

Gaussian Cubes: Real-Time Modeling for Visual Exploration of Large Multidimensional Datasets

Zhe Wang, Nivan Ferreira, Youhao Wei, Aarthy Sankari Bhaskar, Carlos Scheidegger

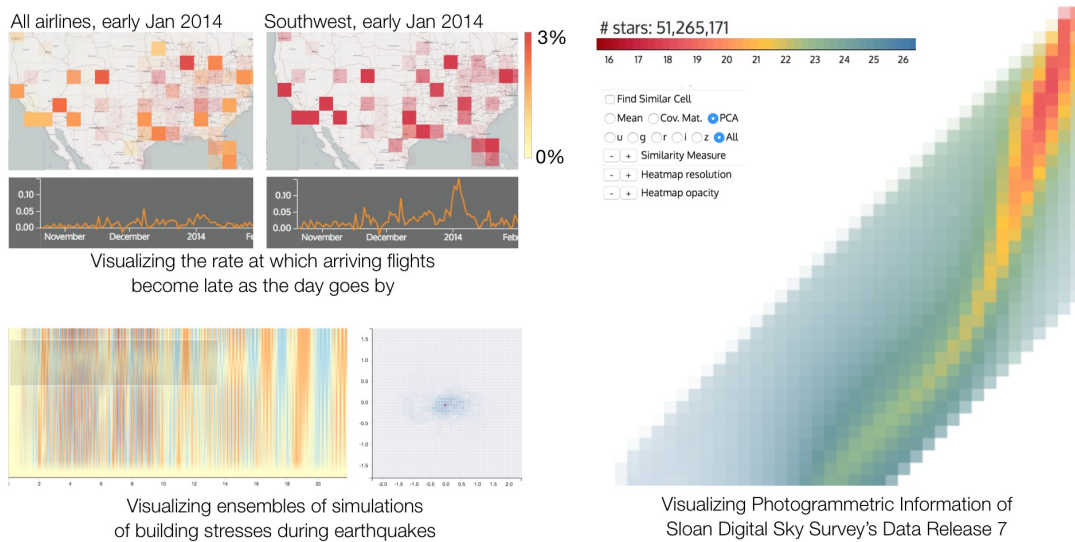


Fig. 1. Some application scenarios for Gaussian Cubes. Top left: screenshot of an interactive session of visual analysis of the Bureau of Transportation Statistic (BTS) on-time performance data, including 160 million flights over a 25-year time span. Gaussian Cubes enable visualizations that show *the slope of the model that describes how flights get later as the day progresses*, and these models can be computed over arbitrary subsets of the data at interactive rates. This makes it easy to spot Southwest Airlines’s alleged practice of indefinitely grounding (but not canceling) delayed flights in early 2014. Subsequently, the Department of Transportation levied on Southwest Airlines the largest fine ever received by an airline [27]. Right: visualization of a model heatmap of a color-color diagram of a large astronomical catalog (the Sloan Digital Sky Survey Data Release 7 [1]), which includes 51 million stars after data cleaning. Bottom left: Gaussian Cubes used as the backing store for a large number of earthquake simulations, enabling fast computation of Principal Component Analysis over arbitrary data subsets. See Section 6 for more details.

Abstract— Recently proposed techniques have finally made it possible for analysts to interactively explore very large datasets in real time. However powerful, the class of analyses these systems enable is somewhat limited: specifically, one can only quickly obtain plots such as histograms and heatmaps. In this paper, we contribute Gaussian Cubes, which significantly improves on state-of-the-art systems by providing interactive *modeling* capabilities, which include but are not limited to linear least squares and principal components analysis (PCA). The fundamental insight in Gaussian Cubes is that instead of precomputing *counts* of many data subsets (as state-of-the-art systems do), Gaussian Cubes precomputes the best multivariate Gaussian for the respective data subsets. As an example, Gaussian Cubes can fit hundreds of models over millions of data points in well under a second, enabling novel types of visual exploration of such large datasets. We present three case studies that highlight the visualization and analysis capabilities in Gaussian Cubes, using earthquake safety simulations, astronomical catalogs, and transportation statistics. The dataset sizes range around one hundred million elements and 5 to 10 dimensions. We present extensive performance results, a discussion of the limitations in Gaussian Cubes, and future research directions.

Index Terms—Data modeling, dimensionality reduction, interactive visualization, data cubes

1 INTRODUCTION

The fundamental difficulty in visual exploration of large datasets can be summarized by two conflicting requirements. First, exploratory analysis requires a large variety of different queries against the dataset, and these queries are not known before the dataset is processed for visualization. Such a constraint naturally pushes implementations towards expressively powerful — but computationally naive — strategies, such as repeated linear scans of the data. Second, user-interaction constraints dictate that the quality of the experience is bound by the ease with which analysts can go through an “exploratory hypothesis cycle”: a sequence of formulating a query, issuing it, receiving the results, examining them, and finally refining some underspecified hypothesis in their mind. This constraint, in turn, pushes implementations

• Zhe Wang, Youhao Wei, Aarthy Sankari Bhaskar and Carlos Scheidegger are with the University of Arizona, E-mail: {zhew, youhaowei, aarthysb, cscheid}@email.arizona.edu.

• Nivan Ferreira is with Universidade Federal de Pernambuco, E-mail: nivan@cin.ufpe.br

Manuscript received xx xxx. 201x; accepted xx xxx. 201x. Date of Publication xx xxx. 201x; date of current version xx xxx. 201x.

For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org.

Digital Object Identifier: xx.xxx/TVCG.201x.xxxxxx/x

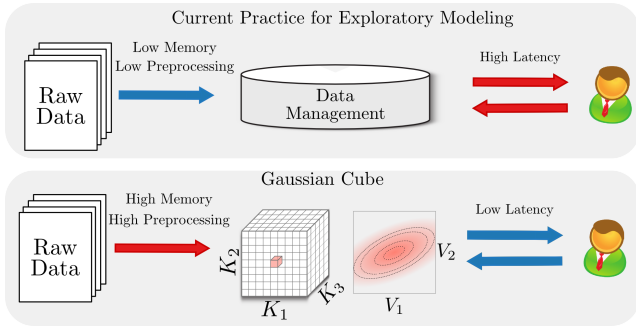


Fig. 2. A summary of our proposed workflow for visual exploratory modeling. In current practice, the building of *models* to explain or explore a dataset typically happens through repeated scans of the dataset. As datasets grow larger, the latency of even a single scan becomes prohibitive. In this paper, we introduce Gaussian Cubes, which extends data cubes in order to support low-latency *exploratory modeling*. Gaussian Cubes enables the computation of model parameters in real-time; with it, we can build interactive visualizations that compare, for example, thousands of principal component analyses over tens of millions of data points on the order of a second (see Section 6.2).

towards computationally-efficient — but expressively impoverished — implementations. This tension underlies much of the current research in interactive systems for large-scale data exploration.

Breakthrough systems like Polaris [41] and formalisms such as conjunctive visual queries [42] have largely solved the issues of expressivity. However, these improvements only brought issues of scalability into sharper focus. If analysts are faced with datasets where a linear scan takes longer than about a second, one can expect the quality of exploratory analysis to suffer [32]. Recently proposed techniques such as imMens [33] and Nanocubes [31] have fundamentally changed the scale of datasets that can be visualized in real time. In order to achieve this performance, these techniques take the classic OLAP *data cube* [20] and tailor it to visualization-specific requirements. Data cubes carefully pre-compute aggregations across different subsets of the data in a way that enables computation of a large class of aggregation queries *without having to refer to the original dataset*. With imMens and Nanocubes, it is now possible to produce popular interactive visualizations such as linked histograms and heat maps for datasets in the order of millions to billions of records, on a commodity computer such as a modern laptop or desktop.

While having such visual summaries in an interactive manner is powerful, they only support a limited class of analysis tasks. One important example of analytical tasks not supported by these techniques is building statistical models from the data. Coupling statistical models with user interactivity is one of the main strengths of modern visual analytics systems [28]. In fact, throughout the analysis process, it is common to derive statistical models to extract features and relations (e.g., regression models), and build complex visual representations (e.g., dimensionality reduction) from the data. However, the computational costs of fitting such models and the need for low latency in exploratory visual analysis [32] prevent the use of these techniques in a truly interactive way, even for reasonably sized datasets. The usual approaches to mitigate this problem are either to make use of only small portions of the data in the interactive analysis or to rely on long preprocessing steps in which the models are built. Both of these approaches are far from ideal. The former might ignore important aspects of the data due to sampling. The latter often restricts the analysts to visualize the results of the modeling without being able to neither change any of the parameters associated with this process nor the portions of the data used to build the model.

In this paper, we contribute Gaussian Cubes, which significantly improves on state-of-the-art systems by providing interactive visual *modeling* capabilities, which include but are not limited to linear least squares and principal components analysis (PCA). Our current implementation of Gaussian Cubes is a relatively simple extension of Nanocubes [31]. As a result, it inherits much of its runtime perfor-

Relation...			...plus modeling variables	
Make	Style	Trans.	Age	Price
Honda	sedan	auto	10	9
BMW	hatch	auto	5	25
Ford	SUV	manual	3	20
Ford	hatch	manual	1	12
BMW	sedan	auto	4	35

Data Cube...				...plus sufficient statistics				
Make	Style	Trans.	Count	$\sum A$	$\sum P$	$\sum A^*A$	$\sum A^*P$	$\sum P^*P$
*	*	*	5	23	101	151	427	2,475
*	*	auto	3	19	69	141	355	1,931
*	*	manual	2	4	32	10	72	544
*	sedan	*	2	14	44	116	230	1,306
*	hatch	*	2	6	37	41	137	1,994
*	SUV	*	1	3	20	9	60	400
*	sedan	auto	2	14	44	116	230	1,306
*	hatch	auto	1	5	25	25	125	625
*	hatch	manual	1	1	12	1	12	144
*	SUV	manual	1	1	20	9	60	400

Fig. 3. On the left, an example of a data cube as it is typically created. From the relation on the top left, the analyst picks a set of column to “cube”. The traditional data cube table (bottom left) collects all possible aggregations —commonly known as “group by’s”— along the selected columns. On the right, we show the added columns of a data cube model for Gaussian Cubes, which makes a distinction between “indexing variables” and “modeling variables” (top right). Gaussian Cubes create data cubes with added columns (bottom right) containing sums of polynomial expansions of the modeling variables (by default, up to degree 2). As we explain in Section 3.2, these sums suffice to find best-fitting linear models in any of the modeling variables. It also enables other modeling techniques, as we discuss further in Section 4.

mance and memory requirements, querying expressivity, and speed. However, we note that the techniques we use are readily available for use in other visualization systems as well. The fundamental insight in Gaussian Cubes is that instead of precomputing *counts* of many data subsets (as imMens and Nanocubes do), Gaussian Cubes precomputes the best multivariate Gaussian distribution for a given set of real-valued variables (Figure 2). As a result, Gaussian Cubes can fit hundreds of models over millions of data points in well under a second, enabling novel types of visual exploration of such large datasets. While the idea of indexing sufficient statistics has been used for data mining and machine learning [35, 13, 38], the main novelty of Gaussian Cubes is to use this idea to extend modern visualization-focused data cube systems. This enables many novel plots that haven’t been previously attempted, mostly because of their unfavorable performance characteristics, and we describe some of these plots in Section 6. We present three case studies that highlight the visualization and analysis capabilities in Gaussian Cubes, using simulations of building stress under earthquakes, astronomical catalogs, and transportation statistics. The dataset sizes range around one hundred million elements and 5 to 10 dimensions. Finally, we also present extensive performance results, a discussion of the limitations in Gaussian Cubes, and future research directions.

2 RELATED WORK

Gaussian Cubes lie at the intersection of data analysis, database management systems, and information visualization. As a result, there exist related work spanning all of these areas.

Visualization and Data Management. Stolte et al.’s Polaris system was a breakthrough system that showed the fundamental relationship between OLAP cubes, aggregation, and interactive visualization [41]. The need for visualization systems to offer interactive query times for large datasets drove the development of visualization-specific data cubes such as imMens [33] and Nanocubes [31]. Gaussian Cubes are a followup to these proposals, and the current implementation is specifically built as an extension to Nanocubes.

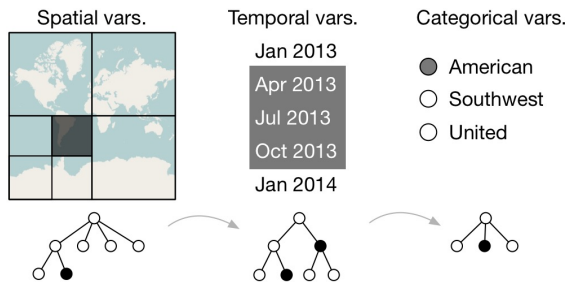


Fig. 4. The implementation of Gaussian Cubes are based on Nanocubes [31], which are an implementation of *spatiotemporal* cubes. Such cubes generate intermediate aggregates that correspond to filtering operations that naturally appear in spatial and temporal queries, such as queries over time intervals and contiguous geographic regions.

Visual encodings of model information. Gaussian Cubes enable computation of *statistical models* at the same interactive rates that previous systems computed subpopulation counts. As a result, we can now leverage a large amount of pre-existing work in visual encodings of statistical information. Cottam et al. propose *abstract rendering*, a pixel-based metaphor in which pixels store binned model information [16]. Abstract rendering is a generalization of the traditional pixel-based visualizations and mappings; for a thorough review of the field, we point interested readers to Keim’s survey [29]. Chan et al. recently developed a technique they call Regression Cube [12], which combines model fitting and dimensionality reduction. The focus of Gaussian Cubes, in contrast, are in enabling *fast* computations over a (possibly more restricted) class of queries.

Data Management. The data management research community has recently become aware of the importance of fluid interaction to the overall usability of data management systems. We highlight here two recent developments. Agarwal et al.’s BlinkDB [3] is an especially efficient implementation of Hellerstein et al.’s vision of online aggregation [24]; BlinkDB creates a large set of stratified samples from which many queries can be answered with relatively high precision and confidence, and at relatively low latency. It offers a natural backend for the developments in visualization of streaming results from a sample-oriented database [19, 18, 5]. Instead of modeling the low-level visual perceptual system in order to provide fast, approximate, perceptually-similar query results, a different avenue of research is to model user interactions, with the goal of *predicting* their activity and hiding latency behind successful predictions [6, 11].

Data mining. The proposal of using preaggregation to speed-up the process of fitting statistical models has been previously explored in the data mining literature. Shao et al. [38] introduced the idea of storing sufficient statistics in data cubes. Based on this idea, they proposed a multivariate aggregate view for relational database, enabling fast data mining queries. The seminal work by Moore et al. [35] proposed precomputation of sufficient statistics to obtain models for different portions of the data. This work inspired further development in the area [13, 44]. Gaussian Cubes leverage this idea and the power of visualization oriented data cube systems to enable both model fitting and exploratory visual analysis.

Much of the work in visual analytics is grounded on the maxim that visual encodings should be intimately related to statistical models that describe the data well [4]. Gaussian Cubes can be seen as enabling interactive, query-based visual analytics for a particular class of models. There have been data cube systems developed for the purposes of faster calculations of some classes of models [14, 15]. In contrast, Gaussian Cubes collect aggregations that support both a large class of models, and exploratory visualization of the underlying patterns, as we show in Section 4.

3 GAUSSIAN CUBES

Gaussian Cubes combine insights from data management systems and basic computational statistics. In this section we present background

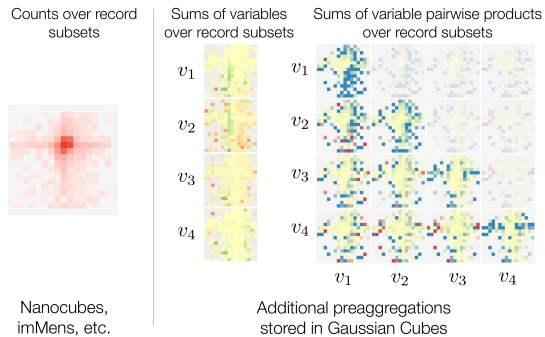


Fig. 5. In addition to sample counts partitioned over the *indexing* variables (the same kind of aggregation scheme used in other visualization-specific data cubes), Gaussian Cubes store the sums of the *modeling* variables, and the sums of their pairwise products. Gaussian Cubes require no changes in the way previous systems lay out their indexing structures, and so the expressivity of the “slicing and dicing” capabilities is unchanged. In exchange for the additional memory usage, we get the ability to fit a number of models over large datasets, at interactive rates.

on these insights as well as the intuition behind Gaussian Cubes. Also, throughout this section, we will use the tables in Figure 3 as running examples.

3.1 Data Cubes: Fast queries from preaggregations

Exploratory analysis has long relied on *aggregations* for simple summaries of relevant information in a dataset of interest. Following common practice, we call the two tables in Figure 3 *relations*. Their columns store *attributes*; in turn, rows store *records*, and individual entries in either are *values*. Any set of records can in principle produce an *aggregation*: a new record that summarizes their information somehow. The prototypical aggregation is the *sum* operation: aggregating the set of records which represent BMW cars would yield a row (BMW, *, *, 2); notice the row has additional attribute, in this case a “count”. Typically, aggregations are built by partitioning on the values of an attribute (this is the SQL *group by* clause [39]).

The data cube, as originally defined, is a relation that stores aggregations of the power set (the set of all subsets) of a user-defined attribute set. On the bottom-left side of Figure 3, we show a data cube on the attribute set {Style, Transmission}. Data cubes formalize the notion that *group by* operations can be created for many possible sets of attributes, and that these aggregations *nest* in a very particular way. Specifically, if one computes an aggregation relation *A* on car makes and styles of a relation *R*, and then aggregates *A* only on car makes, the result is exactly the same as computing the aggregation on car makes directly from *R*.

In a single sentence, the fundamental insight is that *many aggregations can be build incrementally, and efficiently, from previous aggregations*: to find the total number of BMW or Hondas sold, we simply add the counts of the rows corresponding each to total Hondas and total BMWs sold, without having to scan the original relation. This is what allows imMens and Nanocubes (and essentially other data cube structures) to quickly recover a relatively large number of aggregations from a (carefully constructed) relatively small “basis set” of aggregations.

In this section and in the ones which follow, it will be helpful to think of the structure representing a data cube as a directed graph. This observation, to the best of our knowledge, is due to Sismanis et al.[40]. Each node of the graph (stored as a record in the data cube table) encodes an aggregation for a particular set of records, and edges connect coarser aggregations to finer ones. If there exists an edge from node N_i to node N_j , then the aggregation represented by N_i is over a set which contains that of the node N_j . Moving along the edges *refines* the query set (by choosing, for example, a specific spatial region, time interval, or attribute value). For readers interested in more details, we recommend the original presentation from Gray et al.’s classic paper [20].

3.2 Sufficient statistics: what is really required to fit a model?

Gray et al.’s breakthrough paper already notes that it is possible to build aggregations with many different functions besides `sum` (e.g. `min` and `max`). Consider a slightly different example from before: imagine we want to find *averages* (of, for example, sales prices). It is possible to use data cubes for this task, but we need to be somewhat careful; in order for data cubes to work properly, aggregations need to be *associative and commutative*: the order in which aggregations happen must not affect the outcome. Consider the set $\{1, 2, 3\}$. If we (incorrectly) build averages by averaging 1 and 2, and then averaging 1.5 and 3, clearly the result is wrong. The solution, of course, is to compute the appropriate information from which to find averages. In this case, we keep a running sum of the prices *and* a running count; the average is obviously the ratio. In statistics parlance, the sum of prices and the cardinality of a set of records are both *sufficient statistics* to compute the average price of that set of records.

This particular trick is folklore. However, it is not as well-known in some fields that same principle of sufficient statistics applies much more generally, and this principle is central to our proposal. To the best of our knowledge, Gaussian Cubes are the first system to take central advantage of this concept to build fast interactive tools for visual modeling. In fact, one way to think of Gaussian Cubes is as a spatiotemporal data cube (Figure 4) of sufficient statistics, coupled with a system to query and inspect results visually (Figure 5).

Here, let us thoroughly work through a simple example of linear regression. Imagine we have a large dataset of pairs of numbers (x_i, y_i) , and we want to find the linear model that best fits these numbers. In other words, we want an equation

$$y_i = mx_i + b$$

that describes all points equally well. The principle of least-squares says that over all possible choices of m and b , we should pick the one that minimizes the quadratic error E summed over all pairs:

$$E = \sum_i (y_i - mx_i - b)^2$$

We do this by looking for the values for which the derivative of the error with respect to the parameters is zero, $dE/dm = dE/db = 0$. Writing this out,

$$\frac{dE}{dm} = 2 \sum_i (y_i - mx_i - b)y_i = 2(\sum_i y_i^2) - 2m(\sum_i x_i y_i) - 2b(\sum_i y_i) = 0$$

$$\frac{dE}{db} = 2 \sum_i (y_i - mx_i - b) = 2(\sum_i y_i) - 2m(\sum_i x_i) - 2b(\sum_i 1) = 0$$

The crucial observation here is that the model depends on the dataset *only through the sums*. Although additional computation is necessary to obtain the actual parameters, this computation can be done without referring to the original relation. Once we store these sums in a table such as the bottom table in Figure 3, we have everything we need to know in order to compute the parameters of the final model. This is computationally important. If we can find these sums without having to linearly scan the entire dataset, then we obtain a *scalable* method: we have eliminated a runtime dependency on the overall size of the data (the trade-off is that we have, of course, introduced a *preprocessing* requirement and a storage overhead. See Section 6 for more details). Gaussian Cubes can use these sufficient statistics to build a variety of models besides the simplest least squares; we defer a full discussion of the range of applicability to Section 7. We also note that although our implementation is built on top of a specific system, the concept is quite general, and can clearly be applied to other implementations.

3.3 Gaussian Cubes: A Normal Distribution at Every Node

A natural question arises when considering sufficient statistics: which statistics should one store? This decision affects which models can be fit efficiently, and so it merits discussion. As an illustration, it is clear

Input: k : # of Gaussians, x_1, x_2 : Projection Axes, v : Initial Node

Output: $PQ = [\bar{n}_1, \dots, \bar{n}_k]$: Priority queue of Final Nodes

```

PQ.insert( $n$ , projected-variance( $n, x_1, x_2$ ))
repeat
  ( $\bar{n}$ , priority $_{\bar{n}}$ )  $\leftarrow$  PQ.pop-max()
  if priority $_{\bar{n}} = -\infty$  then
    break
  end if
  if  $\bar{n}$ .partitions() =  $\emptyset$  then
    PQ.insert( $n$ ,  $-\infty$ )
  else
    prev-proj-var  $\leftarrow$  projected-variance( $v, x_1, x_2$ )
    priority  $\leftarrow$  new dictionary
    for split in  $\bar{n}$ .partitions() do
      split-vars  $\leftarrow$   $\sum_{n \in \text{split}}$  projected-variance( $n, x_1, x_2$ )
      priority[split]  $\leftarrow$  prev-proj-var - split-vars
    end for
    best-split  $\leftarrow$  argmin(priority)
    for  $n$  in best-split do
      priority  $\leftarrow$  new dictionary
      for split' in  $n$ .partitions() do
        split-vars  $\leftarrow$   $\sum_{n \in \text{split}}$  projected-variance( $n, x_1, x_2$ )
        priority[split]  $\leftarrow$  prev-proj-var - split-vars
      end for
      best-priority  $\leftarrow$  min(priority)
      PQ.insert( $n$ , best-priority)
    end for
  end if
until PQ.length()  $\leq k$ 
return PQ

```

Fig. 6. Algorithm for progressive refinement of a projected Gaussian Cube.

that *some* models cannot be fit using only the sufficient statistics of the previous example, such as one quadratic in x : $y_i = ax_i^2 + bx_i + c$. This means that a full decision of which statistics to precompute will always involve some amount of user input.

At the same time, some classes of sufficient statistics are relatively small, and suffice for a relatively large number of models. In Gaussian Cubes, what we propose to store are statistics to compute *all second-order moments* of a particular subset of variables. The first-order moments suffice to compute averages, and the first- and second-order moments suffice to compute *variances* of these variables. A particularly helpful way to think about these values is that we’re storing information to compute the *number of samples*, their *centroid*, and the *covariance matrix*. This is precisely the information captured by a multivariate normal distribution [10] — hence the name of our proposal.

Computing a traditional data cube requires the analyst to decide on which variables to perform the hierarchical aggregation. Gaussian Cubes introduce an additional decision: over which variables should the analyst compute the second-order moments? For the remainder of the paper, we will refer to the variables in which filtering and grouping can be performed (capabilities existing in traditional data cubes) as the *indexing variables*. The variables with which models are fit, in contrast, will be referred to as *modeling variables*. We currently do not offer an automatic method to make this decision, and leave the choice up to the analysts.

We note that the two sets do not need to be disjoint. In fact, a “fully-materialized” Gaussian cube would include every variable as both indexing and modeling variable. The reason we do not advocate this is simple: even though the total storage of Nanocubes and imMens are typically acceptable, they are ultimately exponential in the size of the indexing variable set. Gaussian Cubes incur an additional multiplicative space overhead that is quadratic on the size of the modeling variable set (see Table 1). This can be seen as both a good and a bad thing. As a negative consequence, some of the data structures we use in our experiments push well into the tens of gigabytes of main memory. On the other hand, a quadratic blowup is better than an exponential one; whenever Gaussian Cubes allow variables which needed to be in the

Dataset	Objects(N)	Memory	Time	Indexing Schema	Modeling Schema	$ dim $
Synthetic	1 M	0.56 GB	14 sec	$x(15), y(15)$	count, a, b, c	10
SDSS DR7 Stars	51 M	12.8 GB	21 min	$i - r(15), i - g(15), g - r(4)$	count, $u, g, r, i, z, e_u, e_g, e_r, e_i, e_z$	66
Flights	163 M	1.74 GB	14 min	lat(25), lon(25), carrier(5), time(16)	count, arrival_time, arrival_delay	6
Earthquake	14 M	14.9 GB	8 min	timestep(15), floor(15), earthquake number(6)	count, shear, diaph. force, moment, acc., interstory drift ratio, drift ratio	28

Table 1. Summary of the datasets and respective Gaussian Cubes used in our experiments. We note that both the overall memory usage and build times are comparable to that of Nanocubes [31]. (In column *Indexing Schema*, the numbers in the parentheses indicate how many bits are used to store that dimension. Column $|dim|$ means the total number of dimensions stored in each Gaussian Cube.)


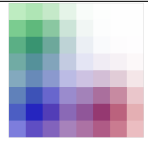
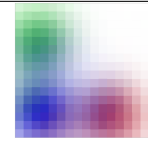
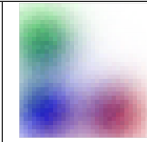
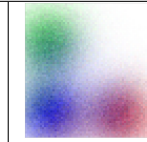
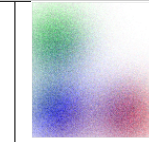
Color Map						
Image Size	4×4	8×8	16×16	32×32	64×64	128×128
Query Time (ms)	2	4	7	21	50	172
Query Time/Cell (ms)	0.125	0.063	0.027	0.021	0.012	0.010
JSON Parsing Time (ms)	3	3	4	5	14	45
PCA Calculation Time (ms)	1	7	33	84	254	718
JSON Size (KB)	2.4	9.1	35.1	136	524	1945.6

Table 2. An illustration of a synthetic dataset design to assess the querying performance of Gaussian Cubes. We note that the query time is essentially proportional to the size of the output image; the query time per cell is essentially constant (the apparent decrease is likely due to a constant overhead from network latency). In addition, the overall time is dominated by the calculation of the Principal Components Analysis. This computation is currently done on the client side in Javascript; there are clear opportunities for parallelization.

indexing set to be pushed over to the modeling set, we can expect an overall *reduction* in overhead.

4 BUILDING VISUALIZATIONS WITH GAUSSIAN CUBES

We now describe how the Gaussian distributions stored in Gaussian Cubes can be used as a way to fit linear models to our data and build visualizations from them. We are concerned with the interactive data exploration scenario in which users are constantly selecting portions of the data and the resulting visualizations (and the models used to build them) need to be updated in real time to reflect those selections. In such scenario, in many typical cases for large datasets, what grows without bounds is the *number of samples*, n . We therefore look for models and visualizations for which we can avoid linear scans over the data.

4.1 Ordinary Least Squares and Generalizations

As discussed in Section 3.2, by using Gaussian Cubes, it is possible to perform linear regression on different subsets of data at interactive rates. In general, it is easy to see that a similar approach can be used to solve a general *linear least squares* problem, i.e., to fit models that depend linearly on the parameters. In fact, considering the model $y = X\beta$, where X is a n by d matrix of observations (data) and y are the observed responses. By using linear least squares, we obtain the parameter β , by minimizing $\|y - X\beta\|^2$. It can be seen that the solution of this problem is given by $\hat{\beta} = (X^T X)^{-1} X^T y$. Although it is typical to consider the matrix inversion calculation to be a time-consuming step (cubic on d), we note that, as previously discussed, the number of samples grows without bounds. Because of this, even though the cost to build the $X^T X$ matrix from a linear scan is $O(nd^2)$, for an overall running time of $O(nd^2 + d^3)$, the $O(n)$ term dominates. If we denote by x^1, \dots, x^d the columns of the matrix X , it is easy to see that

$$(X^T X)_{ij} = \sum_{k=1}^n x_k^i x_k^j.$$

Similarly, the entries of product $X^T y$ are given by

$$(X^T y)_{ij} = \sum_{k=1}^n x_k^i y_k.$$

In the case of Gaussian Cubes, the preaggregations we store are sufficient to compute both $X^T X$ and $X^T y$ effectively in $O(d^2)$ time (assuming all the variables involved are modeling variables). As a result, we expect the overall computation of the solution for a subset of data in a Gaussian Cubes to be on the order of $O(d^3)$. As we show in Section 6.3, this strategy can be used to interactively fit large collections of regression models over millions of records.

In addition, many generalizations of this typical example can *also* be computed directly from the sufficient statistics. We note just one example here, that of *ridge regression* [25]. This model consists of modifying the classical linear regression model to include a regularization term as a way to control model “complexity”. In the simplest formulation of ridge regression, one attempts to minimize $\|y - X\beta\|^2 + \lambda \|\beta\|^2$. In other words, ridge regression tries to balance goodness of fit with the magnitude of the coefficient’s components, which tends to avoid overfitting. The interesting connection with Gaussian Cubes is that the solution here is given by

$$\hat{\beta} = (X^T X + \lambda I)^{-1} X^T y.$$

Thus, once more, we can compute the solution in constant time. A constant source of worry when using regularization is the choice of λ (known as a *hyperparameter* [23]). Notably, with Gaussian Cubes we can visually — and interactively — investigate the effects of different choices of λ , since refitting the models is, in practice, instant.

The framework of ordinary least squares provides a rich setting for future interactive visualization research, which we do not explore here mostly because of space constraints. Possibilities include visualization of ANOVA results, mixed effects least-squares, and even the direct use of per-bin hypothesis tests and effect-size measurements (using, for example, Wald tests or Cohen’s d [10]).

4.2 Principal Components Analysis

Principal Components Analysis is a popular method for dimensionality reduction [17]. In a nutshell, the principal components of a dataset are the directions in which variance is largest (variance being the expected squared distance from the average). By choosing to ignore all but the first few connected components, the analyst’s hope is to preserve most of the signal. Computationally speaking, the principal components are given by the eigenvectors of the covariance matrix. This is particularly

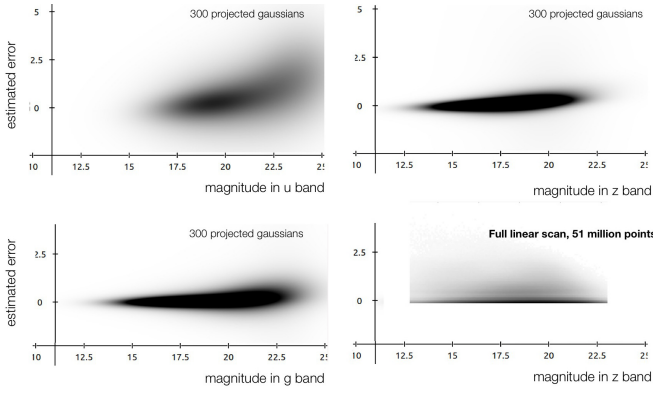


Fig. 7. Approximate scatterplots along non-indexing dimensions. The partition schema doesn’t directly offer a spatial subdivision scheme for the axes being presented, but the traversal algorithm can adaptively subdivide nodes to maximally increase the resolved details of the plot. Here, we show a set of 300 projected Gaussians along different axes of the SDSS dataset. Even though this is a tiny fraction of the available nodes in the graph, they are sufficient to highlight a well-known feature of the dataset: errors in the u band (top left) are much larger than in the other bands [1]. The bottom-right image shows the corresponding (compared to top-right) exact scatterplot, which requires scanning the entirety of the dataset. We discuss the discrepancies in Section 7.

fortunate in the case of Gaussian Cubes, since the modeling variables we store are exactly the ones sufficient to build the covariance matrix.

As in the case of least squares problem, we try to eliminate the dependency on *number of samples*. Although the eigenvector calculation is considered to be the most time-consuming step (cubic in the size of the modeling variable set), which dominates the overall calculation is the construction of the covariance matrix. Actually it is $O(nd^2)$ from a linear scan, where n is the number of rows and d is the number of dimensions (for an overall running time of $O(nd^2 + d^3)$, the $O(n)$ term dominates). In the case of Gaussian Cubes, the preaggregations we store are sufficient to compute the covariance matrix effectively in $O(d^2)$ time. As a result, we expect the overall computation of the PCA for a subset of data in a Gaussian Cubes to be on the order of $O(d^3)$. Just like in the case of least-squares fitting (and sample counts for traditional data cubes), we obtain runtime performances that are effectively *independent* of the overall size of the data. We provide experimental evidence of this in Section 6.1. The procedure to go from moments to covariance via sufficient statistics is spelled out below. Consider a hypothetical 3×3 covariance matrix M :

$$M = \begin{bmatrix} \text{cov}(x,x) & \text{cov}(x,y) & \text{cov}(x,z) \\ \text{cov}(y,x) & \text{cov}(y,y) & \text{cov}(y,z) \\ \text{cov}(z,x) & \text{cov}(z,y) & \text{cov}(z,z) \end{bmatrix} \quad (1)$$

Without loss of generality, consider $\text{cov}(x,y) = \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})$. The time complexity of this term still scales with the dataset size, as $O(n)$. However, the expression can be restated as:

$$\text{cov}(x,y) = (n-1)^{-1} \sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y}) \quad (2)$$

$$= (n-1)^{-1} \sum_{i=1}^n (x_i y_i - x_i \bar{y} - y_i \bar{x} + \bar{x} \bar{y}) \quad (3)$$

$$= (n-1)^{-1} \left(\sum_{i=1}^n x_i y_i - \sum_{i=1}^n x_i \cdot \bar{y} - \sum_{i=1}^n y_i \cdot \bar{x} + n \bar{x} \bar{y} \right) \quad (4)$$

Since $\bar{x} = \sum_{i=1}^n x_i / n$ and $\bar{y} = \sum_{i=1}^n y_i / n$, all we need for the covariance matrix are $\sum_{i=1}^n x_i$, $\sum_{i=1}^n y_i$ and $\sum_{i=1}^n x_i y_i$. Generally, for a d dimensional data set, we need the count n , $\sum_{i=1}^n c_i$, $i = 0, 1, 2, \dots, d$ and $\sum_{i=1}^n n(c_i c_j)$, $i, j = 0, 1, 2, \dots, d$. As we mentioned above, these are precisely the summaries stored in Gaussian Cubes. Thus, we reduce the time for calculating the covariance matrix from $O(nd^2)$ to $O(d^2)$.

5 SCATTERPLOTS OVER PRINCIPAL COMPONENTS WITH GAUSSIAN CUBES

An attentive reader will have noticed that although the algorithm we just described can compute a PCA of a large sample very quickly, the principal components themselves are very rarely the goal of exploratory analysis. In fact, we are typically interested in a *scatterplot* of the sample, using the principal components as the axis. But now we are faced with a problem: the straightforward way to generate a plot requires a scan over the entire dataset (in order, at least to actually rasterize them on the screen!). Although this seems to be the same class of problem that Nanocubes and mMens both solve, the situation here is more complicated: *the axes themselves (principal components) depend on the sample*. We cannot, then, precompute these scatterplots ahead of time!

In lieu of an exact solution, we propose to take advantage of the multivariate normals stored at every level of the data structures of Gaussian Cubes in yet another way to produce approximate scatterplots. Recall that normal distributions have a remarkable property: when transformed by affine transformations (a linear transformation followed by a translation), normal distributions *remain normal*. If values X are drawn from a multivariate normal $N(\mu, \Sigma)$, the distribution of the X values transformed by a matrix M is given by $N(M\mu, M\Sigma M^T)$. As a result, for any desired projection (say, the first two principal components of some sample), given a Gaussian (representing the density of points), we can compute the two-dimensional normal corresponding to its projection.

Therefore, we can generate an approximate scatterplot (a density plot) by using the hierarchical structure of Gaussian Cubes to obtain a refined collection of Gaussians and projecting them. In order to describe how this is done, let us go back to the graph interpretation of a data cube. Each node N in a graph corresponds to a collection of points in the raw data. Hence, given a collection of nodes that corresponds to a partition of the original data points, we can project the Gaussian corresponding to each node to obtain the approximate scatterplot. It is clear that the final result will depend on the partition used. In order to obtain such a partition, we traverse the edges in the graph, which corresponds to performing “splits” in the original data. For any node N in the graph, there are several ways to “split” its sample set by selecting nodes which N connects to. For example, one possible choice from the example data cube in Figure 3 is to split on transmission types, which produces two nodes each with their own multivariate normals: each describing all cars with, respectively, manual and automatic transmissions. Another possible split would partition on car makes: Honda, BMW, and Ford. The final insight is to note that for any desired projection, one of these splits will produce a *better-resolved image*: the faster the variances reduce, the faster the projections are converging to projecting individual points (which would be the ideal outcome).

We are now ready to describe an algorithm to plot approximate scatterplots directly from a Gaussian Cube. We simply traverse the graph of a data cube progressively, using a priority queue to greedily split nodes which would reduce the total projected variance by the largest amount. The pseudo-code for the algorithm is in Figure 6. While we don’t provide any theoretical guarantees of the effectiveness of this algorithm, we find it works quite well in practice, as can be seen in the following sections.

Most importantly, this algorithm provides a way to generate plots with axes *outside* the indexing variable sets of a Gaussian Cube. To the best of our knowledge, this is a novel capability, enabled precisely because of the sufficient statistics stored as modeling variables. Figure 7 shows results obtained by using this algorithm on the SDSS dataset (described in Section 6.2).

6 EXPERIMENTS

Hardware and Backend Software All timing measurements in this section are reported from running Gaussian Cubes on a dual, six-core Intel Xeon E5 server with 256GB of RAM. Besides configuring the server to not perform any power-saving measures by dynamic clock setting, the system runs a stock version of Ubuntu 14.04. In particular, the machine is intermittently used by other projects, and so there are

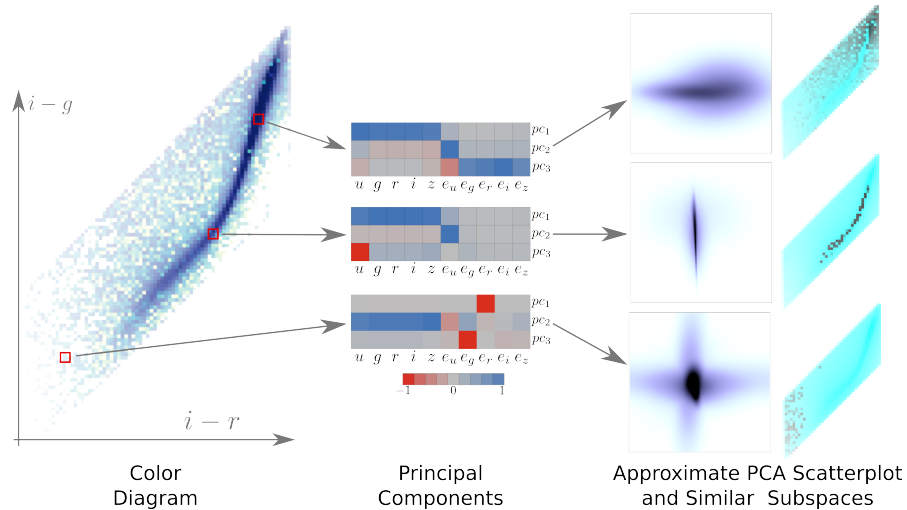


Fig. 8. Showcasing interactive exploration workflows enabled by Gaussian Cubes. The figure on the left shows a visualization of 51 million stars aggregated spatially in what astronomers call a *color diagram*: it shows that most of the visible stars follow a specific one-dimensional curve, the *stellar locus*. In this visualization, the hue corresponds to the average brightness of the stars in each bin. Users can select different *principal subspaces* (middle figure) by clicking on different parts of the image. The principal subspaces can be used to generate *approximate PCA plots* (see Section 5) and compute a colormap based on distances between the subspaces of each region of the plot, and get a visual clustering of places along the diagram where the internal variation of the set of stars is comparable.

occasional load spikes. We have made no attempt to control for these in our timing measurements. All binaries are compiled with g++ 4.8.5, using `-O3` as the only notable compilation flag.

Front End The geographic maps for the visualization front ends we build in this section come from the OpenStreetMap project [22], and are rendered using a slightly modified version of Leaflet [2].

In Table 1, we present a summary of the building time and memory usage of Gaussian Cubes used in our experiments.

6.1 Synthetic Dataset

We evaluate the correctness and performance of Gaussian Cubes by a synthetic dataset. Each of the entries in the dataset has two *key* dimensions and three *value* dimensions. The range of the two key dimensions are all in $[0, 10]$. The keys are sampled from three multivariate Gaussians. The means for the Gaussians are $[7, 2]$, $[2, 7]$ and $[2, 2]$ respectively. All of them have the same covariance matrix which is a diagonal matrix with the diagonal entries $[2, 2, 2]$. The *values* are sampled from different multivariate Gaussians. Specifically, for a data entry whose keys are x, y , the value dimensions a, b, c are sampled from a multivariate Gaussian $\mathcal{N}(m, \Sigma)$, where m is a 3×3 zero matrix and Σ is a diagonal matrix. The diagonal elements are $[x, y, 10 - |x - y|]$.

The synthetic dataset contains 1 million rows in our evaluation. It takes 15 seconds for Gaussian Cubes to load and process the whole dataset. The total memory usage for the 1-million dataset is 570MB. The experimental results are shown in Table 2. In the evaluation, we are building colormaps for the whole dataset based on the covariance matrix of each subset of the data. For example, the first colormap in row 1 shows the dataset is divided into $4 \times 4 = 16$ subsets. Then the covariance matrix is queried from Gaussian Cubes respectively. We only use the diagonal elements c_{00}, c_{11}, c_{22} of the covariance matrix for color mapping. Actually, they are just the variance of the three *value* dimensions. Then c_{00}, c_{11}, c_{22} are mapped to r, g and b respectively. So if c_{00} is large, r will have a large value. Although the second row of Table 2 shows that the query time grows significantly when the dataset is divided into more subsets, it should be pointed out that the JSON file size that is been transferred through the network is also increasing significantly (see the last row). So the network communication is actually dominating the query time. If we look at the query time per cell, it's even decreasing. This proves that we would be able to save much more query time if the query result was encoded in binary format.

6.2 Visualizing variability in the SDSS DR7 catalog

The Sloan Digital Sky Survey is one of the largest astronomical surveys ever undertaken. In this section, we use its seventh data release (“DR7” [1]) to showcase the ability of Gaussian Cubes to handle relatively high-dimensional data for its modeling schema. SDSS DR7 contains survey information of galaxies, quasars and stars. We only use stars in our experiments. DR7 includes a catalog of 180 million stars, where the brightness of each such star was measured at five different wavelengths, known collectively as *ugriz*. In addition to the individual wavelength measurements, DR7 includes an estimate of the error for each wavelength, for a total of 10 real-valued dimensions.

Data cleaning We filter out rows which have missing values in any of the 10 dimensions; other problems in data acquisition are recorded in DR7 as magnitudes of -9999 and 9999 . We filter these as well. The SDSS dataset includes a large amount of bad data. We use the ranges of magnitudes described by Narayan et al. in order to filter the data [36], ultimately yielding a total of 51,265,171 rows.

Data cube schema For most distant stars, it's essentially impossible to know whether they are far away or they shine weakly (these both produce the same photometric effects); as a result, astronomers focus on the differences between the magnitude measurements along different wavelengths (since this factors out the issue of absolute magnitudes). For this example, we use $i - r$ and $i - g$ as the values for the spatial dimension in the indexing variable set. This produces the elongated line we see in Figures 1 and 8. The spatial dimension uses a maximum depth of 15 for the quad-tree (for an effective resolution of 32768×32768), and we compute an additional linear dimension with binned values of $g - r$, using 10 possible bins. As modeling variables, we use all 10 values described above, for a total of 66 attributes in the data cube. The total memory consumption for the cleaned SDSS DR7 Stars is 12.8GB. It takes 21 minutes to build the Gaussian Cubes for it.

Visual clustering of subspaces Based on Gaussian Cubes, we build an interactive user interface to do visual clustering of the subspaces along which stars are distributed in the SDSS DR7. Specifically, an overall view of the whole dataset is shown by default (leftmost picture in Figure 8). Each cell in this overall view is selectable. When a user clicks on a given cell, we calculate the distance between this cell and each of the other cells. Then the default colormap is updated to show the distances as the shown in the right most pictures in Figure 8. In the new colormap, the cells that are close to the clicked cell will be dark brown; the cells that have large distance to the clicked

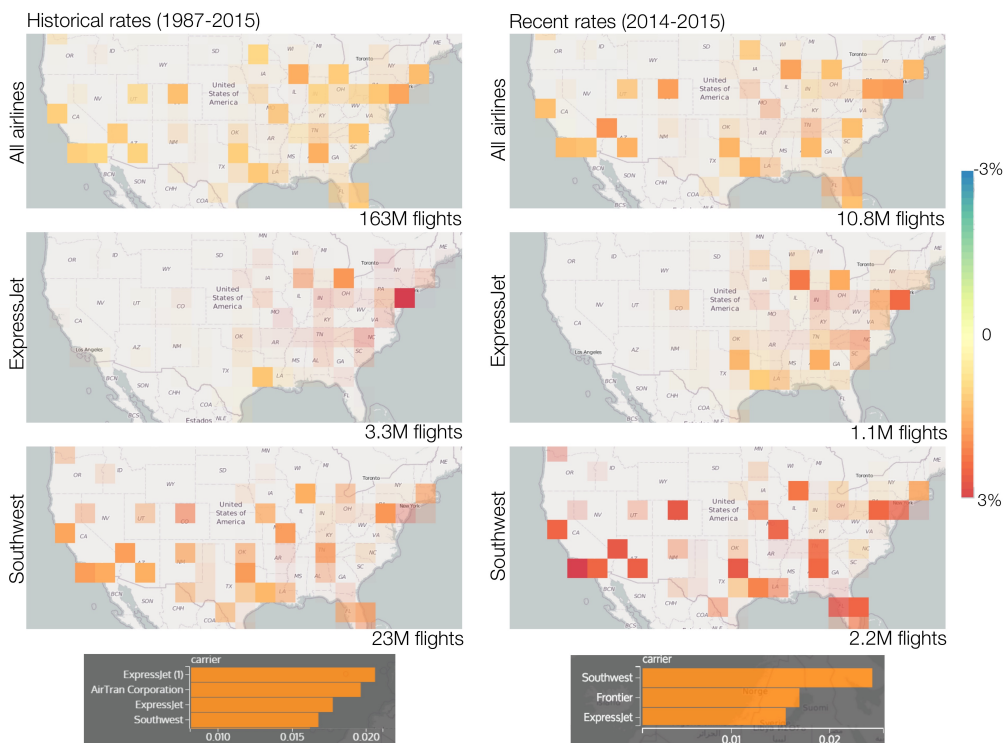


Fig. 9. By colormapping the rate at which flights become late during the day, we typically see snowballing effects specific to pairs of airlines and locations (such as ExpressJet’s case). In Southwest’s case, however, the effects are spread throughout the country. We have found evidence in the press that Southwest Airlines plans flight schedules differently from other companies, and that this difference may account for the effect [21].

cell will be light blue. We define the distances between the cells to be the distances between the *principal subspaces* of the samples. In our experiment, we choose the first three principal components of each cell. This choice was mostly arbitrary; different choices will produce different clusterings, but the general workflow is the same. Let P_0 be the principal subspace of the user clicked cell C_0 , and let P_k be the principal subspace of one of the other cells C_k . The matrices P are defined by computing the eigendecomposition $U\Sigma U^T$ for each cell, and then setting the first three diagonal elements of Σ to one, and the rest of the diagonal to zero. This gives a $d \times d$ projection. Then the distance between C_0 and C_k given by the operator norm of $P_k - P_0$ — the largest eigenvalue of the matrix $P_k - P_0$. As the user selects different subspaces to compare, Gaussian Cubes can update the plots in about 0.1s, much faster than would be possible by computing the covariance matrices by a linear scan of the 51 million stars in the catalog.

6.3 Flight Dataset

In this section, we use a dataset collected by the Bureau of Transportation Statistics, containing on-time performance information for commercial airlines for the past 25 years [9]. We will use it here to showcase the visualization of *regression coefficients*, and we refer the reader to Figure 9 for an illustration of our exploration.

The dataset contains 163,228,431 records and about 70 columns, many of which are redundantly encoded. For this example, the only spatial information we keep is the latitude and longitude of the arrival airport. Although the dataset itself contains only airport identifiers and not spatial information, we chose to perform spatial aggregation on airport locations (by joining the airport identifiers with a separate table containing the respective positions). We made this decision so that the hierarchical aggregation of the spatial dimension could be used for coarser models that would represent regional trends. We note that although the dataset includes flight departure and arrival information, in this example we discard departure information entirely. We did so because we wanted to highlight the novel capabilities of Gaussian Cubes; the ability to index on multiple spatial dimensions is a pre-existing feature of a recent version of nanocubes [31].

The schema for the Gaussian Cube we use in this section uses three indexing variables: a 25-bit spatial dimension encoding the latitude and longitude of the flight arrival, a categorical variable encoding 31 different airlines, and a time variable binned at 1 day resolution indicating the date of arrival of the flight. The modeling variable schema contains two variables: delay at arrival and flight arrival time. For computational convenience, we encode both of the modeling variables in fractions of a day.

We were personally interested in exploring the “snowball” effect which exists in flight data: as the day goes by, flight arrivals get progressively more late (we wish to acknowledge Bostock’s demo of Crossfilter as partial inspiration [8]). In our case, we model this effect as a simple linear relationship between flight delay and flight arrival: $d = at + b$ (we defer a discussion of the power of obviously wrong, but obviously simple models to Section 7), and we state that although the overall lateness of a flight is interesting, it is (for our case) less interesting than the *rate* in which flights become later. In other words, we need to include the intercept coefficient in order to capture an important aspect of the data, but we are ourselves interested in visualizing the *a* coefficient: the *slope* of the lateness curve.

The sufficient statistics for this model are the following sums: $\sum dt, \sum t^2, \sum t, \sum d, \sum 1$. Note that strictly speaking we do not need to compute $\sum d^2$ to fit this particular linear regression problem. However, for the sake of uniformity, the Gaussian Cube we create includes the additional term that is unused in this section.

Because we used the same units for flight arrival time and flight delay, the coefficient a in $d = at + b$ can be readily interpreted as a percentage: if the best-fitting slope is (say) 0.05, then on average, for that particular subset of data, every passing 60 minutes mean an expected delay of 3 minutes. This seems like a small number, but remember that this is an *average*, and so applies to every flight in the set, and it accumulates. At the end of day, a slope of 0.05 would mean that flights arriving at 8:00PM (assuming for now that flights never start the day delayed) would on average be a full hour behind schedule.

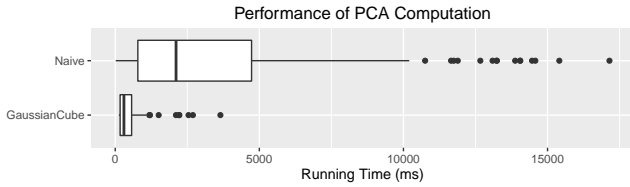


Fig. 10. Comparison of PCA calculation using naive approach and Gaussian Cubes. Multiple portions of the earthquake datasets are selected and the PCA is computed using a naive approach (implemented in Javascript) and Gaussian Cubes. As can be seen, Gaussian Cubes are significantly faster and can provide PCA results at interactive rates.

Visual data exploration We created a heatmap visualization where the color of each bin is decided based on the slope of the model which best fits the containing flights. Since the sufficient statistics are readily available from the results of queries into the Gaussian Cubes, the time to actually fit these models is negligible, and the overall performance is indistinguishable from traditional count-based visualizations. In addition to the slope of the best-fitting model, we use the total *size* of the sample to determine the bin’s opacity. In a sense, we are using sample size as a proxy for confidence in the model, and using opacity to hide model fits which are likely bad.

We concede that this use of sample size is needlessly naive. Nevertheless, this simple visualization readily yielded several interesting leads. First, it becomes apparent that most airlines have one specifically problematic airport. See, for example the middle row of images in Figure 9 for the case of ExpressJet; other companies have similar examples. We believe (but only checked cursorily) these are the airport hubs for the respective airlines, where flights are tightly scheduled and so snowballing is prone to happen. However, one company (and notably just one company) experiences this problem in a widespread fashion: Southwest Airlines. In addition, we found one particular instance (January 2014 in Chicago’s Midway Airport) in which Southwest flights were getting delayed at an average rate of upwards of 10% for a sustained period of about two weeks (see Figure 1). We found indications in the press that this was due to Southwest’s alleged practice of indefinitely delaying (but not canceling) flights, presumably to sidestep costs of rescheduling passenger flights. For this practice, Southwest Airlines was eventually fined over a million dollars [27].

6.4 Earthquake Simulations

Our final case study comes from an ongoing collaboration with civil engineers studying an ensemble of simulations of building stresses under earthquakes. In this project, we are interested in studying the interplay of different physical variables (moment, shear, etc.) on different portions of a building as it undergoes stresses because of an earthquake. In order to support the analysis of these variables, we built a web-based visualization system (the current user interface can be seen on the bottom-left of Figure 1). One of the important tasks that is performed by the domain experts in the process of studying this data is to perform PCA to understand the relationships between the multiple physical variables. The system uses Gaussian Cubes to interactively compute PCA over its variable set, as described in Section 4.2. In order to have an idea of the gains in computation speed obtained by using this approach, we compare it with a naive method in Figure 10. In this experiment, 162 different subcollections of varied sizes are selected from the earthquake dataset during the use of our system. For each subcollection, the PCA is computed using the naive approach, in which covariance matrices are computed for the selected portion of the data, and Gaussian Cubes. As shown in Figure 10, Gaussian Cubes achieve a significantly better performance which enables performing PCA at interactive rates.

7 DISCUSSION, LIMITATIONS, AND FUTURE WORK

We believe Gaussian Cubes offer a significant improvement over the state-of-the-art in exploratory model visualization. However, there are still many limitations and opportunities for improvement.

First of all, the supported models are limited. The model coverage

is determined by the preaggregated values. These include: descriptive statistics, confidence intervals, hypothesis tests, cross-tabulation analysis, analysis of variance, multivariate analysis of variance, linear regression, correlation analysis, principal component analysis, factor analysis, χ^2 analysis of independence [38]. We take residual calculation as a simple example. Assume we want to fit a linear model $y_i = mx_i + b$, that \hat{m}, \hat{b} are the solutions obtained from Gaussian Cubes, and \hat{E} is the residual of the best model. Then,

$$\begin{aligned} \hat{E} &= \sum (y_i - \hat{m}x_i - \hat{b})^2 \\ &= \sum y_i^2 + \hat{m}^2 \sum x_i^2 + \hat{b}^2 \sum 1 - 2\hat{m} \sum x_i y_i + 2\hat{m}\hat{b} \sum x_i - 2\hat{b} \sum y_i \end{aligned} \quad (5)$$

The sums are again prestored in Gaussian Cubes, and so the residual calculation can be done as fast as fitting a model. We want to make clear that it’s likely there are many other models that can be supported. We would like to investigate this in our future work. Secondly, the choice of indexing dimensions determines what kind of analysis could be provided. If the user wants to explore the dataset on any dimensions, Gaussian Cubes will require much more space building index on every dimension. This sacrifice in memory consumption might be acceptable if latency is truly unacceptable. Still, we want to note that a full treatment of the memory-query-time tradeoff for data cubes is an open research question that is beyond the scope of our proposal.

Currently, the process of matching models to visual encodings is manual and laborious. We envision a future class of visualization specifications a la MacKinlay’s classic APT[34] which would automatically take into account knowledge about the particular models being fit to derive appropriate classes of visual representations [30]. These might include glyphs [7], ensembles[26, 37], and other metaphors. Gaussian Cubes, in this context, enables this visualization technology to be used at larger scales than previously possible. While Gaussian Cubes do not currently incorporate perceptual knowledge in its backend, we believe it is possible to integrate perceptual constraints (in the sense of Wu et al.’s vision paper [43]) to influence the progressive scatterplot algorithm of Section 5. In a sense, we would seek to spend computational effort only if it would cause perceptual differences.

We also want to enable model evaluation. Currently, Gaussian Cubes only provide the fitted model without showing how well the model is fitted. A natural next step is to allow users to run model diagnoses, for example, providing exploratory, interactive visualization of residuals.

Leveraging Gaussian Cubes, we are able to build visualizations that, to our best knowledge, have never been attempted at this scale and low latency. An example is the approximate scatterplot proposed in Section 5. We see this as a powerful tool to generate density plots that can be used in a progressive manner [19] as hinted earlier. In fact, more refined versions of the plot can be produced by traversing the Gaussian Cube structure. This gives the user control of the time-accuracy trade-off and can be used to provide immediate feedback to the user. While these plots can reveal important structures on the data at a low computational costs, some artifacts of the approximation can be produced, see right most column in Figure 7. While the approximate (top) plot shows that most of the points are concentrated around 0 error (y-axis), the variance of the Gaussians creates a larger spread than the exact (bottom) plot. Furthermore, also due to the use of Gaussians as modeling distributions, the approximate plot suggests that negative error points might exist, which is not the case as shown in the exact plot. This is due to the symmetry of the Gaussian distribution around its mean. We note that, however crude the approximate plots might be, they are distinct enough from each other to highlight science-relevant aspects of the data. In a sense, we are replacing *ideal, impracticably slow* plots with *rough, practically useful* ones. Nevertheless, we believe further research is needed to understand to what extent users can benefit from these approximate plots and also precisely how these artifacts will influence the understanding of the data.

Finally, the implementation of Gaussian Cubes are an simple extension to Nanocubes. Still, the technology is easily applicable in other systems. In addition, we expect the adaptive traversal of a datacube to be applicable to a variety of other visualization and data mining algorithms, and this is an enticing avenue of future work.

Acknowledgments We wish to acknowledge Dr. Gautham Narayan at NOAO for answering our many questions about SDSS's Data Release 7, and Dr. Robert Fleischman, Ismail Bahadir and Zhi Zhang at UA's Department of Civil Engineering for access to their earthquake simulation data. In addition, we wish to acknowledge Prof. Cynthia Brewer's excellent colormaps, which we use extensively. We want to acknowledge AT&T Labs, and specifically their generous contribution of open-source software. Our implementation will soon be made available at our group website. This work was supported partially by NSF grants IIA-1344024 and III-1513651.

REFERENCES

- [1] K. N. Abazajian, J. K. Adelman-McCarthy, M. A. Agüeros, S. S. Allam, C. A. Prieto, D. An, K. S. Anderson, S. F. Anderson, J. Annis, N. A. Bahcall, et al. The Seventh Data Release of the Sloan Digital Sky Survey. *The Astrophysical Journal Supplement Series*, 182(2):543, 2009.
- [2] V. Agafonkin. Leaflet: An Open-source JavaScript Library for Mobile-friendly Interactive Maps, 2015. <http://leafletjs.com>.
- [3] 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 *Proceedings of the 8th ACM European Conference on Computer Systems*, pages 29–42. ACM, 2013.
- [4] A. Anand, L. Wilkinson, and T. N. Dang. Visual Pattern Discovery Using Random Projections. In *Visual Analytics Science and Technology (VAST), 2012 IEEE Conference on*, pages 43–52. IEEE, 2012.
- [5] M. Barnett, B. Chandramouli, R. DeLine, S. Drucker, D. Fisher, J. Goldstein, P. Morrison, and J. Platt. Stat!: An Interactive Analytics Environment for Big Data. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pages 1013–1016. ACM, 2013.
- [6] L. Battle, R. Chang, and M. Stonebraker. Dynamic Prefetching of Data Tiles for Interactive Visualization (to appear). In *Proceedings of the ACM SIGMOD Conference*. IEEE, 2016.
- [7] R. Borgo, J. Kehler, D. H. Chung, E. Maguire, R. S. Laramée, H. Hauser, M. Ward, and M. Chen. Glyph-based Visualization: Foundations, Design Guidelines, Techniques and Applications. *Eurographics State of the Art Reports*, pages 39–63, 2013.
- [8] M. Bostock. Crossfilter example: Airline on-time performance, 2012. <http://square.github.io/crossfilter> (last accessed Mar 31st 2016).
- [9] Bureau of Transportation Statistics. On-Time Performance. Available at http://www.transtats.bts.gov/Fields.asp?Table_ID=236. Last accessed Mar. 10th, 2016.
- [10] G. Casella and R. L. Berger. *Statistical Inference*, volume 2. Duxbury Pacific Grove, CA, 2002.
- [11] S.-M. Chan, L. Xiao, J. Gerth, and P. Hanrahan. Maintaining Interactivity While Exploring Massive Time Series. In *Visual Analytics Science and Technology, 2008. VAST'08. IEEE Symposium on*, pages 59–66. IEEE, 2008.
- [12] Y.-H. Chan, C. D. Correa, and K.-L. Ma. Regression Cube: A Technique for Multidimensional Visual Exploration and Interactive Pattern Finding. *ACM Transactions on Interactive Intelligent Systems (TiS)*, 4(1):7, 2014.
- [13] B.-C. Chen, L. Chen, Y. Lin, and R. Ramakrishnan. Prediction Cubes. In *Proceedings of the 31st international conference on Very large data bases*, pages 982–993. VLDB Endowment, 2005.
- [14] Y. Chen, G. Dong, J. Han, J. Pei, B. W. Wah, and J. Wang. Regression Cubes with Lossless Compression and Aggregation. *Knowledge and Data Engineering, IEEE Transactions on*, 18(12):1585–1599, 2006.
- [15] Y. Chen, G. Dong, J. Han, B. W. Wah, and J. Wang. Multi-dimensional Regression Analysis of Time-series Data Streams. In *Proceedings of the 28th international conference on Very Large Data Bases*, pages 323–334. VLDB Endowment, 2002.
- [16] J. A. Cottam, A. Lumsdaine, and P. Wang. Abstract Rendering: Out-of-Core Rendering for Information Visualization. In *IS&T/SPIE Electronic Imaging*, pages 90170K–90170K. International Society for Optics and Photonics, 2013.
- [17] G. H. Dunteman. *Principal Components Analysis*, volume 69. Sage, 1989.
- [18] N. Ferreira, D. Fisher, and A. C. König. Sample-oriented Task-driven Visualizations: Allowing Users to Make Better, More Confident Decisions. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 571–580. ACM, 2014.
- [19] D. Fisher, I. Popov, S. Drucker, et al. Trust me, I'm Partially Right: Incremental Visualization Lets Analysts Explore Large Datasets Faster. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1673–1682. ACM, 2012.
- [20] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatarao, F. Pellow, and H. Pirahesh. Data Cube: A Relational Aggregation Operator Generalizing Group-by, Cross-tab, and Sub-totals. *Data Mining and Knowledge Discovery*, 1(1):29–53, 1997.
- [21] A. Griswold. Southwest Airlines Has a Huge Lateness Problem, 2014. http://www.slate.com/blogs/moneybox/2014/08/08/southwest_airlines_delays_why_are_its_planes_late_so_often.html.
- [22] M. Haklay and P. Weber. Openstreetmap: User-generated Street Maps. *Pervasive Computing, IEEE*, 7(4):12–18, 2008.
- [23] T. Hastie, R. Tibshirani, J. Friedman, and J. Franklin. The Elements of Statistical Learning: Data Mining, Inference and Prediction. *The Mathematical Intelligencer*, 27(2):83–85, 2005.
- [24] J. M. Hellerstein, P. J. Haas, and H. J. Wang. Online Aggregation. In *ACM SIGMOD Record*, volume 26, pages 171–182. ACM, 1997.
- [25] A. E. Hoerl and R. W. Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.
- [26] M. Hummel, H. Obermaier, C. Garth, and K. I. Joy. Comparative Visual Analysis of Lagrangian Transport in CFD Ensembles. *Visualization and Computer Graphics, IEEE Transactions on*, 19(12):2743–2752, 2013.
- [27] G. Karp. Southwest Hit with Record Fine for Tarmac Delays at Midway, January 2015. The Chicago Tribune, Chicago.
- [28] D. Keim, G. Andrienko, J.-D. Fekete, C. Görg, J. Kohlhammer, and G. Melançon. *Visual Analytics: Definition, Process, and Challenges*. Springer, 2008.
- [29] D. A. Keim. Designing Pixel-oriented Visualization Techniques: Theory and Applications. *Visualization and Computer Graphics, IEEE Transactions on*, 6(1):59–78, 2000.
- [30] G. Kindlmann and C. Scheidegger. An Algebraic Process for Visualization Design. *Visualization and Computer Graphics, IEEE Transactions on*, 20(12):2181–2190, 2014.
- [31] L. Lins, J. T. Klosowski, and C. Scheidegger. Nanocubes for Real-time Exploration of Spatiotemporal Datasets. *Visualization and Computer Graphics, IEEE Transactions on*, 19(12):2456–2465, 2013.
- [32] Z. Liu and J. Heer. The Effects of Interactive Latency on Exploratory Visual Analysis. *Visualization and Computer Graphics, IEEE Transactions on*, 20(12):2122–2131, 2014.
- [33] Z. Liu, B. Jiang, and J. Heer. imMens: Real-time Visual Querying of Big Data. *Computer Graphics Forum (Proc. EuroVis)*, 32, 2013.
- [34] J. Mackinlay. Automating the Design of Graphical Presentations of Relational Information. *Acm Transactions On Graphics (Tog)*, 5(2):110–141, 1986.
- [35] A. Moore, J. Schneider, B. Anderson, S. Davies, P. Komarek, M. S. Lee, M. Meila, R. Munos, K. Myers, and P. Pelleg. Cached Sufficient Statistics for Automated Mining and Discovery from Massive Data Sources. *Robotics Institute*, page 258, 1999.
- [36] G. Narayan, A. Rest, B. E. Tucker, R. J. Foley, W. M. Wood-Vasey, P. Challis, C. W. Stubbs, R. P. Kirshner, C. Aguilera, A. C. Becker, et al. Light Curves of 213 Type Ia Supernovae from the ESSENCE Survey. *arXiv preprint arXiv:1603.03823*, 2016.
- [37] K. Potter, A. Wilson, P.-T. Bremer, D. Williams, C. Doutriaux, V. Pascucci, and C. R. Johnson. Ensemble-vis: A framework for the Statistical Visualization of Ensemble Data. In *Data Mining Workshops, 2009. ICDMW'09. IEEE International Conference on*, pages 233–240. IEEE, 2009.
- [38] S.-C. Shao. Multivariate and multidimensional olap. In *Advances in Database TechnologyEDBT'98*, pages 120–134. Springer, 1998.
- [39] A. Silberschatz, H. F. Korth, and S. Sudarshan. *Database System Concepts*, volume 4. McGraw-Hill New York, 1997.
- [40] Y. Sismanis, A. Deligiannakis, N. Roussopoulos, and Y. Kotidis. Dwarf: Shrinking the Petacube. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 464–475. ACM, 2002.
- [41] C. Stolte, D. Tang, and P. Hanrahan. Polaris: A System for Query, Analysis, and Visualization of Multidimensional Relational Databases. *Visualization and Computer Graphics, IEEE Transactions on*, 8(1):52–65, 2002.
- [42] C. Weaver. Conjunctive Visual Forms. *Visualization and Computer Graphics, IEEE Transactions on*, 15(6):929–936, 2009.
- [43] E. Wu, L. Battle, and S. R. Madden. The Case for Data Visualization Management Systems: Vision Paper. *Proceedings of the VLDB Endowment*, 7(10):903–906, 2014.
- [44] R. Xi, N. Lin, and Y. Chen. Compression and Aggregation for Logistic Regression Analysis in Data Cubes. *Knowledge and Data Engineering, IEEE Transactions on*, 21(4):479–492, 2009.