based Optimization and Transfer Learning

We extensively use meta-learning techniques to utilize history from similar problems. Most importantly, our optimizer derives best-practice rules from past experience and uses them to create and prune the search space as well as prioritize pipelines. To jump-start the system, we trained it using publicly available competitions (e.g., Kaggle) as well as the sample problems that DARPA provided (note that DARPA has a separate set of problems that we have never seen for evaluation purposes). Furthermore, we made the rules problem-specific. Based on the techniques of [41], we determine the similarity of every problem to previous problems and adjust the importance of rules based on this similarity.

This approach further allows us to transfer existing solutions to new problems. For example, for an image classification task, we might use an existing deep-learning model that was trained on CIFAR, "chop off its head" (i.e., remove the top neuron layers), and replace it with a new set of layers for the given task (a common transfer learning technique). As a result, we are often able to train more complex models in a short amount of time, as our optimizer prefers to start from existing solutions. A nice side effect of the rule- and transfer-learning-based approach is that it also makes the final pipelines simpler to explain to the user.

7.4 User Interactions with Alpine Meadow

While Alpine Meadow can be used without Vizdom and IDEA, it unfolds it full potentially when used together. For example, using Vizdom users can at any time steer the search process, e.g., by restricting the model type or feature preprocessing steps, adding new features, and/or restricting the model building to subpopulations of the data. Furthermore, IDEA ensures that all operations, including the machine learning model, return first results in sub-seconds after every user interaction, and that the key characteristics of the so far best models are always visualized to the user. Furthermore, the best pipelines can be interactively inspected and modified, and the output of a model can be used as an input for other operations in Vizdom, making it for example possible to quickly analyze on what data the model does not perform well on, or to use the model itself to label unlabeled data, etc.

7.5 Initial Results

Figure 4 shows the performance of Alpine Meadow against other systems competing in D3M, which includes teams from Stanford, NYU, UC Berkeley, and many others (anonymized as Systems 2-10). The figure shows how many of the DARPA-provided prediction tasks each system is able to solve, how often they are better than the expert solution, as well as the normalized utility score, which is defined as $\sum_i \frac{s_i - b_i}{|b_i|}$, with s_i (b_i) being the performance of the system (baseline) on problem i. As of March 2018, Alpine Meadow solves not only all of the DARPA-provided challenges, but is also able to outperform the expert solution in 80% of the cases. Furthermore, the normalized score shows that it provides overall good solutions (i.e., if the system were only a bit better in 80% of the cases but otherwise much worse, the score would likely be negative).

	Solved Problems	Better Than Baseline	Normalized Score
Alpine Meadow	100%	80%	0.42
System 2	40%	27%	0.09
System 3	40%	13%	0.02
DARPA Baseline	100%	0%	0.00
System 4	20%	7%	-0.07
System 5	87%	47%	-0.16
System 6	27%	7%	-0.22
System 7	60%	20%	-0.59
System 8	87%	53%	-0.75
System 9	60%	20%	-1.14
System 10	60%	20%	-4.57

Figure 4: DARPA D3M Competition Results on 03/2018

8. QUDE: QUANTIFYING THE UNCERTAINTY IN DATA EXPLORATION

While visual tools are key to democratizing Data Science, they also bring new risks. For example, data is often massaged, filtered, and visualized until the domain expert sees something interesting, and only then is a statistical test performed. However, this ignores the "fishing expedition" before the test—and the increased risk of a false discovery because of it. We therefore believe that a system should automatically track potential common mistakes within a Data Science pipeline. But multi-hypothesis errors are just one type of potential problem among many within Data Science pipelines. Others include the Yule-Simpson effect or, when training models, imbalance of labels or common problems in representing null values. If we want to empower a broader class of users without deep statistical or machine-learning backgrounds to analyze data sets, we should work toward automatically protecting them from these (common) mistakes. Over the last one to two years, we have started to address this problem by developing QUDE (pronounced "cute"), a tool for Quantifying the Uncertainty in Data Exploration. In the following, we highlight four areas where QUDE already helps to detect, quantify, and sometimes even correct common problems.

8.1 Uncertainty as Unknown Unknowns

Incompleteness of data is one of the most common sources of uncertainty in practice. For instance, if unknown data items are missing from the unknown target population (i.e., *we can't tell if the database is complete or not*), even a simple aggregate query result, like `SUM`, can be questionable.

We therefore started to develop techniques that estimate not only the amount of missing data based on techniques from [100, 101, 102] but also the impact those items might have on query results [19, 20]. We assume a simple data integration scenario in which (semi-)independent data sources are integrated into a single database. The overlap between the different data sets allows us to estimate the number of missing items using species estimation techniques [100]. Further, it is possible to make estimates about the values the missing items might have using our novel bucket estimator [19]. This way, Vizdom is able to indicate to the user how much impact missing data might have on the visualization.

8.2 Uncertainty as Undetected Data Errors

For a data scientist, it is important to know whether a data set is clean enough to begin analysis or if it is worthwhile to invest more time and money in cleaning. In the best case, these unknown errors are unimportant corner cases, but often enough they can be crucially overlooked problems that significantly affect any subsequent analytics. This raises a fundamental question: Is it possible to quantify the data quality of a data set with regard to the number of remaining errors in the data set?

While this is a seemingly simple question, it is actually extremely challenging to define data quality without knowing the ground truth [85, 14, 38, 91, 39, 58]. A simple approach is to extrapolate the number of errors from a small "perfectly clean" sample [104, 108, 11, 16, 22, 71]): (1) we take a small sample, (2) perfectly clean it manually or with the crowd, and (3) extrapolate our findings to the entire data set. For example, if we found 10 new errors in a sample of 1000 records out of 1M records, we would assume that the total data set contains 10000 additional errors. However, this naïvee approach presents a chicken-and-egg paradox. If we clean a small sample of data, it may not be representative and thus will give an inaccurate estimate. For larger samples, how can the analyst know that the sample itself is perfectly clean without a quality metric?

We therefore developed the Data Quality Metric (DQM) [18], a statistical estimator based on the principle of diminishing returns, which basically states that every additional error is more difficult to detect. For example, with experts or crowdsourcing, the first (crowd-)worker to pass over the data set finds more new errors than every subsequent worker, and so on. The key insight is to estimate this diminishing return rate (i.e., fewer errors are found in each pass) and project this rate forward to estimate the number of errors if there were an infinite number of workers (i.e., all discoverable errors). The ratio of the current errors to the estimated discoverable errors then functions as an indicator of the cleanliness of the data set.

8.3 Uncertainty as False Discovery

While interactivity is key to the usability of advanced visual analytical tools [69], using them also significantly increases the risk of making spurious discoveries. Such risk has two aspects: **(1)** the statistical significance of the visualized results is unclear, and **(2)** the growing number of hypotheses being tested during exploration increases with every single visualization.

The first aspect of risk is important because visualizations have the power to influence human perception and understanding.Suppose that an ice cream company salesperson is exploring a data set about sales. First, she wants to get a yearly distribution of the sales figures. So, she compares the sales of the last five years using a histogram of sales per year. In the second step, she is interested in learning if sales differ significantly across states. She thus compares sales per state over the last five years.

Suppose the histogram shows that sales in Vermont were higher than in Rhode Island. Consider how tempting it is for an unsophisticated user to conclude that Vermonters buy more ice cream just based on the visualization. Although a statistically inclined user would formally analyze this observation by using hypothesis testing, she would have to redirect her attention to work with a different statistical tool (e.g., R) before proceeding to the next data exploration step. After such a context switch, the insight might turn out completely wrong due to random noise. At scale, the division of labor between data exploration

and hypothesis testing will cause even more waste of human efforts on such spurious insights. Thus, *if a visualization provides any insights, these should be tested immediately for their significance*. If that is not be the case, the value of the visualization would be very limited, as the user would not be allowed to make any conclusions based on the visualization. Thus, if we consider a visualization as something more than a pretty picture presented to the user (i.e., more than just a listing of facts), we should always test the insight the user gains from the visualization for its significance and inform the user about it. A central challenge of our work is the understanding of the hypothesis derived by the user given a certain data visualization. With respect to the previous example, the hypothesis derived by the user could be: (1) Vermonters buy more ice cream than Rhode Islanders, (2) Rhode Islanders buy more than Vermonters, or (3) they buy ice cream in the same amount.

The second aspect of risk is arguably even more severe. With every additional hypothesis test, the chance of finding a false discovery increases. This problem is known as the "*multiple comparisons problem*" (MCP) and has been studied extensively in statistics literature [12, 7, 46, 60].

Data exploration on systems such as Vizdom [26] or Tableau not only increase the risk of false discovery, but also changes the way that statistical tests are applied. Suppose in the previous example the salesperson explores various relationships in the sales data set through visualizations until she sees a visualization that she deems useful (e.g., significantly more ice cream sales to males in Massachusetts compared to California). With some background in statistics, she validates this insight by using an appropriate test with a significance level of 5%. Suppose the observed p-value is below the significance level, so she rejects the null hypothesis and believes that there is only a 5% chance that she incorrectly rejected the null hypothesis in case it was true. However, this way of applying statistical testing is wrong. What the user ignores is that before she did the test, she had already searched through the data set for a while and observed different insights and, implicitly, their corresponding hypotheses, albeit untested. Thus, by the time the user applied the statistical test, she was already inadvertently trapped in the multiple comparisons problem. This was also confirmed in a recent user study we performed [115]. In our experiment using synthetic data sets with known ground truth labels, over 60% of user insights were false.

Unfortunately, existing work to control for MCP are often not directly applicable to interactive data exploration or have other severe drawbacks. For example, the most common approach of using holdouts allows verifying the gathered insights just **once**; any additional data exploration session would require a complete new holdout unless the testing over the holdout was MCP-controlled, so the problem remains (see [116] for more details). Furthermore, splitting a data set into exploration and holdout data sets can significantly lower the power (i.e., the chance to find real insights), especially for rare events. In contrast, statistical techniques such as the Benjamini-Hochberg procedure to bound the False Discovery Rate (FDR) or the Bonferroni procedure to bound the Family-Wise Error Rate (FWER) were not designed for incremental testing. We therefore started to develop techniques which (1) try to infer based on the visualizations what the user might be inferring/testing and (2) automatically control the FDR rate. For example, in [116, 117] we develop an automatic MCP control based on alpha-investing fully integrated into Vizdom. However, the whole area is still in its infancy and many interesting research challenges remain.

8.4 Uncertainty as Hidden Facts

Visualizations as a form of aggregation can "hide" data errors or the incompleteness of data, and sometimes even mislead. As mentioned in Section 3, we recently observed this phenomenon when analyzing the age distribution of patients in the MIMIC-II data set. The distribution was visualized using histograms with a bucket size of 10 years. Nothing was suspicious about the visualization, which showed that very young and older people are in the emergency room slightly more often. Only after zooming in did we find that the data set did not contain any patients between 1 and 9 years; rather, all patients in this bucket had an age of 0. It turned out that the data came from an emergency room for adults and the value 0 was used if the age was not known. Even more severe, filtering out the 0-aged patients significantly changed our conclusions regarding younger and older patients.

Inspired by this result, we developed techniques [47] to automatically detect forms of Simpson's Paradox, which is a special type of error in which a high-level aggregation leads to a wrong conclusion. The two main challenges to enabling efficient online detection are sheer data size as well as the number of different attribute combinations that need to be tested. Therefore, for our algorithms, we applied two main techniques: (1) For dealing with large data sets, we developed a set of approximate algorithms that can stream over the data and decide in a probabilistic manner if the data is likely to contain a Simpson's Paradox. This allows our algorithms to make a prediction at interactive speeds of how likely it is to find a paradox after seeing only a small amount of data. (2) Since many different attribute combinations need to be tested to detect a Simpson's Paradox, we devised a technique that leverages ideas from multiarmed bandits to find a good trade-off between exploration and exploitation. These techniques allow us to scale out to large data sets or data sets with many different attributes.

To summarize, QUDE is a first step toward building an assistant that can help domain experts who are not trained data scientists to make discoveries on their own. However, clearly we are still at the beginning, and many interesting research challenges remain open as described in Section 10.

9. RELATED WORK

Our work around Northstar spans many different research areas — in fact, so many areas that we need to refer to the individual publications around Northstar for a more detailed listing of related work and only highlight a few of them here.

Vizdom: the visualization community has produced various systems that facilitate data exploration by domain experts. For example, Tableau and its research predecessors Polaris [97] and imMens [70] are systems for analyzing data sets through visualizations with a high degree of customizability. To achieve the low latencies required for user-driven data exploration, these

systems either use heavily optimized DBMSs or precomputation of results. Though we support similar data exploration workflows, we apply an intuitive pen-&-touch interface in Vizdom and focus on progressive computation and visualization to guarantee interactivity thresholds. Furthermore, we aim to support users in the entire Data Science process from data preparation over exploration to model building.

IDEA: Most related to IDEA is the work in approximate query processing using sampling [3] and online aggregation [50]. However, systems that use biased sampling (e.g., AQUA [4], BlinkDB [5], DICE [56]) typically require extensive preprocessing or foreknowledge about the expected workload, which goes against the ad hoc nature of interactive data exploration. On the other hand, systems that perform online aggregation (e.g., CONTROL [49], DBO [55], HOP [21], FluoDB [112]) typically cannot deal with black-box operations and/or outliers. Verdict [84] uses the results of past queries to improve approximations for future queries in a process called "database learning." However, Verdict requires up-front offline parameter learning, as well as a sufficient number of training queries, in order for users to begin seeing large benefits. Also related to IDEA are techniques to improve the "visual experience;" for example, approximated histogram [73], better smoothing of timeseries data [89], or visualizing data transformations [59]. Those techniques are largely orthogonal to the ideas presented here, but it would be very interesting to integrate them into Northstar. Most related to IDEA's execution model are probably FluxQuery [31] and zenvisage [94], although their focus is on SQL workloads. Finally, in order to better support user sessions in DBMSs, various techniques have been developed to reuse results [98, 54, 79, 29]. Nonetheless, these techniques do not consider reuse in the context of (partial) query results with associated error.

Alpine Meadow: Most related to Alpine Meadow is the work on ML algorithm selection and hyper-parameter tuning [96, 8, 72, 68, 66, 45]. For example, TuPAQ [96] and Hyperband [66] use variations of the multiarmed bandit (MAB) algorithm to better allocate computational resources for hyperparameter tuning. Other solutions like Auto-WEKA [99, 62] or its sister package Auto-sklearn [40] are more similar to our approach, as they also consider various feature selection and data transformation algorithms with the intent of generating ML pipelines. Still, these solutions are built for offline use and run for a predefined amount of time, rendering them unfeasible for interactive settings. Furthermore, they also do not consider transfer learning or data cleaning steps, and do not offer good support to take user input into account.

QUDE: Obviously, QUDE builds upon the vast amount of techniques developed in the statistics community (see [48] for an overview). Surprisingly, even though statistical errors are highly common [53], there is very little work in automating these techniques in the form of a Data Science assistant to prevent layman users. Some notable exceptions are the recent efforts to automatically detect bias in machine learning, algorithm building, and analytics [44, 109] or the use of perceptual models to quantify when approximations are safe [6].

10. THE FUTURE OF INTERACTIVE DATA SCIENCE

Northstar provides one of the very first interactive Data Science environments with the goal of democratizing Data Science. It addresses a wide range of research problems to make analytics and model building more interactive, many of which we discovered only during the course of this project. However, even more challenges remain, and we believe interactive Data Science and the goal of making analytics more accessible for a broader range of users creates a new research field in itself. We highlight a few potential future research challenges below.

10.1 Formal Execution Model

As outlined in Section 6.1, the execution model of IDEA is not a traditional query processor, a pure AQP engine, or a streaming engine. However, a precise model for the execution engine does not yet exist. Arguably, our AQP formulation of Section 6.5 goes in the right direction, but it is mainly focused on the reuse of results. Even further complexity arises when considering more diverse data models, such as time series data, which come with their own semantics and algebra [34]. Nonetheless, we believe that a strong, formalized foundation can guide the development of future accelerators for interactive Data Science and their optimizations.

10.2 Other Data Types

Many real-world use cases require dealing with more than one data type. For example, a sales prediction model might start out with structured data gathered from a data warehouse, but then might include features from the product description, reviews, or the quality of product pictures. Thus, we need tools which allow data scientists to deal efficiently with different data types from semi-structured data to videos. This is particularly challenging as many of the state-of-the-art models for conceptual tasks (e.g., video object detection, audio transcription, or entity discovery in text) are neural nets, which are notoriously hard to train in sub-seconds. However, techniques like transfer learning [105] might make it possible, as it allows us to start from a good existing solution. Yet it is not without challenges, as the problem now becomes how to efficiently find the most relevant already-trained model and how to best integrate it into the current analytical pipeline.

10.3 Data Integration and Transformation

Data scientists commonly want to integrate several data sources, which often requires an intensive data cleaning and munging [57] to get all the data into the right format and consolidated so it can actually be analyzed. While there has been a lot of work in data integration [28], most of the existing work is not necessarily result-oriented, is computationally very expensive, and/or still imposes a lot of scripting on users. For example, it would be great if the system could highlight how potential data errors might influence the individual operations on the screen (see also 8) and what concrete cleaning steps a user might want to take. Similarly, the process of transformation should be made easier. Systems like Trifacta [2] or BoostClean [65] are already big steps in the right direction; however, we believe the integration with the Data Science analytics system must

be much closer to better support the iterative Data Science life cycle. Ideally, users should be able to seamlessly move between the different tasks of data collecting, cleaning and integration, explorative analysis, and building and evaluating models, all in a visual manner.

10.4 Better Approximation Techniques for Legacy Systems

As outlined throughout the entire paper, approximate query processing and progressive results are key to enabling a fluid user experience. While many techniques have been developed over the years [13], we found several issues keeping them from beings fully applicable for us. Most importantly, commercial database systems currently barely support AQP, and it is unrealistic to assume that existing legacy systems, such as a data warehouse or distributed file system, can easily be replaced with an AQP engine. This was one of the reasons we implemented our AQP techniques in the middle layer, IDEA, which then connects to the existing legacy system without actually changing it. However, this design imposes a whole new set of challenges, including: (1) how IDEA can guarantee that it gets a sufficiently randomized data stream out of the legacy system (see also [82]), (2) how IDEA can best take advantage of the features the legacy system provides (e.g., predicate push-downs) while maintaining fast response times for initial results, and (3) the possibility of leveraging the underlying system to better deal with extreme values/outliers (e.g., if the data warehouse has an index, can the index be partially used?).

Another exciting future direction for AQP techniques is to learn models that represent the data distribution, as done in learned indexes [63]. [63] shows how a CDF model can be used to enhance index structures with quite promising results. Now assuming a CDF model exists, for example in the form of a learned index, the same model could now also be directly used to answer queries. In fact, our AQP formulation for enabling approximate result reuse from Section 6.5 already provides the foundation to do that, as it treats results as random variables.

10.5 Risk Control

As outlined in Section 8, democratizing Data Science should also mean protecting users from common (and not-so-common) mistakes. With QUDE, we took a first step in this direction, but many open challenges remain. For example, there is a growing trend toward creating recommendation engines, which propose interesting visualizations (e.g., [103, 74, 32, 90, 88]) or exploration steps [75], or automatically test for correlations [15]. Those systems are potentially checking thousands of hypotheses in just a few seconds and are smoking guns disguised as water pistols. As a result, it is almost guaranteed that the system will find something "interesting" regardless of whether the observed phenomenon is statistically relevant. Even worse, without knowing how exactly the system tried to find something "interesting" (e.g., a visualization, correlation, etc.) and how many correlations were tested, it is later often impossible to correct for the MHP. In some cases, like data polygamy, even a holdout technique does not work.

The same also holds true for automatically finding machine-learning models as done with Alpine Meadow, which can be regarded as a model recommendation engine. That is, training machine-learning models can be seen as a form of testing, with the hypothesis being that a given model generalizes over unseen data, which is usually evaluated using techniques like cross-validation. However, the more model pipelines and hyper-parameters we test, the greater the chance that we find a model that just by chance works well over the cross-validation data sets (see [30] for a more detailed description). Hence, controlling the MHP for recommendation engines is a largely unsolved research question and we only recently made some interesting progress on that front by using VC dimensions.

Similarly, there exist many more types of errors that a system could warn the user about. For example, there has been a lot of excitement recently about automatically detecting bias [44, 109, 17] or helping the user to understand results using lineage [86, 87]. We therefore believe that there is a lot of potential for building a virtual Data Science Assistant, a tool that helps the domain expert in the discovery and model-building process and prevents him from making mistakes.

11. CONCLUSION

We speculate that in the near future many conference rooms will be equipped with an interactive whiteboard, like the Microsoft Surface Hub, and that we can use such whiteboards to enable domain experts and data scientists to work together during a single meeting to visualize, transform, and analyze even the most complex data on the spot. We showed that democratizing Data Science requires us to completely rethink the full analytical stack, from the interface to the "guts," as well as take a more holistic view of problems and bridge various research communities. With Northstar, we explored a first system design for true interactive Data Science and collaboration using interactive whiteboards. Furthermore, Northstar's deployments in industry and academia, as well as our various user studies [113, 115, 114], have shown that Northstar helps to gain insights faster and avoid problematic discoveries, and adds significant value in practice.

12. ACKNOWLEDGMENTS

The work presented in this paper was mainly done by my fantastic PhD students and PostDocs, and they truly deserve all the credit. Most notably, I would like to thank my PostDoc Emanuel Zgraggen and my students Yeounoh Chung, Andrew Crotty, Alex Galakatos, Ani Kristo, Zeyuan Shang, Leonhard Spiegelberg, and Erfan Zamanian. Furthermore, I was very lucky to have many fantastic collaborators, most notably Carsten Binnig and Eli Upfal. I would also like to thank Andy Van Dam (who probably started this whole endeavor by introducing me to Emanuel), as well as Benedetto Buratti, Ugur Çetintemel, Cyrus Cousins, Philipp Eichmann, Yue Guo, Lorenzo De Stefani, Stan Zdonik, Robert C. Zeleznik, and Zheguang Zhao. Finally, I would like to thank Eugene Wu and Aditya Parameswaran for their valuable comments on this paper. This research was funded in part by the DARPA Award 16-43-D3M-FP-040, NSF CAREER Award IIS-1453171, NSF Award IIS-1514491, NSF Award IIS-1562657, Air Force YIP AWARD FA9550-15-1-0144, and gifts from Google, Microsoft, Intel, and Oracle.

13. REFERENCES

[1] Project tungsten: Bringing apache spark closer to bare metal. https://databricks.com/blog/2015/04/28/project-tungsten-bringing-spark-closer-to-bare-metal.html. Accessed: 2018-07-15.

[2] Trifacta. http://www.trifacta.com.

[3] S. Acharya, P. B. Gibbons, and V. Poosala. Congressional Samples for Approximate Answering of Group-By Queries. In *SIGMOD*, pages 487–498, 2000.

[4] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy. The Aqua Approximate Query Answering System. In *SIGMOD*, pages 574–576, 1999.

[5] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden, and I. Stoica. BlinkDB: Queries with Bounded Errors and Bounded Response Times on Very Large Data. In *EuroSys*, pages 29–42, 2013.

[6] D. Alabi and E. Wu. Pfunk-h: approximate query processing using perceptual models. In *HILDA@SIGMOD*, page 10, 2016.

[7] Y. Benjamini et al. Controlling the false discovery rate. *Journal of the Royal Statistical Society, Series B*, 57(5), 1995.

[8] J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

[9] C. Binnig, B. Buratti, Y. Chung, C. Cousins, T. Kraska, Z. Shang, E. Upfal, R. Zeleznik, and E. Zgraggen. Towards interactive curation & automatic tuning of ml pipelines. In *Proceedings of the Second Workshop on Data Management for End-To-End Machine Learning*, DEEM'18, pages 1:1–1:4, New York, NY, USA, 2018. ACM.

[10] C. Binnig, A. Crotty, A. Galakatos, T. Kraska, and E. Zamanian. The end of slow networks: It's time for a redesign. *PVLDB*, 9(7):528–539, 2016.

[11] P. Bohannon, W. Fan, M. Flaster, and R. Rastogi. A cost-based model and effective heuristic for repairing constraints by value modification. In *Proceedings of SIGMOD*, pages 143–154, 2005.

[12] C. E. Bonferroni. *Teoria statistica delle classi e calcolo delle probabilita*. Libreria internazionale Seeber, 1936.

[13] S. Chaudhuri, B. Ding, and S. Kandula. Approximate query processing: No silver bullet. In *SIGMOD*, pages 511–519, New York, NY, USA, 2017. ACM.

[14] Y. Cheah and B. Plale. Provenance quality assessment methodology and framework. *J. Data and Information Quality*, 5(3):9:1–9:20, 2015.

[15] F. Chirigati, H. Doraiswamy, T. Damoulas, and J. Freire. Data polygamy: The many-many relationships among urban spatio-temporal data sets. In *SIGMOD*, pages 1011–1025, 2016.

[16] J. Chomicki and J. Marcinkowski. Minimal-change integrity maintenance using tuple deletions. *Information and Computation*, 197(1-2):90–121, 2005.

[17] Y. Chung, T. Kraska, N. Polyzotis, and S. E. Whang. Slice finder: Automated data slicing for model validation. *CoRR*, abs/1807.06068, 2018.

[18] Y. Chung, S. Krishnan, and T. Kraska. A data quality metric (DQM): how to estimate the number of undetected errors in data sets. *PVLDB*, 10(10):1094–1105, 2017.

[19] Y. Chung, M. L. Mortensen, C. Binnig, and T. Kraska. Estimating the impact of unknown unknowns on aggregate query results. In *SIGMOD*, pages 861–876, 2016.

[20] Y. Chung, M. L. Mortensen, C. Binnig, and T. Kraska. Estimating the impact of unknown unknowns on aggregate query results. *ACM Trans. Database Syst.*, 43(1):3:1–3:37, 2018.

[21] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, K. Elmeleegy, and R. Sears. MapReduce Online. In *NSDI*, pages 313–328, 2010.

[22] G. Cong, W. Fan, F. Geerts, X. Jia, and S. Ma. Improving data quality: Consistency and accuracy. In *Proc. of VLDB*, pages 315–326, 2007.

[23] M. Correll and M. Gleicher. Error bars considered harmful: Exploring alternate encodings for mean and error. *TVCG*, 20(12):2142–2151, 2014.

[24] A. Crotty, A. Galakatos, K. Dursun, T. Kraska, C. Binnig, U. Çetintemel, and S. Zdonik. An architecture for compiling udf-centric workflows. *PVLDB*, 8(12):1466–1477, 2015.

[25] A. Crotty, A. Galakatos, K. Dursun, T. Kraska, U. Çetintemel, and S. B. Zdonik. Tupleware: "big" data, big analytics, small clusters. In *CIDR*, 2015.

[26] A. Crotty, A. Galakatos, E. Zgraggen, C. Binnig, and T. Kraska. Vizdom: interactive analytics through pen and touch. *PVLDB*, 8(12):2024–2027, 2015.

[27] A. Crotty, A. Galakatos, E. Zgraggen, C. Binnig, and T. Kraska. The case for interactive data exploration accelerators (IDEAs). In *HILDA@SIGMOD*, 2016.

[28] A. Doan, A. Halevy, and Z. Ives. *Principles of Data Integration*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 2012.

[29] K. Dursun, C. Binnig, U. Çetintemel, and T. Kraska. Revisiting Reuse in Main Memory Database Systems. In *SIGMOD*, pages 1275–1289, 2017.

[30] C. Dwork, V. Feldman, M. Hardt, T. Pitassi, O. Reingold, and A. Roth. Generalization in adaptive data analysis and holdout reuse. In *NIPS*, pages 2350–2358, 2015.

[31] R. Ebenstein, N. Kamat, and A. Nandi. *FluxQuery*: An execution framework for highly interactive query workloads. In *SIGMOD*, pages 1333–1345, 2016.

[32] H. Ehsan, M. A. Sharaf, and P. K. Chrysanthis. Muve: Efficient multi-objective view recommendation for visual data exploration. In *ICDE*, pages 731–742, May 2016.

[33] P. Eichmann, C. Binnig, T. Kraska, and E. Zgraggen. Idebench: A benchmark for interactive data exploration. *CoRR*, abs/1804.02593, 2018.

[34] P. Eichmann, A. Crotty, A. Galakatos, and E. Zgraggen. Discrete time specifications in temporal queries. In *CHI*, pages 2536–2542, 2017.

[35] M. El-Hindi, Z. Zhao, C. Binnig, and T. Kraska. Vistrees: fast indexes for interactive data exploration. In *HILDA@SIGMOD*, 2016.

[36] N. Elmqvist, A. V. Moere, H.-C. Jetter, D. Cernea, H. Reiterer, and T. Jankun-Kelly. Fluid interaction for information visualization. *Information Visualization*, page 1473871611413180, 2011.

[37] G. M. Essertel, R. Y. Tahboub, J. M. Decker, K. J. Brown, K. Olukotun, and T. Rompf. Flare: Native compilation for heterogeneous workloads in apache spark. *CoRR*, abs/1703.08219, 2017.

[38] A. Even and G. Shankaranarayanan. Dual assessment of data quality in customer databases. *J. Data and Information Quality*, 1(3), 2009.

[39] W. Fan, F. Geerts, J. Li, and M. Xiong. Discovering conditional functional dependencies. *IEEE Trans. Knowl. Data Eng.*, 23(5):683–698, 2011.

[40] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter. Efficient and robust automated machine learning. In *NIPS*, pages 2962–2970, 2015.

[41] M. Feurer, J. T. Springenberg, and F. Hutter. Initializing bayesian hyperparameter optimization via meta-learning. In *AAAI*, pages 1128–1135, 2015.

[42] D. Fisher, R. DeLine, M. Czerwinski, and S. Drucker. Interactions with big data analytics. *interactions*, 19(3):50–59, 2012.

[43] A. Galakatos, A. Crotty, E. Zgraggen, C. Binnig, and T. Kraska. Revisiting reuse for approximate query processing. *PVLDB*, 10(10):1142–1153, 2017.

[44] S. Galhotra, Y. Brun, and A. Meliou. Fairness testing: testing software for discrimination. In *ESEC/FSE*, pages 498–510, 2017.

[45] D. Golovin, B. Solnik, S. Moitra, G. Kochanski, J. Karro, and D. Sculley. Google vizier: A service for black-box optimization. In *KDD*, pages 1487–1495. ACM, 2017.

[46] M. G. G'Sell et al. Sequential selection procedures and false discovery rate control. *Journal of the Royal Statistical Society*, 78(2), 2016.

[47] Y. Guo, C. Binnig, and T. Kraska. What you see is not what you get!: Detecting simpson's paradoxes during data exploration. In *HILDA@SIGMOD*, pages 2:1–2:5, 2017.

[48] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001.

[49] J. M. Hellerstein, R. Avnur, A. Chou, C. Hidber, C. Olston, V. Raman, T. Roth, and P. J. Haas. Interactive Data Analysis: The Control Project. *IEEE Computer*, pages 51–59, 1999.

[50] J. M. Hellerstein, P. J. Haas, and H. J. Wang. Online Aggregation. In *SIGMOD*, pages 171–182, 1997.

[51] S. Idreos, M. L. Kersten, and S. Manegold. Database Cracking. In *CIDR*, pages 68–78, 2007.

[52] INRIA. scikit-learn: Machine learning in python. http://scikit-learn.org. Accessed: 2018-07-15.

[53] J. P. A. Ioannidis. Why most published research findings are false. *Plos Med*, 2(8), 2005.

[54] M. Ivanova, M. L. Kersten, N. J. Nes, and R. Goncalves. An Architecture for Recycling Intermediates in a Column-Store. In *SIGMOD*, pages 309–320, 2009.

[55] C. M. Jermaine, S. Arumugam, A. Pol, and A. Dobra. Scalable Approximate Query Processing with the DBO Engine. In *SIGMOD*, pages 725–736, 2007.

[56] N. Kamat, P. Jayachandran, K. Tunga, and A. Nandi. Distributed and Interactive Cube Exploration. In *ICDE*, pages 472–483, 2014.

[57] S. Kandel, A. Paepcke, J. Hellerstein, and J. Heer. Wrangler: interactive visual specification of data transformation scripts. In *CHI*, 2011.

[58] K. Keeton, P. Mehra, and J. Wilkes. Do you know your iq?: a research agenda for information quality in systems. *SIGMETRICS Performance Evaluation Review*, 37(3):26–31, 2009.

[59] M. A. Khan, L. Xu, A. Nandi, and J. M. Hellerstein. Data tweening: Incremental visualization of data transforms. *PVLDB*, 10(6):661–672, 2017.

[60] A. Kirsch, M. Mitzenmacher, A. Pietracaprina, G. Pucci, E. Upfal, and F. Vandin. An efficient rigorous approach for identifying statistically significant frequent itemsets. *Journal of the ACM (JACM)*, 59(3):12, 2012.

[61] A. Kohn, V. Leis, and T. Neumann. Adaptive execution of compiled queries. In *ICDE*, 2018.

[62] L. Kotthoff, C. Thornton, H. H. Hoos, F. Hutter, and K. Leyton-Brown. Auto-weka 2.0: Automatic model selection and hyperparameter optimization in weka. *The Journal of Machine Learning Research*, 18(1):826–830, 2017.

[63] T. Kraska, A. Beutel, E. H. Chi, J. Dean, and N. Polyzotis. The case for learned index structures. In *SIGMOD*, pages 489–504, 2018.

[64] T. Kraska, A. Talwalkar, J. C. Duchi, R. Griffith, M. J. Franklin, and M. I. Jordan. Mlbase: A distributed machine-learning system. In *CIDR*, 2013.

[65] S. Krishnan, M. J. Franklin, K. Goldberg, and E. Wu. Boostclean: Automated error detection and repair for machine learning. *arXiv preprint arXiv:1711.01299*, 2017.

[66] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *arXiv preprint arXiv:1603.06560*, 2016.

[67] L. Li, K. G. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*, 18:185:1–185:52, 2017.

[68] T. Li, J. Zhong, J. Liu, W. Wu, and C. Zhang. Ease. ml: towards multi-tenant resource sharing for machine learning workloads. *PVLDB*, 11(5):607–620, 2018.

[69] Z. Liu and J. Heer. The effects of interactive latency on exploratory visual analysis. *InfoVis*, 2014.

[70] Z. Liu, B. Jiang, and J. Heer. *imMens*: Real-time visual querying of big data. *Comput. Graph. Forum*, 32(3):421–430, 2013.

[71] A. Lopatenko and L. Bravo. Efficient approximation algorithms for repairing inconsistent databases. In *ICDE*, pages 216–225, 2007.

[72] G. Luo. A review of automatic selection methods for machine learning algorithms and hyper-parameter values. *Network Modeling Analysis in Health Informatics and Bioinformatics*, 5(1):18, 2016.

[73] S. Macke, Y. Zhang, S. Huang, and A. G. Parameswaran. Adaptive sampling for rapidly matching histograms. *PVLDB*, 11(10):1262–1275, 2018.

[74] J. Mackinlay. Automating the design of graphical presentations of relational information. *ACM Trans. Graph.*, 5(2):110–141, Apr. 1986.

[75] T. Milo and A. Somech. Deep reinforcement-learning framework for exploratory data analysis. In *Workshop on Exploiting Artificial Intelligence Techniques for Data Management (aiDM@SIGMOD)*, pages 4:1–4:4, 2018.

[76] Microsoft hololens. `https://www.microsoft.com/en-us/hololens`. Accessed: 2018-07-15.

[77] Mimic ii data set. `https://mimic.physionet.org/`. Accessed: 2018-07-15.

[78] B. Myers, M. Oskin, and B. Howe. Compiling queries for high-performance computing. `https://dada.cs.washington.edu/research/tr/2016/02/UW-CSE-16-02-02.pdf`. Accessed: 2018-07-15.

[79] F. Nagel, P. A. Boncz, and S. Viglas. Recycling in Pipelined Query Evaluation. In *ICDE*, pages 338–349, 2013.

[80] J. Nielsen. Powers of 10: Time scales in user experience. *Retrieved January*, 5:2015, 2009.

[81] B. of Transportation Statistics. Bureau of transportation statistics. `http://www.transtats.bts.gov`, 2017. Accessed: 2017-10-21.

[82] F. Olken. *Random Sampling from Databases*. PhD thesis, University of California at Berkeley, 1993.

[83] S. Palkar, J. J. Thomas, D. Narayanan, P. Thaker, R. Palamuttam, P. Negi, A. Shanbhag, M. Schwarzkopf, H. Pirk, D. saman Amarasinghe, S. Madden, and M. Zaharia. Evaluating end-to-end optimization for data analytics applications in weld. *PVLDB*, 11(9):1002–1015, 2018.

[84] Y. Park, A. S. Tajik, M. J. Cafarella, and B. Mozafari. Database Learning: Toward a Database that Becomes Smarter Every Time. In *SIGMOD*, pages 587–602, 2017.

[85] L. Pipino, Y. W. Lee, and R. Y. Wang. Data quality assessment. *Commun. ACM*, 45(4):211–218, 2002.

[86] F. Psallidas and E. Wu. Provenance for interactive visualizations. In *HILDA@SIGMOD*, pages 9:1–9:8, 2018.

[87] F. Psallidas and E. Wu. Smoke: Fine-grained lineage at interactive speed. *PVLDB*, 11(6):719–732, 2018.

[88] X. Qin, Y. Luo, N. Tang, and G. Li. Deepeye: An automatic big data visualization framework. *Big Data Mining and Analytics*, 1(1):75–82, March 2018.

[89] K. Rong and P. Bailis. ASAP: prioritizing attention via time series smoothing. *PVLDB*, 10(11):1358–1369, 2017.

[90] S. F. Roth, J. Kolojejchick, J. Mattis, and J. Goldstein. Interactive graphic design using automatic presentation knowledge. In *SIGCHI*, pages 112–117, 1994.

[91] V. Sessions and M. Valtorta. Towards a method for data accuracy assessment utilizing a bayesian network learning algorithm. *J. Data and Information Quality*, 1(3), 2009.

[92] W. Shen. Data-driven discovery of models (d3m). `https://www.darpa.mil/program/data-driven-discovery-of-models`. Accessed: 2018-07-15.

[93] B. Shneiderman. Response time and display rate in human performance with computers. *ACM Computing Surveys (CSUR)*, 16(3):265–285, 1984.

[94] T. Siddiqui, A. Kim, J. Lee, K. Karahalios, and A. Parameswaran. Effortless data exploration with zenvisage: An expressive and interactive visual analytics system. *PVLDB*, 10(4):457–468, 2016.

[95] E. R. Sparks, A. Talwalkar, M. J. Franklin, M. I. Jordan, and T. Kraska. Tupaq: An efficient planner for large-scale predictive analytic queries. *CoRR*, abs/1502.00068, 2015.

[96] E. R. Sparks, A. Talwalkar, D. Haas, M. J. Franklin, M. I. Jordan, and T. Kraska. Automating model search for large scale machine learning. In *SOCC*, pages 368–380, 2015.

[97] C. Stolte, D. Tang, and P. Hanrahan. Polaris: A system for query, analysis, and visualization of multidimensional relational databases. *IEEE Trans. Vis. Comput. Graph.*, 8(1):52–65, 2002.

[98] K. Tan, S. Goh, and B. C. Ooi. Cache-on-Demand: Recycling with Certainty. In *ICDE*, pages 633–640, 2001.

[99] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown. Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. In *KDD*, pages 847–855. ACM, 2013.

[100] B. Trushkowsky, T. Kraska, M. J. Franklin, and P. Sarkar. Crowdsourced enumeration queries. In *ICDE*, pages 673–684, 2013.

[101] B. Trushkowsky, T. Kraska, M. J. Franklin, and P. Sarkar. Answering enumeration queries with the crowd. *Commun. ACM*, 59(1):118–127, 2016.

[102] B. Trushkowsky, T. Kraska, M. J. Franklin, P. Sarkar, and V. Ramachandran. Crowdsourcing enumeration queries: Estimators and interfaces. *IEEE Trans. Knowl. Data Eng.*, 27(7):1796–1809, 2015.

[103] M. Vartak, S. Madden, A. G. Parameswaran, and N. Polyzotis. SEEDB: automatically generating query visualizations. *PVLDB*, 7(13):1581–1584, 2014.

[104] J. Wang, S. Krishnan, M. J. Franklin, K. Goldberg, T. Kraska, and T. Milo. A sample-and-clean framework for fast and accurate query processing on dirty data. In *SIGMOD*, pages 469–480, 2014.

[105] K. Weiss, T. M. Khoshgoftaar, and D. Wang. A survey of transfer learning. *Journal of Big Data*, 3(1):9, May 2016.

[106] E. Wu, L. Battle, and S. R. Madden. The case for data visualization management systems: Vision paper. *PVLDB*, 7(10):903–906, 2014.

[107] E. Wu, F. Psallidas, Z. Miao, H. Zhang, L. Rettig, Y. Wu, and T. Sellam. Combining design and performance in a data visualization management system. In *CIDR*, 2017.

[108] M. Yakout, A. K. Elmagarmid, J. Neville, M. Ouzzani, and I. F. Ilyas. Guided data repair. *PVLDB*, 4(5):279–289, 2011.

[109] K. Yang and J. Stoyanovich. Measuring fairness in ranked outputs. In *SSDBM*, pages 22:1–22:6, 2017.

[110] E. Zamanian, C. Binnig, T. Kraska, and T. Harris. The end of a myth: Distributed transaction can scale. *PVLDB*, 10(6):685–696, 2017.

[111] zdnet. Microsoft: Surface hub demand is strong; product is now in stock. `https://www.cs.waikato.ac.nz/ml/weka/`.

[112] K. Zeng, S. Agarwal, A. Dave, M. Armbrust, and I. Stoica. G-OLA: Generalized On-Line Aggregation for Interactive Analysis on Big Data. In *SIGMOD*, pages 913–918, 2015.

[113] E. Zgraggen, A. Galakatos, A. Crotty, J.-D. Fekete, and T. Kraska. How progressive visualizations affect exploratory analysis. *TVCG*, 2016.

[114] E. Zgraggen, R. Zeleznik, and S. M. Drucker. Panoramicdata: data analysis through pen & touch. *TVCG*, 20(12):2112–2121, 2014.

[115] E. Zgraggen, Z. Zhao, R. Zeleznik, and T. Kraska. Investigating the effect of the multiple comparisons problem in visual analysis. In *CHI*, page 479. ACM, 2018.

[116] Z. Zhao, L. D. Stefani, E. Zgraggen, C. Binnig, E. Upfal, and T. Kraska. Controlling false discoveries during interactive data exploration. In *SIGMOD*, pages 527–540, 2017.

[117] Z. Zhao, E. Zgraggen, L. D. Stefani, C. Binnig, E. Upfal, and T. Kraska. Safe visual data exploration. In *SIGMOD*, pages 1671–1674, 2017.