Enabling Data Science for the Majority

Aditya Parameswaran
University of California, Berkeley
adityagp@berkeley.edu

ABSTRACT

Despite great strides in the generation, collection, and processing of data at scale, data science is still extremely inconvenient for the vast majority of the population. The driving goal of our research, over the past half decade, has been to make it easy for individuals and teams—regardless of programming or analysis expertise—manage, analyze, make sense of, and draw insights from large datasets. In this article, we reflect on a comprehensive suite of tools that we've been building to empower everyone to perform data science more efficiently and effortlessly, including DATASPREAD, a scalable spreadsheet tool that combines the benefits of spreadsheets and databases, and ZENVISAGE, a visual exploration tool that accelerates the discovery of trends or patterns. Our tools have been developed in collaboration with experts in various disciplines, including neuroscience, battery science, genomics, astrophysics, and ad analytics. We will discuss some of the key technical challenges underlying the development of these tools, and how we addressed them, drawing from ideas in multiple disciplines. In the process, we will outline a research agenda for tool development to empower everyone to tap into the hidden potential in their datasets at scale.

PVLDB Reference Format:
Aditya Parameswaran. Enabling Data Science for the Majority. *PVLDB*, 12(12): 2309 - 2322, 2019.
DOI: https://doi.org/10.14778/3352063.3352148

1. INTRODUCTION

Based on popular media, one may be tempted to think that data science—the science of extracting value from data—has been incredibly successful. In many ways, it has: it has led to some of the most exciting emergent technologies of our time, ranging from fraud detection and virtual assistants, to computer-assisted diagnosis and driverless cars. Unfortunately, the ground reality is not so rosy; data science has had a disproportionate impact. On the one hand, we have a small number of internet companies—we refer to these as "the 1%" of data science—who have been able to successfully leverage data science, with the help of skilled programmers (aka data scientists) and hardware resources. On the other hand, we have the majority: the 99% of organizations who haven't had much success—spanning manufacturing, finance, journalism, government, natural resources, basic and social sciences, and medicine.

Unfortunately, academic research and popular media have both been somewhat myopically focused on the 1%, while ignoring the needs of the 99%, precipitating a crisis. Many industry experts have written about the fact that many organizations have too much data but aren't able to make sense of it. A Forrester report in 2016 says "organizations are drowning in data and starving for insight" [3]. Hal Varian, Google's chief economist, says "an organization with data but no one to analyze it cannot take advantage of it" [68]. A McKinsey report argues that the number of data-savvy undergraduate majors needs to quadruple soon, from 10% to 40%, to keep up with demand [26]—nearly accounting for half of all majors. What we're experiencing is therefore a *data science divide*. What problems do the 99% face? Let's consider a typical example.

EXAMPLE 1. *Consider a journalist who is trying write an article on how weather in the USA has impacted flight delays. She wants to investigate whether climate change has caused flight delays to get worse. She goes to the US Bureau of Transportation website [1] and downloads the dataset of all flights in the US since 1987. There are about 500M records in this dataset (Figure 1a).*

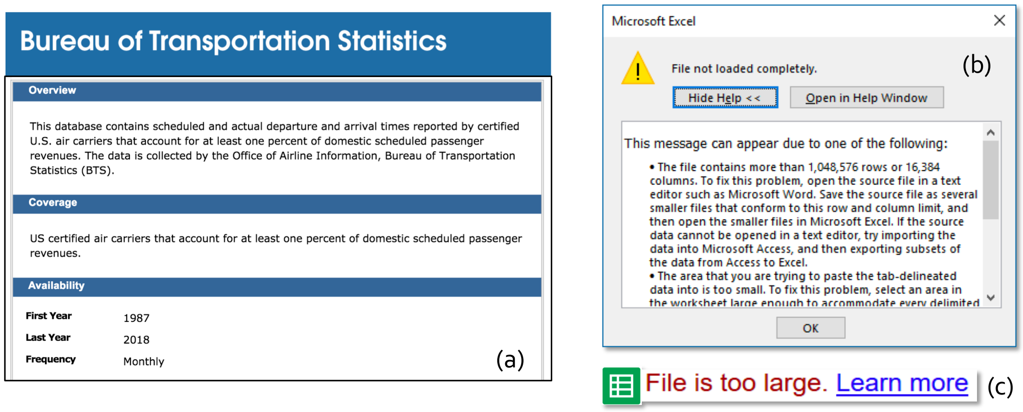
Now, in order to analyze it, she tries loading this dataset in Microsoft Excel; she gets a dialog that her file is not loaded completely (Figure 1b). She tries the same operation in Google Sheets and she gets a dialog that the file is too large (Figure 1c). So she can persist with whatever she's managed to load within Microsoft Excel, or a sample within Google Sheets, but it's not the entire dataset—so any analysis she performs may not be accurate. So, unfortunately, one can't even open and explore large datasets in tools that the 99% use.

Next, with whatever she's managed to load within Microsoft Excel, say our journalist wants to see which cities have had increasing weather delays over time. One way to do this is to plot the weather delay by year for each city, and manually inspect it to see if the "shape" of that line chart is roughly increasing. However, there are over 300 cities in this dataset. If we estimate that it takes her 1 minute per city to select, generate, and examine this plot, it would take her close to 5 hours overall. One option is to use a visualization tool like Tableau [65] to generate small multiples [66] of all of the visualizations at once—but she would still need to manually inspect hundreds of visualizations, which would take her minutes to do. If it takes her this long to answer one question on one dataset, it's going to severely limit the number of questions she can ask.

Overall, at every stage of analysis, from loading the data, to more sophisticated analysis and visualization, we have issues.

The Origin of the Issues. Why did our journalist have so many issues? There are two different, but interlinked explanations.

Proceedings of the VLDB Endowment, Vol. 12, No. 12
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3352063.3352148


Bureau of Transportation Statistics

Overview

This database contains scheduled and actual departure and arrival times reported by certified U.S. air carriers that account for at least one percent of domestic scheduled passenger revenues. The data is collected by the Office of Airline Information, Bureau of Transportation Statistics (BTS).

Coverage

US certified air carriers that account for at least one percent of domestic scheduled passenger revenues.

Availability

First Year	1987
Last Year	2018
Frequency	Monthly

(a)

Microsoft Excel

File not loaded completely.

(b)

This message can appear due to one of the following:

- The file contains more than 1,048,576 rows or 16,384 columns. To fix this problem, open the source file in a text editor such as Microsoft Word. Save the source file as several smaller files that conform to this row and column limit, and then open the smaller files in Microsoft Excel. If the source data cannot be opened in a text editor, try importing the data into Microsoft Access, and then exporting subsets of the data from Access to Excel.
- The area that you are trying to paste the tab-delineated data into is too small. To fix this problem, select an area in the worksheet large enough to accommodate every delimited

File is too large. Learn more

(c)

Figure 1: Journalist wanting to study the impact of weather on flight delays (a) downloads the dataset from the Bureau of Transportation website; (b) tries loading in Excel, receiving an error; (c) and receives a similar error from Google Sheets.

One explanation comes from the tools. In brief, there are no tools that the journalist could meaningfully use. She could use standard programming languages like Java, Python, or R, but she would need to know how to code, and how to manage the scale of data. Or, she could use relational databases, but she would need to know SQL, and it would be hard for her to translate needs like "find me cities where the weather delay is roughly increasing" to SQL queries. And as we saw earlier, interactive analysis and BI tools like Microsoft Excel, PowerBI (`http://powerbi.microsoft.com`), or Tableau don't scale too well, and make it hard to interrogate data, since the onus is on the journalist to perform manual inspection rather than be able to ask sophisticated queries.

The second explanation arises from the human in-the-loop. As dataset sizes have grown drastically, what has not grown at a comparable rate is the *human time* available for analysis, the human *cognitive load* capacities, and the *programming skills* to extract value from data. For example, the number of programmers in the US was projected to only increase 25% in a 5 year period ending in 2018 [4], while the amount of data has scaled exponentially. Unfortunately, these human-in-the-loop bottlenecks are not being addressed in present tools.

Human-In-The-Loop Data Analytics Tools. In either case, the solution is a new breed of data analytics tools that incorporates humans as a first-class citizen in data analysis, along with data. We, along with others in the community (see `hilda.io`), have been calling these types of tools *human-in-the-loop data analysis tools*, or HILDA tools. These tools *reduce human effort* and time spent, *minimize complexity*, and *reduce the need for programming* and analysis skills. As one would expect, developing such tools is challenging. The first reason is that one needs to leverage ideas and techniques from *multiple disciplines*: we need human-computer interaction (HCI), since we need to take the human perspective into account; we need data mining, since we need to go beyond SQL to more human-centric needs; and databases, because scalability and interactivity is still paramount. The second reason is that to develop HILDA tools, we need to *revisit all layers of the system stack*, from interfaces all the way to storage and indexing. Since the end-user requirements have changed, the interfaces must be changed, and these changes translate to changes down the stack, as we will demonstrate in the rest of this article. The third and final reason is that since we need to minimize human effort, complexity, and the need for programming skills, the onus is instead on the tools to do the heavy lifting. What was once manual will need to be performed automatically by the tool.

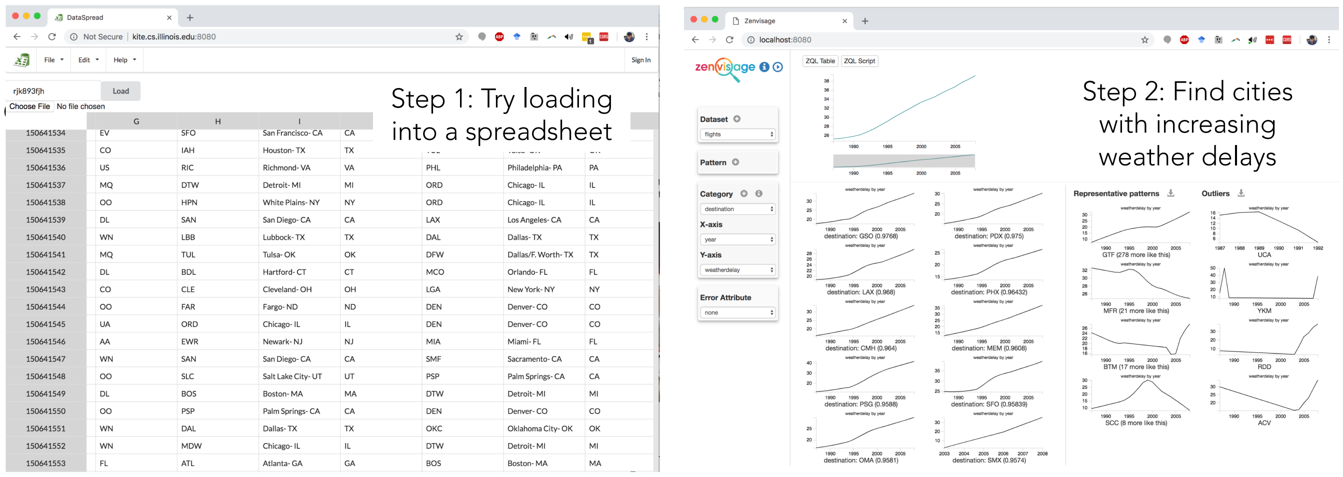
Our Research Mission and Progress. Since 2013, our research mission has been to bridge the data science divide with the goal of *enabling everyone to extract insights from large datasets, minimizing code, effort, complexity and time.* And we've been doing so by *building* HILDA *tools that make simplify, accelerate, scale-up, and manage data science activities.*

Returning back to our journalist example, we've actually built HILDA tools that address the two steps that she needed to perform. DATASPREAD is a scalable spreadsheet where she can load and explore her entire flight dataset—with 100s of millions of rows, interactively (Figure 2 left). ZENVISAGE is a scalable visual data exploration tool where she can sketch a pattern on a canvas, and instantly receive cities with increasing weather delays (Figure 2 right). We will describe these tools in Sections 2 and 3 respectively.

These two HILDA tools fit into a broader *Maslow's Hierarchy for* HILDA that we've been developing over the past five years (Figure 3). Abraham Maslow, a psychologist, hypothesized that human needs can be organized in the form of a hierarchy, with the more basic needs, such as food, water, and shelter, at the bottom, and the more advanced needs, such as fulfilling one's true potential, at the top [5]. Our Maslow's hierarchy for HILDA is analogous to Maslow's original hierarchy of needs, but is instead organized along the increasing sophistication of data analytics needs. At the bottom, we have CATAMARAN, a automatic data extraction tool that makes it possible to *touch* your data by extracting structure from messy semi-structured log files [23]; then, we have DATASPREAD, a scalable spreadsheet tool that allows you to *browse* and explore one's data, and make modifications [15]; then, we have ZENVISAGE, a visual data exploration tool to view and *play* with one's data, and see interesting insights [61]; then, we have HELIX, a human-in-the-loop machine learning tool that allows novices to iterate on machine learning models easily to gain a deeper *understanding* [77]; and finally, ORPHEUSDB, a data versioning tool that manages the data science process and facilitates *collaboration* [28].

These tools are largely field-independent and have been successfully applied in multiple fields. We've worked with neuroscientists, geneticists, battery scientists, ad analysts, among others, in ensuring that it meets their needs and use-cases.

In the rest of this article, we will describe our HILDA tools, the issues they were meant to address, the technical challenges in solving the issues, as well as an overview of solution strategies. Since this article is meant to be a retrospective on a body of work, it will necessarily be biased and incomplete; we will provide pointers to the original papers for a thorough treatment of related work. Instead, the related work discussed in this article will be work that has directly impacted our thinking. Furthermore, due to space limitations, we won't be able to cover all of the tools that we've developed. We will focus on two tools, DATASPREAD and ZENVISAGE.



rjk893fjh

	G	H	I				
150641534	EV	SFO	San Francisco- CA	CA			
150641535	CO	IAH	Houston- TX	TX			
150641536	US	RIC	Richmond- VA	VA	PHL	Philadelphia- PA	PA
150641537	MQ	DTW	Detroit- MI	MI	ORD	Chicago- IL	IL
150641538	OO	HPN	White Plains- NY	NY	ORD	Chicago- IL	IL
150641539	DL	SAN	San Diego- CA	CA	LAX	Los Angeles- CA	CA
150641540	WN	LBB	Lubbock- TX	TX	DAL	Dallas- TX	TX
150641541	MQ	TUL	Tulsa- OK	OK	DFW	Dallas/F. Worth- TX	TX
150641542	DL	BDL	Hartford- CT	CT	MCO	Orlando- FL	FL
150641543	CO	CLE	Cleveland- OH	OH	LGA	New York- NY	NY
150641544	OO	FAR	Fargo- ND	ND	DEN	Denver- CO	CO
150641545	UA	ORD	Chicago- IL	IL	DEN	Denver- CO	CO
150641546	AA	EWR	Newark- NJ	NJ	MIA	Miami- FL	FL
150641547	WN	SAN	San Diego- CA	CA	SMF	Sacramento- CA	CA
150641548	OO	SLC	Salt Lake City- UT	UT	PSP	Palm Springs- CA	CA
150641549	DL	BOS	Boston- MA	MA	DTW	Detroit- MI	MI
150641550	OO	PSP	Palm Springs- CA	CA	DEN	Denver- CO	CO
150641551	WN	DAL	Dallas- TX	TX	OKC	Oklahoma City- OK	OK
150641552	WN	MDW	Chicago- IL	IL	DTW	Detroit- MI	MI
150641553	FL	ATL	Atlanta- GA	GA	BOS	Boston- MA	MA

Dataset
flights
Pattern
Category
destination
X-axis
year
Y-axis
weatherdelay
Error Attribute
none

destination: GSO (0.9768)
destination: PDX (0.975)
destination: LAX (0.968)
destination: PHX (0.96432)
destination: CMH (0.964)
destination: MEM (0.9608)
destination: PSG (0.9588)
destination: SFO (0.95839)
destination: OMA (0.9581)
destination: SMX (0.9574)

Figure 2: Instead, the journalist can use DATASPREAD to explore her dataset, with over 150M rows (left), and can sketch an increasing pattern on ZENVISAGE to find matches below for cities with increasing weather delays (right).

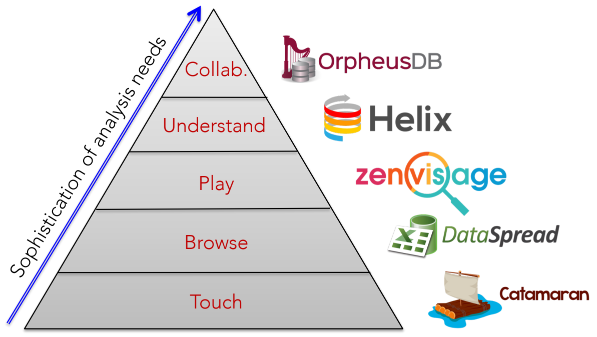


Figure 3: A Maslow's Hierarchy for HILDA.

2. ENABLING DATA MANIPULATION

If one has a goal of enabling data science for the majority, one must start with spreadsheets. Spreadsheets are arguably the most popular tool for managing and analyzing data on the planet, vastly outstripping most relational databases. Recent estimates from Microsoft posit around 750M users of Microsoft Excel, around 10% of the world's population [46]. In contrast, the number of programmers is less than 20M [4].

2.1 Spreadsheet Issues and our Vision

To better understand the ways spreadsheets are used in practice, and the issues that people have when using spreadsheets, we turned to a time-tested approach from HCI: user surveys [48, 73]. In our CHI'2018 paper [43], we analyzed a number of posts on the Excel forum on Reddit—a popular Q&A and discussion website (`reddit.com`). We found that spreadsheets are used by individuals in literally every sector, ranging from professional ones, like stock tracking, finance, inventory tracking, real estate and manufacturing, scientific data tracking, accounting, and patient data recording, as well as personal ones, like quantified self, fantasy football and other sports tracking, as well as personal finance management.

2.1.1 *Spreadsheet Popularity*

Why are spreadsheets so popular? Work from the HCI community has identified several reasons for why this may be the case. Shneiderman attributed the popularity of spreadsheets to its *direct manipulation* capabilities [60]. Nardi and Miller performed an ethnographic study of spreadsheet usage, and determined that the usefulness of spreadsheets stemmed from an intuitive "table-oriented layout", as well as "computation without having to write code" that sits alongside the data [51]. A subsequent paper by the same authors claimed that within organizations, spreadsheets are exceedingly useful as a means to share knowledge and facilitate collaborative work [50]. As the saying goes, in most business applications, the "export to excel" button is the third-most commonly used button from the menu bar, after open and cancel [11].

2.1.2 *Spreadsheet Issues*

At the same time, spreadsheets are not scalable or interactive. Spreadsheets impose limits on scalability: as we saw in our journalist example, one cannot even open a spreadsheet with more than 1 million rows in Excel or Google Sheets. Moreover, spreadsheets are not interactive; most computation takes far too long to complete. As an example, in Figure 4, we plot the time it takes to do a VLOOKUP from one array to another on Microsoft Excel—a standard operation in spreadsheets [7], essentially a join of two arrays. On the x axis, we plot the size of the arrays, e.g., for size 20,000, there are 20,000 VLOOKUP formulae, each into an array of size 20,000. (So, this would be a foreign-key join of two tables with 20,000 rows each.) Even at the 40,000 mark, the time taken is 1/3rd of a minute, well above what it would take to do similar join in a relational database, which would take less than a second (indicated by the blue arrow). This is despite the fact that the spreadsheet data is entirely stored within main memory. It was shocking to us that spreadsheets, despite decades of development, suffer from such poor performance.

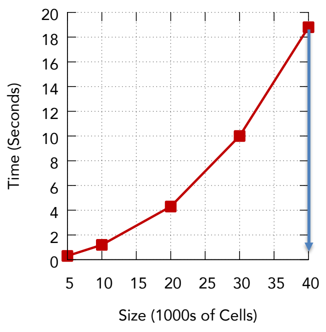


Figure 4: VLOOKUP scales to minutes on 100s of thousands of rows, while a regular join on disk-resident data would take less than a second (indicated by the blue arrow).

There were several instances of scalability and interactivity issues that we discovered in our Reddit survey [43]. We've paraphrased some examples below.

Like our journalist, several users had issues with **organization** where they wanted to operate on larger datasets and had trouble importing it into spreadsheets, or expressed concerns with the size of their spreadsheet and wanted to shrink it down:

- "*I am trying to import a file with 2 million rows, but the entire file is not imported. Is there a way to get the entire data in?*"
- "*I inherited a slow spreadsheet of about 50MB in my new job, and I can tell it's going to crash. How do I shrink it down?*"

Several users had issues in spreadsheet **manipulation**, when adding or deleting rows and columns:

- "*When my large spreadsheet started acting up, I tried to delete a few rows, and got an error message 'Excel cannot complete this task with available resources.'*"
- "*Can someone help me understand how a 150MB Excel file can take forever to delete rows/columns, even with 4GB RAM?*"

A number of users had trouble with spreadsheet **computation**:

- "*I use a spreadsheet to track my entire life. I cull data every year to keep it manageable. Excel is locking up during basic calculations.*"
- "*I am not a power user: I use formulae on a 500k row, 20 col. spreadsheet. Why does the beach ball*[1] *come up so often?*"

Overall, as described, spreadsheet users had issues with organizing data, manipulating data, and doing computation atop data.

So what answers did other users on the Reddit forum have for these concerns? The vast majority recommended using a database (45%)—one that should bring our community great joy! Others recommended turning off recalculations (15%); or storing in a compressed format (15%), among other solutions [43]. Nearly every solution apart from that of using databases was a temporary workaround, as opposed to a permanent solution.

2.1.3 DataSpread

While we can rejoice given that databases were suggested as an alternative to spreadsheets, they are not easy to use or flexible. Indeed, as the author himself can attest, while teaching a large undergraduate databases class, we use spreadsheets for managing grades, despite professing our undying love for databases. Moreover, our PhD students often use spreadsheets to manage experimental data, when coincidentally developing and experimenting on databases.

So, our guiding question was: can we build a tool that combines the ease of use and flexibility of spreadsheets, with the scalability and interactivity of databases? This was the goal of our tool, DataSpread. We wanted DataSpread to have a spreadsheet-like frontend and a relational database backend [12,15]. This solves dual problems of providing a scalable and interactive backend for spreadsheets, and an intuitive and flexible frontend for databases.

Unfortunately, building DataSpread is extremely challenging, given the fundamental differences between spreadsheets and databases—see Figure 5. As an example, databases opt for a rigid, unordered structure, while spreadsheets support an ad-hoc, flexible structure, with order as a first-class citizen.

In the next three sections, we will describe how we addressed these challenges in supporting scalable spreadsheet organization, manipulation, and computation.

2.2 Spreadsheet Organization

[1]Beach balls are spinning wheel icons that appear when MacOS computers hang.

	Aspect	Databases	Spreadsheets
I. Organization	structure	rigid	flexible
	presentation	unordered, uniform	ordered, ad-hoc
II. Manipulation	modality	relation at a time	cells, using position
	granularity	predicate-based	direct edits + add/delete row/columns
III. Computation	modality	external	in-situ, with data
	granularity	queries on relations	formulae on cells

Figure 5: Comparison between Spreadsheets and Databases.

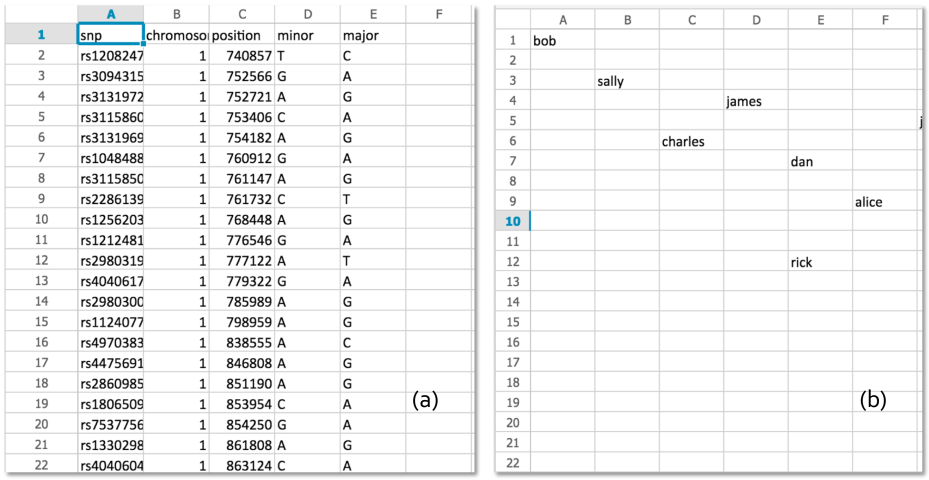
The first and most fundamental question is: *how do we organize or represent large volumes of spreadsheet data in a backend database*? This is hugely important since it impacts the scalability as well as interactivity of the spreadsheet, as we will show.

Spreadsheets vary a lot in terms of structure, from very dense (Figure 6 left) to very sparse (Figure 6 right). If we had a dense spreadsheet, it is straightforward to represent it in a relational database, as a *table* with attributes: [row #, column A, column B, . . .]. However, this representation is wasteful for a sparse spreadsheet since our table will be filled in largely with null values and possibly unnecessary columns. If we had a sparse spreadsheet, it is straightforward to represent all of the filled-in cells in a relational database as *triples* with attributes [row #, column #, value]—this is essentially a key-value type representation with the row # and column # capturing the positional key. However, this representation is wasteful for a dense spreadsheet, since we will need to add a new tuple for each filled-in cell (with its associated overhead), and store the row and column number for each cell.

In practice, even within a single spreadsheet, the density can vary widely. Nardi and Miller reported that *"spreadsheets are visually modularized to show their sub-parts: users segment using criteria such as years, months, regions, companies, and departments"* [51]. So there are often many tabular regions within a spreadsheet, embedded along with formulae.

Our approach, therefore, is to opt for a *hybrid* representation, where we combine the benefits of tables and triples (i.e., the dense and sparse approaches outlined above). We carve out dense areas and store each one as a separate table, while the remaining sparse areas are represented in a single triples table. Thus, the hybrid representation relies on a decomposition or partitioning of the spreadsheet into regions. Figure 7 displays one such decomposition, where the spreadsheet is decomposed into red and blue areas; each red area is stored as a separate table, while the blue areas are stored together in a single triples table. Unfortunately, we determined that identifying this optimal decomposition is NP-Hard via a reduction from the minimum edge-length partitioning of rectilinear polygons [13]. So, instead, we opt for a restricted space of decompositions called *recursive decompositions* that are formed by recursively decomposing areas of the spreadsheet (an analogy is to use scissors to cut a sheet of paper with vertical or horizontal cuts that cut the remaining region entirely). It turns out that identifying the best recursive decomposition is in PTIME via dynamic programming [13]. While this is promising, the complexity is still $O(n^5)$. We employ two tricks to bring the effective complexity down. First, we collapse down rows and columns that have the same signature of filled cells to give a smaller representation that uses weighted row/columns, reducing the effective n for the algorithm. Second, we can apply greedy decomposition instead of dynamic programming. Both these tricks give us a solution that is close to the overall optimal decomposition [13]. Our results show that the recursive-decomposition-based hybrid representation gives us a 2-3× reduction in storage or time to perform formula computation relative to a single table or triples-only representation [13].

Moreover, we get another benefit for free: representing the data in a backend database allows us to scroll to arbitrary locations on



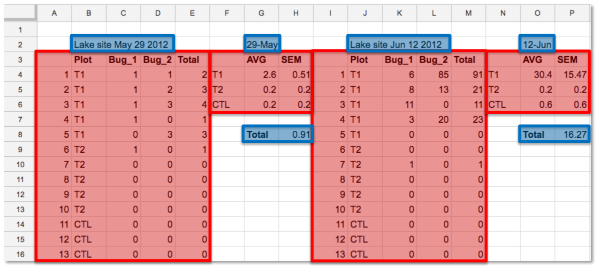
	A	B	C	D	E	F
1	snp	chromoso	position	minor	major	
2	rs1208247	1	740857	T	C	
3	rs3094315	1	752566	G	A	
4	rs3131972	1	752721	A	G	
5	rs3115860	1	753406	C	A	
6	rs3131969	1	754182	A	G	
7	rs1048488	1	760912	G	A	
8	rs3115850	1	761147	A	G	
9	rs2286139	1	761732	C	T	
10	rs1256203	1	768448	A	G	
11	rs1212481	1	776546	G	A	
12	rs2980319	1	777122	A	T	
13	rs4040617	1	779322	G	A	
14	rs2980300	1	785989	A	G	
15	rs1124077	1	798959	A	G	
16	rs4970383	1	838555	A	C	
17	rs4475691	1	846808	A	G	
18	rs2860985	1	851190	A	G	
19	rs1806509	1	853954	C	A	
20	rs7537756	1	854250	G	A	
21	rs1330298	1	861808	A	G	
22	rs4040604	1	863124	C	A	

(a)

	A	B	C	D	E	F
1	bob					
2						
3		sally				
4				james		
5						j
6			charles			
7					dan	
8						
9						alice
10						
11						
12					rick	
13						
14						
15						
16						
17						
18						
19						
20						
21						
22						

(b)

Figure 6: Spreadsheets can vary widely in density from very dense (right) to very sparse (left).



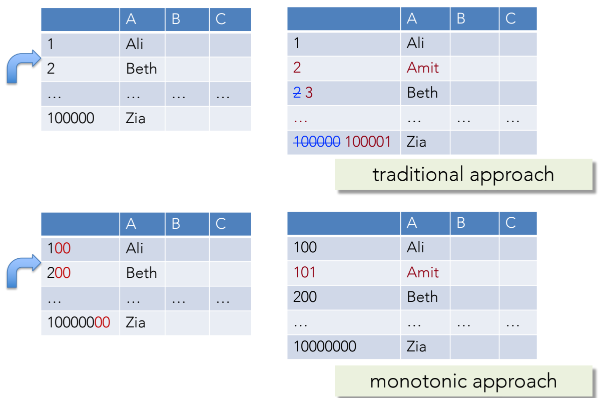
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1																
2		Lake site May 29 2012					29-May			Lake site Jun 12 2012					12-Jun	
3		Plot	Bug_1	Bug_2	Total		AVG	SEM		Plot	Bug_1	Bug_2	Total		AVG	SEM
4	1	T1	1	1	2	T1	2.6	0.51	1	T1	6	85	91	T1	30.4	15.47
5	2	T1	1	2	3	T2	0.2	0.2	2	T1	8	13	21	T2	0.2	0.2
6	3	T1	1	3	4	CTL	0.2	0.2	3	T1	11	0	11	CTL	0.6	0.6
7	4	T1	1	0	1				4	T1	3	20	23			
8	5	T1	0	3	3		Total	0.91	5	T1	0	0	0		Total	16.27
9	6	T2	1	0	1				6	T2	0	0	0			
10	7	T2	0	0	0				7	T2	1	0	1			
11	8	T2	0	0	0				8	T2	0	0	0			
12	9	T2	0	0	0				9	T2	0	0	0			
13	10	T2	0	0	0				10	T2	0	0	0			
14	11	CTL	0	0	0				11	CTL	0	0	0			
15	12	CTL	0	0	0				12	CTL	0	0	0			
16	13	CTL	0	0	0				13	CTL	0	0	0			

Figure 7: A hybrid decomposition example: the regions in red can be represented as tables, while the regions in blue can be represented as triples.

arbitrarily large datasets instantaneously (Figure 2 left). There are two aspects that make this feasible. First, paging of data from disk allows us to retrieve pages (rows) from a specific location on-demand within the spreadsheet. Disk speeds have progressed to the point where one can do a random access to a specific location within interactive time-scales. Second, a positional index structure allows us to identify the page(s) on disk that correspond to a specific row. We describe this index structure next.

2.3 Spreadsheet Manipulation

The next question that we need to tackle is *how do we support efficient positional data manipulation?* Spreadsheets operate on data that is ordered, and adding (deleting) rows (columns) at a specific location is a fundamental operation that happens often on spreadsheets. Furthermore, formulae refer to data by position, so it is crucial to ensure that formula accesses based on position stay interactive, even at scale and while edits are made to data or formulae.



	A	B	C
1	Ali		
2	Beth		
…	…	…	…
100000	Zia		

	A	B	C
1	Ali		
2	Amit		
~~2~~ 3	Beth		
…	…	…	…
~~100000~~ 100001	Zia		

traditional approach

	A	B	C
100	Ali		
200	Beth		
…	…	…	…
10000000	Zia		

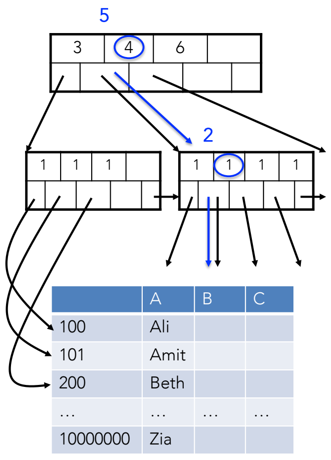
	A	B	C
100	Ali		
101	Amit		
200	Beth		
…	…	…	…
10000000	Zia		

monotonic approach

Figure 8: Two standard approaches for recording position.

Unfortunately, the *traditional* approach to recording position is problematic. Say we simply record the position as a separate attribute in the table approach from the previous section. (Similar considerations apply even for the triples or hybrid representation.) Adding a row early, e.g., adding Amit between Ali and Beth in Figure 8 top, can cause all subsequent row numbers to be incremented, thus turning an $O(1)$ operation to an $O(n)$ one, where nearly every single row of the table is edited. We call this a *cascading update*. This cascading update is the reason we had spreadsheet users complaining about issues such as "why does it take forever to delete rows?" (Section 2.1.2). On the other hand, locating the kth record is easy via a traditional index (such as a B+tree) on the row number.

Another approach is what we refer to as the *monotonic* approach. We don't store the row numbers directly, but instead store monotonically increasing proxies. For example, instead of storing 1, 2, . . ., we store 100, 200, . . ., where there is a one-to-one mapping between the proxies and the original row numbers based on ordering, as shown in Figure 8 bottom. Here, we can easily add a new row between row 1 and row 2—in fact, we can effectively add up to 99 rows—with no cascading updates. However, locating the kth record is not so easy with this approach, requiring a sort of the monotonically increasing proxies, effectively an $O(n \log n)$ operation—this is because the mapping between row numbers and the proxies are lost. (Note that a traditional B+tree will not work in this case, since it will locate data by the value of the proxy as opposed to the actual row number, the latter of which is needed.)



	A	B	C
100	Ali		
101	Amit		
200	Beth		
…	…	…	…
10000000	Zia		

Figure 9: The counted approach.

So, in DATASPREAD, we adopt a hybrid approach [13]. We represent data as in the monotonic approach, with monotonically increasing proxies, but to also efficiently locate data by position, we employ a so-called *counted B+Tree*—a classical index structure that has never been used (to our knowledge) in a systems context [2]. This index structure stores the number of records for each sub-tree. In the example displayed, to retrieve the 5th record, we would walk down the second path (since $3 < 5 \leq (3+4)$), following which we would walk down the second path (since $1 < 2 \leq (1+1)$), and thereby locate the desired record. This approach offers a lookup by location in $O(\log n)$, with additions and deletions also in $O(\log n)$, all by traversing the tree and updating the counts along the path. As expected, the hybrid approach does extremely well in practice, matching the performance of the monotonic approach for updates, and the performance of the traditional approach for locating data by position [13].

2.4 Spreadsheet Computation

The third question we tackled for DATASPREAD was *how do we support efficient in-situ formula computation?* Formulae are extensively embedded within spreadsheets; as we described in Section 2.1.1, one of the chief reasons why spreadsheets are so popular is the fact that they support computation that sits alongside data.

Unfortunately, complex and/or large spreadsheets often become very sluggish, sometimes hanging for minutes when a change is made. This is because traditional spreadsheets opt for a *synchronous computation model*, wherein consistency is emphasized. That is, as soon as a change is made, all of the dependent cells are computed, and then and only then is control returned to the user. Thus, the spreadsheet opts to always show a consistent view to the user. However, the spreadsheet remains entirely inaccessible when the computation is being performed. If we consider the cells that are unavailable or inaccessible to the user over time as in Figure 10, then, with the synchronous model (red curve), as soon as a change is made, the entire sheet is unavailable until the computation is complete; in the figure, all 10,000 spreadsheet cells are unavailable, until the computation is complete at the 5 second mark.

Recognizing the loss of interactivity, spreadsheets also offer an alternative—a manual computation mode, wherein no formulae are computed until the user clicks the "compute now" button. Here, the spreadsheet opts for interactivity, at the cost of consistency. (However, when the user clicks the "compute now" button, the spreadsheet is once again inaccessible during computation.) Thus, the two options provided by present-day spreadsheets either opt for interactivity at the cost of consistency, or vice versa.

Instead, we opt for an *asynchronous computation model*, aiming to allow for interactivity as well as consistency. When changes are made to the spreadsheet, instead of the spreadsheet hanging entirely, we instead blur out the cells that are dependent on the change, replacing them with a progress bar, and computing them asynchronously in the background. The rest of the cells that are not dependent on the change are returned to the user, allowing the user to continue to explore the spreadsheet. We show an example in Figure 11, where formulae for average departure and arrival delays are being computed for the journalist in Example 1—displayed as progress bars, while the journalist can continue to explore the rest of the spreadsheet. Thus there is no perceived interactivity loss, except for the cells that are anyway dependent on the change and therefore have to be recomputed. If we consider Figure 10, the asynchronous computation model (blue curve) has two phases, one, wherein the cells that are dependent on the computation are identified and marked as such, following which control is returned to the user (labeled as 1), and two, wherein computation is scheduled for the cells that are to be recomputed (labeled as 2). If we measure the benefit of the asynchronous computation model relative to the synchronous one—we can do so by computing the area under the curve of unavailable cells over time, a metric we term *unavailability*—then the asynchronous computation model has a much lower unavailability. To reduce the total unavailability, we must try to limit the area under the curve of unavailable cells for each of the two phases. Unfortunately, doing so for either phase is challenging, as we will describe next.

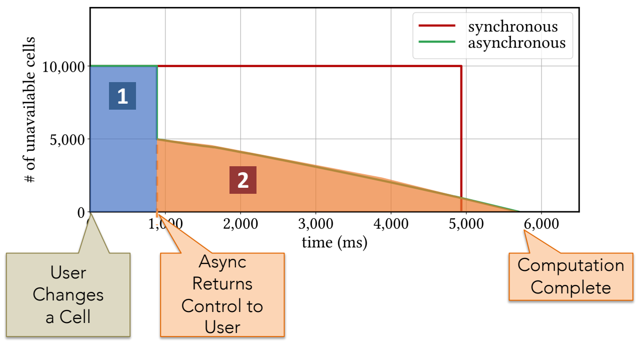
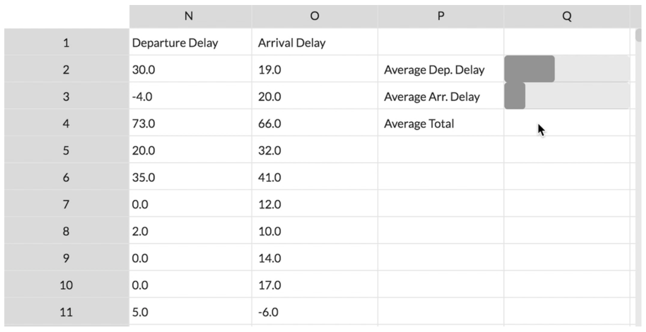


Figure 10: Chart of unavailability over time for synchronous and asynchronous schemes. For synchronous, the entire spreadsheet stays unavailable until computation is complete. For asynchronous, control of most of the sheet, except the cells dependent on the change, is returned to the user, with computation happening in the background.



	N	O	P	Q
1	Departure Delay	Arrival Delay		
2	30.0	19.0	Average Dep. Delay	
3	-4.0	20.0	Average Arr. Delay	
4	73.0	66.0	Average Total	
5	20.0	32.0		
6	35.0	41.0		
7	0.0	12.0		
8	2.0	10.0		
9	0.0	14.0		
10	0.0	17.0		
11	5.0	-6.0		

Figure 11: Asynchronous Computation Example.

Consider the first phase, of identifying dependent cells, replacing them with a progress bar, and returning control to the user. Here, identifying dependents, implemented naively, could involve traversing the graph of dependencies, which could be $O(n)$ in the worst case, negating the benefits of asynchronous computation. One approach that doesn't have the same limitation is to precompute the dependents of each cell, and store that instead for easy access. Unfortunately, even the number of dependents of a cell can be $O(n)$. What we opt to do in DATASPREAD is to instead compress the dependents of a cell using ideas similar to Section 2.2. That is, we try to represent all the dependents of a cell using rectangular regions, described only by the coordinates of the top left and bottom right corners. This can lead to a much more compact representation of a collection of cells as rectangles. At the same time, there could be false positives (cells that are deemed to be dependent but actually are not)—this is fine because false positives only lead to redundant computation, and do not impact correctness. We cannot have false negatives, however, as that would lead to loss of consistency. Identifying the optimal grouping of cells into rectangles is NP-HARD; however, greedy grouping strategies work quite well [14].

The second phase involves computing dependent cells. There are many orders in which these dependent cells can be computed, prioritizing for what is easiest to compute, what other dependent cells are dependent on, or what is currently in the user window, among other objectives. Unfortunately, finding the best computation order is NP-HARD, and even examining all of the cells to find a "good" computation order can negate the benefits of asynchronous computation [14]. Instead, we opt for an simple "on-the-fly" scheduling approach where we pick a few cells at a time, and compute the lowest cost one among them, which does surprisingly well in practice.

Overall, these two approaches, when combined, lead to a substantial reduction in unavailability, leading to a much more seamless and interactive experience for the user, while not sacrificing consistency.

2.5 Next Steps and Selected Related Work

While we've made substantial progress in making it easy to organize, manipulate, and perform computation within large spreadsheets that go beyond main-memory limitations, we're still far from our original goal of combining spreadsheets and databases to provide intuitive and flexible exploration, while not sacrificing interactivity and scalability. There are several directions we are exploring currently, including:

Spreadsheet Querying. To have DATASPREAD function as a frontend for databases, we need to be able to support more expressive queries from the frontend. DATASPREAD can now support relational algebra and SQL queries in conjunction with formulae, all within the frontend [15]. We can also import tables from the backend database and display it on the sheet [12], with the table and the spreadsheet view kept in sync. This required addressing challenges in ensuring that the relational query results do not spill-over and overwrite other data on the spreadsheet, among others.

Spreadsheet Navigation. When dealing with a very large spreadsheet, say, one with a billion rows, even scrolling to get to a desired location can be rather cumbersome. In fact, getting a high-level view of the spreadsheet, so that one can determine where to scroll is itself challenging. We've been developing NOAH, a hierarchical spreadsheet exploration interface that allows one to "zoom-in" and "zoom-out" of the spreadsheet with a few mouse clicks [55].

Spreadsheet Versioning. One of the most problematic aspects about spreadsheets is the preponderance of errors; this fact has been extensively written about and documented [52]. We're developing techniques to make it easy to maintain spreadsheet history and be able to identify the source of spreadsheet errors, leveraging our work on ORPHEUSDB [28].

During our development of DATASPREAD so far, we were inspired by work by Liu and Jagadish on developing a spreadsheet algebra [41], as well as work from the same group on database usability [18, 30]. ABC, a project from the 90's from UC Berkeley [56, 57, 58], was a huge influence in DATASPREAD development, and shared a similar vision. Many others have been exploring ways to increase the expressiveness of spreadsheets [11, 67], and making it easier to express queries [8, 29, 49]—a space that is closely related to DATASPREAD. We've also been excited to see progress from industry on similar projects, including Airtable (`airtable.com`) and Hillview [17].

3. ENABLING DATA VISUALIZATION

Once we have the ability to examine a dataset in a tool like DATASPREAD or Microsoft Excel, often the next step is visual data exploration. While spreadsheet tools certainly support visualization, visual analytics tools like Tableau [65] treat visualization as more of a first-class citizen. During visual data exploration using a tool like Tableau or Excel, the user often starts with some desired pattern or hypothesis. Like for our journalist, she wanted to find cities with increasing weather delays over time. And the current approach in such a tool is to generate a specific visualization, for some city, see if it matches the desired pattern of increasingness, and if it doesn't, then repeat until she finds those where the weather delays are increasing. Manual generation and examination of a collection of visualizations to test for patterns, hypotheses, or insights in this manner is extremely laborious and time-consuming. There are two reasons for this: first, that there are too many visualizations to examine—e.g., in our weather example, there were over 300 visualizations, and second, that on very large datasets, even the generation of a single visualization can take a very long time.

3.1 Visualization Issues and Our Vision

Pattern search of the form described previously appears in virtually every discipline. Here are three groups of researchers with similar issues that we have been working closely with:

Genomics researchers at the KnowEnG genomics center at UIUC and Mayo Clinic [64] often need to find gene properties that explain the results of gene expression experiments, visualized as a two-dimensional scatterplot. Some of the genes are positively expressed, while the others are negatively expressed. The goal is to find scatterplots (where the X and Y axes are gene properties) where the positive genes are well-separated from the negative ones. To do this, they try a range of properties as X and Y axes, until they find ones where the separation holds.

Astronomers at the Dark Energy Survey [21] want to find astronomical objects with specific shapes for their light curves (a line chart of light intensity over time). One such shape that they are particularly interested in are astronomical objects that have a sharp dip and then slow rise in their light curve. This is indicative of a star with a planet surrounding it, reminiscent of the sun-earth relationship—and may be valuable for future human colonization once we have thoroughly decimated our planet. At present, the astronomers spend a vast amount of time going through light curves of 1000s of astronomical objects manually until they find those that match the specification.

Battery Scientists at Carnegie Mellon University aiming to develop lightweight batteries for electric vehicles and aircraft [71] can now easily generate vast sums of simulation data on solvent physical properties capturing aspects such as potential, energy, heat, and stability. Often they are looking for solvent classes that have specific relationships between two physical variables—but their current approach is to ignore the data entirely ("let's not bother; it is too painful to explore!") and perform trial-and-error-based physical testing. This results in wasted costs and resources, all because the data is too cumbersome to explore.

In all three cases, there is a need to find specific visual patterns to test hypotheses and derive insights, but it is too cumbersome and time-consuming to do so. They could write code to do this, and some of them do (e.g., the astronomers do write code in Python for data preprocessing), but it is no less cumbersome for them.

So the question that we sought to answer was: instead of combing through these visualizations manually, can we develop systems and techniques that accept as input desired visual patterns, and can accelerate the search for these patterns? This is the goal of our visual data exploration system, ZENVISAGE. Our first step towards that goal was to simply accelerate the generation of a specific visualization (Section 3.2), followed by searching across a collection of visualizations (Section 3.3).

3.2 Visualization Generation

The first question we addressed is *how do we accelerate the generation of a single visualization?* Fortunately, most visualizations can be expressed as a SQL query of the following form:

```
SELECT X1, X2, ..., F1(Y1), F2(Y2), ...
FROM R WHERE <PRED>
GROUP BY X1, X2, ...
```

For example, the line chart of weather delays by time would have a single X1 corresponding to month, and a single F(Y1) corresponding to average(weather_delay). Multiple Xs or Ys correspond to multiple dimensions being displayed on the same visualization: for example, a heatmap may have X1 and X2 corresponding to the vertical and horizontal axes, while F1(Y1) will correspond to the intensity, possibly depicted using color hues. We focus on a single relation for simplicity; most visual analytics tools assume a denormalized setup as well.

Now that we've mapped a visualization to a SQL query, we can apply standard tricks to speed up SQL query execution. For example, we can leverage materialization, but the space of visualizations that can be generated over the course of ad-hoc data exploration is far too large for materialization to be applicable—when one considers arbitrary combinations of predicates to select the data that is then visualized. This is the same "curse of dimensionality" that pops up when trying to materialize data cubes beyond a small number of dimensions [39].

Another approach is to leverage *approximation*—generate the visualization such that it "looks" like the visualization computed on the entire dataset, while in fact it is computed on a sample. If the approximate visualization "looks" like the actual visualization, then users can make decisions using the approximate one without waiting for the visualization on the entire dataset to be generated. Usually, these decisions rely on the "big picture", and inaccurate values are often not a barrier to decision-making.

This problem is similar to canonical approximate query processing (AQP) [24], except that there are two key differences. First, the objectives are often not the same: for example, in AQP, the error on each aggregate is the chief measure of interest, while here, even if we get individual aggregates wrong, it can still lead to a useful visualization. In particular, if users are interested in the overall trend, even if individual aggregates are wrong, when compared with other aggregates, if it still leads to a similar looking visualization (in that the bars or trend is similar, or the color hue is similar), this would be adequate to draw correct conclusions. Thus, there are perceptual aspects that can inform our approximation schemes. Second, since visual data exploration is an ad-hoc process, it is impossible to expect a workload that we can use to generate precomputed samples up front, unlike traditional sampling engines [9, 10].

We have addressed this problem from two different perspectives. These two perspectives take advantage of an online sampling (as opposed to offline, like traditional AQP) engine that we developed, called NEEDLETAIL; NEEDLETAIL uses compressed bitmaps to select tuples to read on-the-fly that satisfy arbitrary combinations of predicates appearing during ad-hoc visual data exploration [34].

3.2.1 *Accelerating Comparisons*

When making decisions and drawing insights, users care more about the "big picture"—the trends and comparisons—as opposed to actual values. For example, consider generating a bar chart of average delay by airline on flight dataset, like our journalist—see Figure 12 left. The fact that the average delay of UA is greater than that of SW is usually more important than getting the exact average delay of UA. So, the question is: *how can we use online sampling techniques to guarantee the correct pairwise ordering of bars within a bar chart?* Ordering the bars correctly allows users to draw correct comparisons across them.

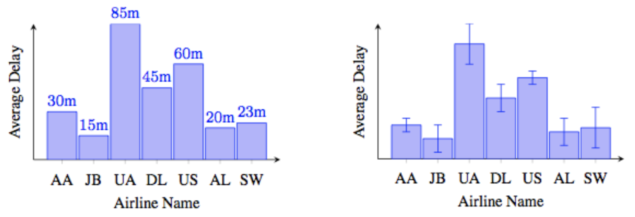


Figure 12: Bar chart visualization of the average delay by airline on the entire dataset (left); with confidence intervals after some samples have been taken (right).

Say we had already taken some samples leading to the bars (with confidence intervals) as shown in Figure 12 right. Should we sample more from UA (which has a larger confidence interval, but this interval overlaps less with others), or from AA (which has a smaller confidence interval, but this interval overlaps more with others)?

Turns out the best approach is to, at each round, continue taking samples from all groups or bars (i.e., airlines) that are still *active*; an active group is one whose confidence interval still overlaps with others. In Figure 12 right, all but UA and US's confidence intervals are still active. This simple sampling algorithm is guaranteed to be *optimal* in that it takes no more samples than is necessary in the worst case [33]. A further improvement to this algorithm only refines confidence intervals to the point where they are visually perceptible—beyond which users cannot tell the difference on the screen. The algorithm is also several orders of magnitude faster on real-world datasets compared to other algorithms that provide comparable output guarantees [33].

3.2.2 *Accelerating Termination*

The previous approach provides a promising approach for seeing the visualization early by changing the output guarantee. However, when the dataset is very large, users often want to see the visualization as it is being generated, as opposed to waiting for a long time. So the question is *how can we use online sampling to facilitate the generation of visualizations that incrementally improve over time?* One approach is adopt online aggregation [25] and display the visualization as samples are drawn, as in Figure 13—as samples are drawn over time (from left to right), the trend line (top) or heatmap (bottom) varies rapidly and then eventually stabilizes. Unfortunately, at intermediate steps, it is hard to draw conclusions with confidence.

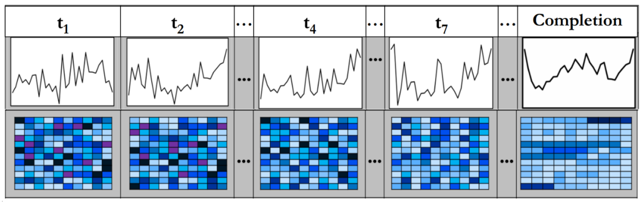


Figure 13: Online aggregation approach: The visualization varies rapidly and then eventually stabilizes.

Instead, we target the generation of visualizations *incrementally*, adding in the most important features within the visualization before the less important ones [54]. For example, for a trend line (Figure 14 top), we would first approximate it with a flat line, then split that into two, and so on and so forth—at the kth iteration, we would show the user the best k-segment approximation of the trend line. Similarly, for a heatmap, we would repeatedly divide a rectangle within the heatmap into four quadrants based on which division

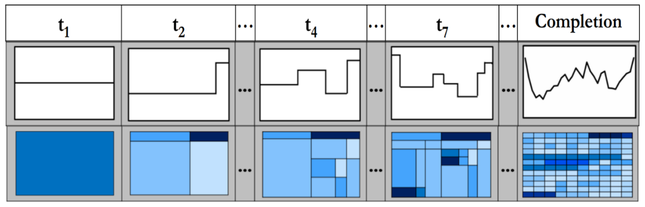


Figure 14: Incremental approach: One refinement is shown to the user at each round, prioritizing the important features of the visualization over others

would be most informative. For example, at time t_7 in Figure 14 bottom, we can already tell that the areas with high intensity are at the top right and bottom left corners, as well as the middle areas[2].

Given the new interface, one natural question is whether users are able to interpret and make use of these approximations. Via user studies, we found that our incremental approach led to double the decision-making accuracy of a standard online-aggregation-like scheme that displays all of the features of the visualization as they are being generated (as in Figure 13) [54]. This was surprising to us, since the incremental approach knowingly obscures the freshest estimates of each aggregate, in favor of returning refinements that are guaranteed to be the best possible one at that point. We also showed that the incremental approach is able to generate visualizations that preserve all of the important features often 50× faster than other approaches [54].

3.3 Visualization Search

Freshly armed with techniques to accelerate the generation of visualizations, we now return to the original question of *how do we accelerate the search of visual patterns across a collection?* In developing ZENVISAGE, our solution for this question, there were three aspects we had to cover. First, *expressiveness*: does the tool generalize to the spectrum of use-cases and expertise? Second, *usability*: is the the tool usable and useful? Third, *scalability*: does it scale to large datasets? We start by discussing expressiveness.

3.3.1 Expressiveness

Our starting point for ZENVISAGE was a sketching interface to search for patterns. Our interface looks like Figure 15, and was developed via an elaborate process of user-centered design with the participation of stakeholders that will be described in Section 3.3.2 [37]. We display ZENVISAGE operating on a real-estate dataset. We've selected the attributed soldpricepersqft as the Y axis, the month as the X axis, and the category (or Z axis) as city: that is, we're exploring a collection of visualizations of soldpricepersqft by month, one for each city. Immediately, the system provides representative patterns (k) and outliers (l) on the panel on the right. In particular, Port Orchard's pattern of rapidly increasing and decreasing, followed by a slow rise is a common pattern, with 62 other cities sharing the same pattern, while Kendall is an example of an outlier that contains a spike. Then, the central canvas panel allows users to draw a desired pattern (a) with results showing up below (j)—the best matches for the sketched pattern are Egg Harbor Township and Brick. Users can also drag-and-drop a visualization onto the canvas from the results panel (j) or the representative or outlier panels (k, l) to seed the sketched pattern. Finally, the user can input a pattern via data (i) or provide an input equation (b) to seed the search. The user can also adjust the match criteria, including the similarity function (f), the smoothing prior to the match (c), the subset of the x axis to be considered for the match (d), whether the x axis location is important at all (e), or a filter to select the subset of data for all of the visualizations (g). The user can also define their own categories to compose visualizations (h).

[2]This has the added benefit of looking like many of Mondrian's paintings [6].

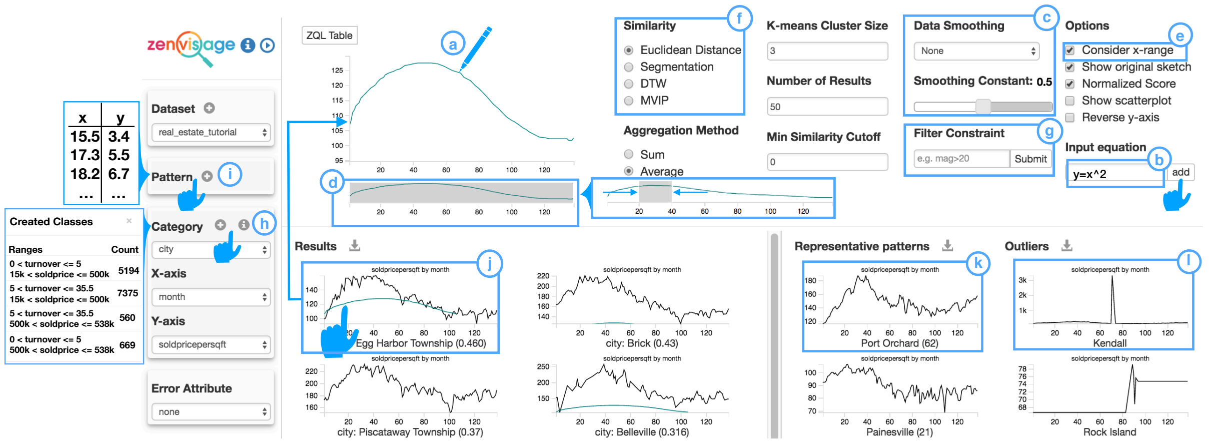
Overall, this interface satisfies simple pattern search needs via sketching and drag-and drop, and provides context via representative and outlier patterns. However, there are many needs that go beyond one step. For example, one may want to search for multiple patterns simultaneously, e.g., find cities where both foreclosures and sales prices are increasing. Or, one may want to pivot and look at the data from a different perspective, e.g., for cities where the foreclosures are similar to Berkeley, we may want to find typical trends for sales prices. For these more complex needs, we introduce a second mode, called *ZQL*, short for ZENVISAGE Query Language.

ZQL is targeted to be a visualization exploration language, for exploring collections of visualizations at a time. We designed ZQL to be to visualizations what SQL is to data. ZQL operates on visualization collections and returns visualization collections. ZQL's design draws on data manipulation languages like Query By Example [79] and visualization specification languages like VizQL [65] and Grammar of Graphics [72]. ZQL conceptually captures two types of operations: *composition* and *processing*. The composition operation enables us to compose a collection of visualizations, while the processing operation allows us to operate on a collection of visualizations, by filtering, comparing, or sorting these visualizations. In our representation in subsequent examples, we use a circle to depict a single visualization and a square to depict a process operation. (In actuality, our query language is textual [61], but we use a pictorial representation here for ease of understanding.)

One of the simplest ZQL queries, with only a single composition and no processing, looks like Figure 17 right. Here, the X axis is $\star$, the Y axis is average soldprice, and Z axis is Beijing city—meaning that there is one visualization for each possible X axis attribute for the subset of data corresponding to Beijing city. A slightly more complex ZQL query is shown in Figure 17 left. Here, the X axis is year, the Y axis is $\star$, and Z axis is city.$\star$. This collection contains one visualization for each combination of Y axis attribute and city value. For example, if there are m different attributes that can be displayed on the Y axis, and n different aggregates that can be applied to each of them, and p different values for city, there are $m \times n \times p$ visualizations in that collection.

The process operation also makes use of various in-built functions such as checking for similarity (i.e., compare two visualizations for similarity), increasingness (i.e., check if a visualization has an increasing shape), and representativeness (i.e., identify representative visualizations in a collection), among others. Users can also add their own domain-specific processing functions as well.

We illustrate the capabilities of ZQL via two examples. Say we want to find the states where the soldprice trend is most similar to the soldpricepersqft (i.e., sold price per square foot) trend; this query is shown in Figure 17 top. The way to do this is to compose two collections of visualizations. One with X1 = year, Y1 = avg(soldprice), and Z1 = state.$\star$; thus there is one visualization for each possible state. Then, there is another collection of visualizations with X2 = year, Y2 = avg(soldprice), and we reuse the same Z1 variable—indicating that this collection of visualizations is coupled with the first one when we iterate on the Z1 variable. Subsequently, we compare pair of visualizations via a processing operation in the center of Figure 17 top, while iterating on the Z1 variable. The Z1 values where the two visualizations are deemed to be *similar* are assigned to Z2, via the `ArgSimilar` operation, akin to `ArgMax`. As a final step, the average(soldprice) visualizations for the states corresponding to Z2 are visualized via composition.



x	y
15.5	3.4
17.3	5.5
18.2	6.7
...	...

Created Classes

Ranges	Count
0 < turnover <= 5 15k < soldprice <= 500k	5194
5 < turnover <= 35.5 15k < soldprice <= 500k	7375
5 < turnover <= 35.5 500k < soldprice <= 538k	560
0 < turnover <= 5 500k < soldprice <= 538k	669

Dataset: real_estate_tutorial
Category: city
X-axis: month
Y-axis: soldpricepersqft
Error Attribute: none

Similarity: [✓ Euclidean Distance] Segmentation DTW MVIP
Aggregation Method: Sum [✓ Average]
K-means Cluster Size: 3
Number of Results: 50
Min Similarity Cutoff: 0
Data Smoothing: None
Smoothing Constant: 0.5
Filter Constraint: e.g. mag>20 Submit
Options: [✓ Consider x-range] [✓ Show original sketch] [✓ Normalized Score] Show scatterplot Reverse y-axis
Input equation: y=x^2 add

Figure 15: ZENVISAGE frontend; explanations for a–l provided in the text.

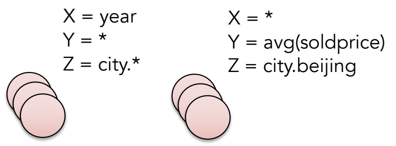


Figure 16: Basic ZQL queries containing composition only.

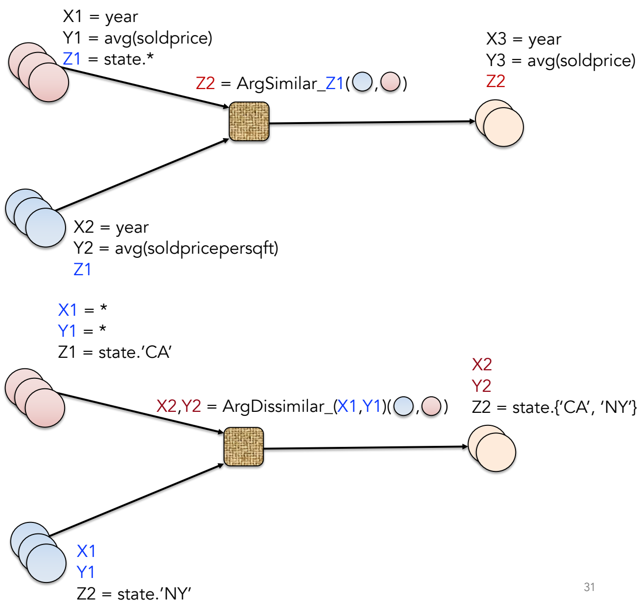


Figure 17: ZQL query examples: find states where the average soldprice and soldpricepersqft trends are similar (top); find visualizations where the states of California and New York are most dissimilar (bottom).

As another example, we can find the visualizations for which the states of California (CA) and New York (NY) are most different. Here, as previously, we compose two collections of visualizations. For the first collection, the X1 and Y1 are set to $\star$ indicating that they can take on any value (i.e., any attribute). The Z1 value is set to be state.CA, indicating that we are focusing on the subset of data corresponding to California. For the second collection, we reuse the X1 and Y1 from the first collection, indicating that this collection is coupled with the first one on X1 and Y1, while Z2 is set to be state.NY. Then, we use a process operation to compare pairs of visualizations while iterating over the X1, Y1 variables. The X1, Y1 attributes for which the visualizations are deemed to be different are assigned to X2, Y2. As a final composition step, we output the corresponding visualizations.

ZQL can be used to issue queries that effectively achieve a range of other goals, including drill-downs (e.g., are there NY cities with an opposite soldprice trend to the overall NY state trend), exceptions (e.g., are there cities with an increasing soldprice trend and an increasing foreclosure rate trend), correlations (e.g., are there attributes that have a similar trend to soldprice for NY cities), and pivots (e.g., for cities similar to NYC for soldprice, what are typical trends for foreclosure rates) [61, 62]. We can also show that ZQL is expressive—it builds on a visual exploration algebra that we develop and has nice theoretical properties [61].

3.3.2 Usability

Now, does the added expressiveness provided by ZENVISAGE, lead to greater usability? To achieve real-world usage beyond a perfunctory nod towards usability, we've been working with astronomy, genetics, and material science researchers for over a year now to refine ZENVISAGE via a process of user-centered design that involved participatory design elements [37]. Figure 18 displays a chart of features we've added to ZENVISAGE. We've focused on our interactive interface and sketch-based querying mode as a starting point; future studies will target ZQL as well.

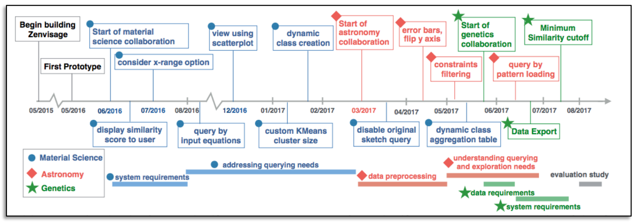


Figure 18: Timeline of features being added to ZENVISAGE.

Our collaborating researchers have used the version of ZENVISAGE that was developed for various findings, including the fact that a dip in a light curve was caused by malfunctioning equipment (for astronomy), a relationship between two specific physical properties of electrolytes was independent of a third one (for battery science), and a reproduction of characteristic gene expression profiles from a recent paper (for genetics).

In our evaluation of how researchers use ZENVISAGE, we discovered various things. First, we discovered that scientists prefer to start from an existing visualization in the canvas, as opposed to drawing one from scratch. For example, a common pattern was to drag-and-drop an existing visualization from the representative or outlier panel, and then modify it (e.g., by smoothing it). Starting from an existing visualization was four times as common as starting from scratch [37]. We also discovered that different domains had typical, but different sensemaking workflows. Borrowing from the terminology of Pirolli and Card [53], astronomers used a lot of "top-down" theory-driven exploration where they asked questions based on what they knew, such as by dragging and dropping a known pattern. Genomics researchers used a lot of "bottom up" data-driven exploration, where they inferred patterns based on what they observed, such as using the representative pattern and sketch capabilities. Battery scientists, on the other hand, focused on understanding and manipulating the space of attributes or visualizations as opposed to top-down or bottom-up exploration.

3.3.3 Scalability

Finally, how do we effectively and efficiently execute queries issued to ZENVISAGE, either via the interactive interface (which can in-turn be rewritten as a ZQL query), or via ZQL?

Consider a backend database where the data resides. Then, a simple mechanism to execute a ZQL query such as those in Figure 17 is to translate each visualization that is composed into a SQL query, as described in Section 3.2. We do so for each visualization in the collection, and once the visualizations are composed, we can apply processing on them, followed by further composition, as needed. Unfortunately, even just composing a single collection of visualizations can be problematic. If we have a collection composed of, say, 100 visualizations, we will end up issuing 100 independent SQL queries to the backend database, resulting in 100 scans of the underlying relation (assuming a denormalized table on which the query is being issued)—taking forever to get a response. The generation techniques described in Section 3.2 will come in handy to accelerate the generation of a single visualization, but they do not make the problem disappear. Next, we will develop other techniques that help accelerate the execution of ZQL queries of the above form.

One simple technique to accelerate the execution of ZQL queries is to share accesses to the database, by combining multiple visualizations into a single SQL query. We explored two variants for this. The first variant combines queries for various Z values into an additional group-by clause. That is, we turn `SELECT X, F(Y) WHERE Z=z GROUP BY X` for various z into a `SELECT X, F(Y), Z GRP. BY X, Z ORDER BY Z`. This simple optimization often leads to a 100× speedup. The second variant is to combine one or more group-bys into a single one. This optimization is more problematic since the number of groups grows multiplicatively in the number of distinct values per attribute [70]. Finding the best way to pack group-bys into as small a number of queries is actually NP-HARD, but heuristics based on first-fit algorithms for bin-packing do quite well [70].

Another technique is to extend the generation-based online sampling techniques from Section 3.2 to select from a collection of visualizations. Say we are searching for visualizations that are similar to a target sketch. By focusing on visualizations that are deemed to be closer to the target and therefore likely to be ranked highly in our search results, and pruning away visualizations that are not close by, we can terminate more quickly with up to 35× speedup over approaches that don't employ sampling, with near-perfect accuracies [44].

Other techniques rely on *parallelism*, i.e., parallel issuance of SQL queries, allowing the database to rely on multiple cores for execution while sharing the buffer pool, *speculative execution*, i.e., eagerly processing compositions that appear downstream in the ZQL query by combining that with compositions that appear early on, and *cache consciousness*, i.e., grouping portions of data that are iterated on multiple times into chunks that fit within the cache followed by reordering of accesses to exploit cache locality, reducing unnecessary data transfer between memory and cache [61, 70].

3.4 Next Steps and Selected Related Work

Despite all of our progress in efficiently and effortlessly generating and searching across visualizations using ZENVISAGE, the underlying challenges in visual data exploration are far from solved. The holy grail here is to be able to instantly identify visual insights from large datasets with little effort. We're starting to extend our preliminary work towards this holy grail in many ways:

More Natural Modalities. In many situations, users do not have a very specific sketch in mind, but instead have a more vague pattern (e.g., two peaks, followed by a plateau). We're developing natural language search capabilities to facilitate the search for such patterns [63]. Challenges include parsing the search query, and identifying matches while tolerating substantial amounts of noise.

More Visualization Types. While ZENVISAGE's interactive interface allows searching for line chart patterns, it doesn't yet support the search for scatterplots. We've been developing sketch-based search techniques for scatterplots [36], wherein the goals are to translate polygonal sketch specifications into matching objectives, and dealing with the explosion in the number of data points that arise across scatterplots.

More Recommendations. Broadly, when users do not know where to start exploration, they need to be recommended visualizations as a starting point. However, identifying what would constitute an "interesting" insight or discovery to users is challenging [38, 69]. As a first step, we've explored the construction of a dashboard of visualizations that summarizes all of the interesting and prominent distributions in a dataset, while avoiding false discoveries [35].

There has been a lot of recent work on visualization generation and search that has inspired our work. We share the research goals set out by Wu and Nandi towards explicitly creating and building perceptual models for visualization, as a step towards approximate query processing for visualization [75]. The excellent paper on M4 targets the accelerated generation of time-series taking into account perceptual aspects [31]; an orthogonal but important direction by Wu et al [76] addresses the need to persist outliers in visualizations. Work from the visualization community, such as Nanocubes [40] and Immens [42], has adapted data cube techniques for exploring a restricted space of visualizations. Other recent and complementary work has evaluated the benefits of progressive visualization methods [47, 78], reaching similar conclusions as our work.

We've been encouraged that others in the visualization and database community have been building flexible interfaces for searching for visualizations, including Datatone [22], Eviza [59], TimeSearcher [27], and Qetch [45]. At the same time, others have started exploring visualization recommendation interfaces [19, 20, 32, 74], while minimizing false discoveries [16].

4. CONCLUSION

Our research over the past five years has targeted the pressing need for powerful tools that can help end-users—regardless of skill level—extract insights from large data sets. In designing these tools, we have recognized that human attention is, in fact, the scarcest resource—and, therefore, our designs optimize the participation of humans in the data science process, impacting every layer of the system stack. The data management community, in conjunction with the data mining and human-computer interaction communities, have an important role to play in making data science accessible to all. We hope you'll join us in this exciting journey!

Acknowlegements

Many thanks to the VLDB Endowment Awards Committee for this wonderful recognition for our work. This work was primarily done by my talented, energetic, and creative group of Ph.D. students, both past and present, Mangesh Bendre, Akash Das Sarma, Silu Huang, Yihan Gao, Doris Lee, Stephen Macke, Sajjadur Rahman, Tarique Siddiqui, Tana Wattanawaroon, Doris Xin, and Liqi Xu. Several masters and undergraduate students have also contributed to the projects listed, but they are far too many to name! It has been a joy and a privilege to work alongside all of them.

I've learned a tremendous amount from my brilliant collaborators, Eric Blais, Amol Deshpande, Aaron Elmore, Karrie Karahalios, Ronitt Rubinfeld, Saurabh Sinha, and Venkat Viswanathan, as well as their students. Karrie Karahalios, in particular, was instrumental in grounding much of this work in the context of the HCI literature and best-practice. Not much changes after one gets a Ph.D.; even as a faculty member, I've turned to my mentors, both formal and informal: Hector Garcia-Molina, Alon Halevy, Joe Hellerstein, H. V. Jagadish, Samuel Madden, Alkis Polyzotis, and Jennifer Widom, for wise counsel time and time again. Many thanks are due to the University of Illinois, for being an excellent home for nearly five years, and to several Illinois faculty for their help, advice, and friendship: Indy Gupta, Jiawei Han, Jian Peng, Hari Sundaram, and Marianne Winslett. Thanks are also due to my new home, the University of California, Berkeley.

None of this research would be possible without support from funding organizations—both federal and from industry. We gratefully acknowledge support from grants IIS-1513407, IIS-1633755, IIS-1652750, and IIS-1733878 awarded by the National Science Foundation, grants 1U54GM114838 awarded by NIGMS and 3U54-EB020406-02S1 awarded by NIBIB through funds provided by the trans-NIH Big Data to Knowledge (BD2K) initiative (`www.bd2k.nih.gov`), grant W911NF-18-1-0335 awarded by the Army, and funds from Microsoft, Adobe, Toyota Research Institute, Google, and the Siebel Energy Institute.

5. REFERENCES

[1] Airline On-Time Performance Data, Bureau of Transportation Statistics, 2019. `https://www.transtats.bts.gov/DatabaseInfo.asp?DB_ID=120&DB_URL=`.

[2] Counted B-Trees, Simon Tatham, 2019. `https://www.chiark.greenend.org.uk/~sgtatham/algorithms/cbtree.html`.

[3] Digital Insights Are The New Currency Of Business, Forrester Report, 2018. `https://www.forrester.com/report/Digital+Insights+Are+The+New+Currency+Of+Business/-/E-RES119109`.

[4] India to overtake US on number of developers by 2017, Computer World, 2013. `https://www.computerworld.com/article/2483690/india-to-overtake-u-s--on-number-of-developers-by-2017.html`.

[5] Maslow's hierarchy of needs, 2019. `https://en.wikipedia.org/wiki/Maslow%27s_hierarchy_of_needs`.

[6] Piet Mondrian Wikipedia Page, 2019. `https://en.wikipedia.org/wiki/Piet_Mondrian`.

[7] Examples of commonly used formulas. `support.office.com/en-us/article/examples-of-commonly-used-formulas-b45a3946-819e-455e-ac20-770ea6aa05da`, 2017.

[8] A. Abouzied et al. Dataplay: interactive tweaking and example-driven correction of graphical database queries. In *UIST*, 2012.

[9] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy. The aqua approximate query answering system. In *ACM Sigmod Record*, volume 28, pages 574–576. ACM, 1999.

[10] 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, New York, NY, USA, 2013. ACM.

[11] E. Bakke and D. R. Karger. Expressive query construction through direct manipulation of nested relational results. In *SIGMOD*. ACM, 2016.

[12] M. Bendre, B. Sun, D. Zhang, X. Zhou, K. C.-C. Chang, and A. Parameswaran. Dataspread: Unifying databases and spreadsheets. *PVLDB*, 8(12):2000–2003, 2015.

[13] M. Bendre, V. Venkataraman, X. Zhou, K. C. Chang, and A. G. Parameswaran. Towards a holistic integration of spreadsheets with databases: A scalable storage engine for presentational data management. In *34th IEEE International Conference on Data Engineering, ICDE 2018, Paris, France, April 16-19, 2018*, pages 113–124, 2018.

[14] M. Bendre, T. Wattanawaroon, K. Mack, K. Chang, and A. Parameswaran. Anti-freeze for large and complex spreadsheets: Asynchronous formula computation. In *Proceedings of the 2019 International Conference on Management of Data*, pages 1277–1294. ACM, 2019.

[15] M. Bendre, T. Wattanawaroon, S. Rahman, K. Mack, Y. Liu, S. Zhu, Y. Lu, P. Yang, X. Zhou, K. C. Chang, K. Karahalios, and A. G. Parameswaran. Faster, higher, stronger: Redesigning spreadsheets for scale. In *35th IEEE International Conference on Data Engineering, ICDE 2019, Macao, China, April 8-11, 2019*, pages 1972–1975, 2019.

[16] C. Binnig, L. De Stefani, T. Kraska, E. Upfal, E. Zgraggen, and Z. Zhao. Toward sustainable insights, or why polygamy is bad for you. In *CIDR*, 2017.

[17] M. Budiu, P. Gopalan, L. Suresh, U. Wieder, H. Kruiger, and M. K. Aguilera. Hillview: A trillion-cell spreadsheet for big data. *arXiv preprint arXiv:1907.04827*, 2019.

[18] Z. Chen and M. Cafarella. Automatic web spreadsheet data extraction. In *Proceedings of the 3rd International Workshop on Semantic Search Over the Web*, pages 1:1–1:8. ACM, 2013.

[19] F. Chirigati, H. Doraiswamy, T. Damoulas, and J. Freire. Data polygamy: the many-many relationships among urban spatio-temporal data sets. In *Proceedings of the 2016 International Conference on Management of Data*, pages 1011–1025. ACM, 2016.

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

[21] Dark Energy Survey Collaboration: Fermilab, University of Illinois at Urbana-Champaign, University of Chicago, Lawrence Berkeley National Laboratory, Cerro-Tololo Inter-American Observatory and Flaugher, Brenna. The dark energy survey. *International Journal of Modern Physics A*, 20(14):3121–3123, 2005.

[22] T. Gao, M. Dontcheva, E. Adar, Z. Liu, and K. G. Karahalios. Datatone: Managing ambiguity in natural language interfaces for data visualization. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software &#38; Technology*, UIST '15, pages 489–500, New York, NY, USA, 2015. ACM.

[23] Y. Gao, S. Huang, and A. Parameswaran. Navigating the data lake with datamaran: Automatically extracting structure from log datasets. In *Proceedings of the 2018 International Conference on Management of Data*, SIGMOD '18, pages 943–958, New York, NY, USA, 2018. ACM.

[24] M. N. Garofalakis and P. B. Gibbon. Approximate query processing: Taming the terabytes. In *VLDB*, pages 725–, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[25] J. M. Hellerstein, P. J. Haas, and H. J. Wang. Online aggregation. *ACM SIGMOD Record*, 26(2):171–182, jun 1997.

[26] N. Henke, J. Bughin, M. Chui, J. Manyika, T. Saleh, B. Wiseman, and G. Sethupathy. The age of analytics: Competing in a data-driven world. *McKinsey Global Institute*, 4, 2016.

[27] H. Hochheiser and B. Shneiderman. Interactive exploration of time series data. In *The Craft of Information Visualization*, pages 313–315. Elsevier, 2003.

[28] S. Huang, L. Xu, J. Liu, A. J. Elmore, and A. Parameswaran. Orpheus db: bolt-on versioning for relational databases. *PVLDB*, 10(10):1130–1141, 2017.

[29] S. Idreos et al. dbtouch: Analytics at your fingertips. In *CIDR*, 2013.

[30] H. V. Jagadish et al. Making database systems usable. In *SIGMOD*, pages 13–24. ACM, 2007.

[31] U. Jugel, Z. Jerzak, G. Hackenbroich, and V. Markl. M4: a visualization-oriented time series data aggregation. *PVLDB*, 7(10):797–808, 2014.

[32] S. Kandel, R. Parikh, A. Paepcke, J. Hellerstein, and J. Heer. Profiler: Integrated statistical analysis and visualization for data quality assessment. In *Advanced Visual Interfaces*, 2012.

[33] A. Kim, E. Blais, A. Parameswaran, P. Indyk, S. Madden, and R. Rubinfeld. Rapid sampling for visualizations with ordering guarantees. *PVLDB*, 8(5):521–532, 2015.

[34] A. Kim, L. Xu, T. Siddiqui, S. Huang, S. Madden, and A. Parameswaran. Optimally leveraging density and locality for exploratory browsing and sampling. In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics*, page 7. ACM, 2018.

[35] D. J. L. Lee, H. Dev, H. Hu, H. Elmeleegy, and A. G. Parameswaran. Avoiding drill-down fallacies with vispilot: assisted exploration of data subsets. In *IUI*, pages 186–196, 2019.

[36] D. J. L. Lee, J. Kim, R. Wang, and A. G. Parameswaran. SCATTERSEARCH: visual querying of scatterplot visualizations. *CoRR*, abs/1907.11743, 2019.

[37] D. J. L. Lee, J. Lee, T. Siddiqui, J. Kim, K. Karahalios, and A. G. Parameswaran. You can't always sketch what you want: Understanding sensemaking in visual query systems. *VAST at VIS*, 2019.

[38] D. J. L. Lee and A. G. Parameswaran. The case for a visual discovery assistant: A holistic solution for accelerating visual data exploration. *IEEE Data Eng. Bull.*, 41(3):3–14, 2018.

[39] X. Li, J. Han, and H. Gonzalez. High-dimensional olap: a minimal cubing approach. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, pages 528–539. VLDB Endowment, 2004.

[40] L. D. Lins, J. T. Klosowski, and C. E. Scheidegger. Nanocubes for real-time exploration of spatiotemporal datasets. *IEEE TVCG*, 19(12):2456–2465, 2013.

[41] B. Liu and H. Jagadish. A spreadsheet algebra for a direct data manipulation query interface. In *ICDE*, pages 417–428. IEEE, 2009.

[42] Z. Liu, B. Jiang, and J. Heer. immens: Real-time visual querying of big data. In *CGF*, volume 32, pages 421–430. Wiley Online Library, 2013.

[43] K. Mack, J. Lee, K. C. Chang, K. Karahalios, and A. G. Parameswaran. Characterizing scalability issues in spreadsheet software using online forums. In *Extended Abstracts of the 2018 CHI Conference on Human Factors in Computing Systems, CHI 2018, Montreal, QC, Canada, April 21-26, 2018*, 2018.

[44] S. Macke, Y. Zhang, S. Huang, and A. Parameswaran. Fastmatch: Adaptive algorithms for rapid discovery of relevant histogram visualizations. *PVLDB*, 2017.

[45] M. Mannino and A. Abouzied. Expressive time series querying with hand-drawn scale-free sketches. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI '18, pages 388:1–388:13, New York, NY, USA, 2018. ACM.

[46] Microsoft UK Enterprise Team. How finance leaders can drive performance. `https://enterprise.microsoft.com/en-gb/articles/roles/finance-leader/how-finance-leaders-can-drive-performance/`, 2015.

[47] D. Moritz, D. Fisher, B. Ding, and C. Wang. Trust, but verify: Optimistic visualizations of approximate queries for exploring big data. In *CHI*, pages 2904–2915. ACM, 2017.

[48] B. A. Myers, A. J. Ko, T. D. LaToza, and Y. Yoon. Programmers are users too: Human-centered methods for improving programming tools. *Computer*, 49(7):44–52, 2016.

[49] A. Nandi, L. Jiang, and M. Mandel. Gestural Query Specification. *VLDB Endowment*, 7(4), 2013.

[50] B. A. Nardi and J. R. Miller. An ethnographic study of distributed problem solving in spreadsheet development. In *Proceedings of the 1990 ACM conference on Computer-supported cooperative work*, pages 197–208. ACM, 1990.

[51] B. A. Nardi and J. R. Miller. *The spreadsheet interface: A basis for end user programming*. Hewlett-Packard Laboratories, 1990.

[52] R. R. Panko. What we know about spreadsheet errors. *Journal of Organizational and End User Computing (JOEUC)*, 10(2):15–21, 1998.

[53] P. Pirolli and S. Card. The sensemaking process and leverage points for analyst technology as identified through cognitive task analysis. In *Proceedings of international conference on intelligence analysis*, volume 5, pages 2–4, 2005.

[54] S. Rahman, M. Aliakbarpour, H. K. Kong, E. Blais, K. Karahalios, A. Parameswaran, and R. Rubinfield. I've seen "enough": Incrementally improving visualizations to support rapid decision making. In *PVLDB*, 2017.

[55] S. Rahman, M. Bendre, P. Yang, S. Z. Yuyang Liu, Z. Su, K. Chang, K. Karahalios, and A. Parameswaran. Extending Spreadsheets to Support Seamless Navigation at Scale. `http://dataspread.github.io/papers/noah.pdf`, Technical Report, 2019.

[56] V. Raman et al. Online dynamic reordering for interactive data processing. In *VLDB*, volume 99, pages 709–720, 1999.

[57] V. Raman and J. M. Hellerstein. Potter's wheel: An interactive data cleaning system. In *VLDB*, volume 1, pages 381–390, 2001.

[58] V. Raman, B. Raman, and J. M. Hellerstein. Online dynamic reordering. *The VLDB Journal*, 9(3):247–260, Dec. 2000.

[59] V. Setlur, S. E. Battersby, M. Tory, R. Gossweiler, and A. X. Chang. Eviza: A Natural Language Interface for Visual Analysis. *Proceedings of the 29th Annual Symposium on User Interface Software and Technology - UIST '16*, pages 365–377, 2016.

[60] B. Shneiderman. Direct Manipulation: A Step Beyond Programming Languages. *IEEE Computer*, 16(8):57–69, 1983.

[61] 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.

[62] T. Siddiqui, J. Lee, A. Kim, E. Xue, X. Yu, S. Zou, L. Guo, C. Liu, C. Wang, K. Karahalios, et al. Fast-forwarding to desired visualizations with zenvisage. In *CIDR*, 2017.

[63] T. Siddiqui, P. Luh, Z. Wang, K. Karahalios, and A. Parameswaran. Shapesearch: flexible pattern-based querying of trend line visualizations. *PVLDB*, 11(12):1962–1965, 2018.

[64] S. Sinha, J. Song, R. Weinshilboum, V. Jongeneel, and J. Han. Knoweng: a knowledge engine for genomics. *Journal of the American Medical Informatics Association*, 22(6):1115–1119, 2015.

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

[66] E. R. Tufte. *The visual display of quantitative information*, volume 2. Graphics press Cheshire, CT, 2001.

[67] J. Tyszkiewicz. Spreadsheet as a relational database engine. In *SIGMOD*, pages 195–206. ACM, 2010.

[68] H. Varian. Artificial intelligence, economics, and industrial organization. Technical report, National Bureau of Economic Research, 2018.

[69] M. Vartak, S. Huang, T. Siddiqui, S. Madden, and A. G. Parameswaran. Towards visualization recommendation systems. *SIGMOD Record*, 45(4):34–39, 2016.

[70] M. Vartak, S. Rahman, S. Madden, A. Parameswaran, and N. Polyzotis. Seedb: efficient data-driven visualization recommendations to support visual analytics. *PVLDB*, 8(13):2182–2193, 2015.

[71] V. Viswanathan and B. M. Knapp. Potential for electric aircraft. *Nature Sustainability*, 2(2):88–89, 2019.

[72] L. Wilkinson. The grammar of graphics. In *Handbook of Computational Statistics*, pages 375–414. Springer, 2012.

[73] J. O. Wobbrock and J. A. Kientz. Research contributions in human-computer interaction. *interactions*, 23(3):38–44, 2016.

[74] K. Wongsuphasawat, Z. Qu, D. Moritz, R. Chang, F. Ouk, A. Anand, J. Mackinlay, B. Howe, and J. Heer. Voyager 2 : Augmenting Visual Analysis with Partial View Specifications. 2017.

[75] E. Wu and A. Nandi. Towards perception-aware interactive data visualization systems. In *DSIA Workshop, IEEE VIS*, 2015.

[76] Y. Wu, B. Harb, J. Yang, and C. Yu. Efficient evaluation of object-centric exploration queries for visualization. *PVLDB*, 8(12):1752–1763, 2015.

[77] D. Xin, S. Macke, L. Ma, J. Liu, S. Song, and A. Parameswaran. Helix: Holistic optimization for accelerating iterative machine learning. *PVLDB*, 12(4):446–460, 2018.

[78] E. Zgraggen, A. Galakatos, A. Crotty, J.-D. Fekete, and T. Kraska. How progressive visualizations affect exploratory analysis. *IEEE transactions on visualization and computer graphics*, 23(8):1977–1987, 2016.

[79] M. M. Zloof. Query-by-example: A data base language. *IBM systems Journal*, 16(4):324–343, 1977.