Quantitative Data Cleaning for Large Databases

Joseph M. Hellerstein*
EECS Computer Science Division
UC Berkeley
http://db.cs.berkeley.edu/jmh

February 27, 2008

1 Introduction

Data collection has become a ubiquitous function of large organizations – not only for record keeping, but to support a variety of data analysis tasks that are critical to the organizational mission. Data analysis typically drives decision-making processes and efficiency optimizations, and in an increasing number of settings is the *raison d'etre* of entire agencies or firms.

Despite the importance of data collection and analysis, data *quality* remains a pervasive and thorny problem in almost every large organization. The presence of incorrect or inconsistent data can significantly distort the results of analyses, often negating the potential benefits of information-driven approaches. As a result, there has been a variety of research over the last decades on various aspects of *data cleaning*: computational procedures to automatically or semi-automatically identify – and, when possible, correct – errors in large data sets.

In this report, we survey data cleaning methods that focus on errors in *quantitative* attributes of large databases, though we also provide references to data cleaning methods for other types of attributes. The discussion is targeted at computer practitioners who manage large databases of quantitative information, and designers developing data entry and auditing tools for end users. Because of our focus on quantitative data, we take a statistical view of data quality, with an emphasis on intuitive outlier detection and exploratory data analysis methods based in *robust statistics* [Rousseeuw and Leroy, 1987, Hampel et al., 1986, Huber, 1981]. In addition, we stress algorithms and implementations that can be easily and efficiently implemented in very large databases, and which are easy to understand and visualize graphically. The discussion mixes statistical intuitions and methods, algorithmic building blocks, efficient relational database implementation strategies, and user interface considerations. Throughout the discussion, references are provided for deeper reading on all of these issues.

1.1 Sources of Error in Data

Before a data item ends up in a database, it typically passes through a number of steps involving both human interaction and computation. Data errors can creep in at every step of the process from initial data acquisition to archival storage. An understanding of the sources of data errors can be useful both in designing data collection and curation techniques that mitigate

*This survey was written under contract to the United Nations Economic Commission for Europe (UNECE), which holds the copyright on this version.

the introduction of errors, and in developing appropriate post-hoc data cleaning techniques to detect and ameliorate errors. Many of the sources of error in databases fall into one or more of the following categories:

- **Data entry errors:** It remains common in many settings for data entry to be done by humans, who typically extract information from speech (e.g., in telephone call centers) or by keying in data from written or printed sources. In these settings, data is often corrupted at entry time by typographic errors or misunderstanding of the data source. Another very common reason that humans enter "dirty" data into forms is to provide what we call *spurious integrity*: many forms require certain fields to be filled out, and when a data-entry user does not have access to values for one of those fields, they will often invent a default value that is easy to type, or that seems to them to be a typical value. This often passes the crude data integrity tests of the data entry system, while leaving no trace in the database that the data is in fact meaningless or misleading.

- **Measurement errors:** In many cases data is intended to measure some physical process in the world: the speed of a vehicle, the size of a population, the growth of an economy, etc. In some cases these measurements are undertaken by human processes that can have errors in their design (e.g., improper surveys or sampling strategies) and execution (e.g., misuse of instruments). In the measurement of physical properties, the increasing proliferation of sensor technology has led to large volumes of data that is never manipulated via human intervention. While this avoids various human errors in data acquisition and entry, data errors are still quite common: the human design of a sensor deployment (e.g., selection and placement of sensors) often affects data quality, and many sensors are subject to errors including miscalibration and interference from unintended signals.

- **Distillation errors:** In many settings, raw data are preprocessed and summarized before they are entered into a database. This data distillation is done for a variety of reasons: to reduce the complexity or noise in the raw data (e.g., many sensors perform smoothing in their hardware), to perform domain-specific statistical analyses not understood by the database manager, to emphasize aggregate properties of the raw data (often with some editorial bias), and in some cases simply to reduce the volume of data being stored. All these processes have the potential to produce errors in the distilled data, or in the way that the distillation technique interacts with the final analysis.

- **Data integration errors:** It is actually quite rare for a database of significant size and age to contain data from a single source, collected and entered in the same way over time. In almost all settings, a database contains information collected from multiple sources via multiple methods over time. Moreover, in practice many databases evolve by merging in other pre-existing databases; this merging task almost always requires some attempt to resolve inconsistencies across the databases involving data representations, units, measurement periods, and so on. Any procedure that integrates data from multiple sources can lead to errors.

1.2 Approaches to Improving Data Quality

The "lifetime" of data is a multi-step and sometimes iterative process involving collection, transformation, storage, auditing, cleaning and analysis. Typically this process includes people and equipment from multiple organizations within or across agencies, potentially over large spans of time and space. Each step of this process can be designed in ways that can encourage data quality. While the bulk of this report is focused on post-hoc data auditing and cleaning, here we mention a broad range of approaches that have been suggested for maintaining or improving data quality:

- **Data entry interface design.** For human data entry, errors in data can often be mitigated through judicious design of data entry interfaces. Traditionally, one key aspect of this was the specification and maintenance of database *integrity constraints*, including data type checks, bounds on numeric values, and referential integrity (the prevention of references to non-existent data). When these integrity constraints are enforced by the database, data entry interfaces prevent data-entry users from providing data that violates the constraints. An unfortunate side-effect of this *enforcement* approach is the spurious integrity problem mentioned above, which frustrates data-entry users and leads them to invent dirty data. An alternative approach is to provide the data-entry user with convenient affordances to understand, override and explain constraint violations, thus discouraging the silent injection of bad data, and encouraging annotation of surprising or incomplete source data. We discuss this topic in more detail in Section 7.
- **Organizational management.** In the business community, there is a wide-ranging set of principles regarding organizational structures for improving data quality, sometimes referred to as *Total Data Quality Management*. This work tends to include the use of technological solutions, but also focuses on organizational structures and incentives to help improve data quality. These include streamlining processes for data collection, archiving and analysis to minimize opportunities for error; automating data capture; capturing metadata and using it to improve data interpretation; and incentives for multiple parties to participate in the process of maintaining data quality [Huang et al., 1999].
- **Automated data auditing and cleaning.** There are a host of computational techniques from both research and industry for trying to identify and in some cases rectify errors in data. There are many variants on this theme, which we survey in Section 1.3.
- **Exploratory data analysis and cleaning.** In many if not most instances, data can only be cleaned effectively with some human involvement. Therefore there is typically an interaction between data cleaning tools and data visualization systems. Exploratory Data Analysis [Tukey, 1977] (sometimes called Exploratory Data Mining in more recent literature [Dasu and Johnson, 2003]) typically involves a human in the process of understanding properties of a dataset, including the identification and possible rectification of errors. Data *profiling* is often used to give a big picture of the contents of a dataset, alongside metadata that describes the possible structures and values in the database. Data visualizations are often used to make statistical properties of the data (distributions, correlations, etc.) accessible to data analysts.

In general, there is value to be gained from all these approaches to maintaining data quality. The prioritization of these tasks depends upon organizational dynamics: typically the business management techniques are dictated from organizational leadership, the technical analyses must be chosen and deployed by data processing experts within an Information Technology (IT) division, and the design and rollout of better interfaces depends on the way that user tools are deployed in an organization (e.g., via packaged software, downloads, web-based services, etc.) The techniques surveyed in this report focus largely on technical approaches that can be achieved within an IT organization, though we do discuss interface designs that would involve user adoption and training.

1.3 Data Cleaning: Types and Techniques

Focusing more specifically on post-hoc data cleaning, there are many techniques in the research literature, and many products in the marketplace. (The KDDNuggets website [Piatetsky-Shapiro, 2008] lists a number of current commercial data cleaning tools.) The space of techniques and products can be categorized fairly neatly by the types of data that they target. Here we provide a brief overview of data cleaning techniques, broken down by data type.

- **Quantitative data** are integers or floating point numbers that measure quantities of interest. Quantitative data may consist of simple sets of numbers, or complex arrays of data in multiple dimensions, sometimes captured over time in *time series*. Quantitative data is typically based in some unit of measure, which needs to be uniform across the data for analyses to be meaningful; unit conversion (especially for volatile units like currencies) can often be a challenge. Statistical methods for *outlier detection* are the foundation of data cleaning techniques in this domain: they try to identify readings that are in some sense "far" from what one would expect based on the rest of the data. In recent years, this area has expanded into the more recent field of data mining, which emerged in part to develop statistical methods that are efficient on very large data sets.

- **Categorical data** are names or codes that are used to assign data into categories or groups. Unlike quantitative attributes, categorical attributes typically have no natural ordering or distance between values that fit quantitative definitions of outliers. One key data cleaning problem with categorical data is the mapping of different category names to a uniform namespace: e.g., a "razor" in one data set may be called a "shaver" in another, and simply a "hygiene product" (a broader category) in a third. Another problem is identifying the miscategorization of data, possibly by the association of values with "lexicons" of known categories, and the identification of values outside those lexicons [Raman and Hellerstein, 2001]. Yet another problem is managing data entry errors (e.g. misspellings and typos) that often arise with textual codes. There are a variety of techniques available for handling misspellings, which often adapt themselves nicely to specialized domains, languages and lexicons [Gravano et al., 2003].

- **Postal Addresses** represent a special case of categorical data that is sufficiently important to merit its own software packages and heuristics. While postal addresses are often free text, they typically have both structure and intrinsic redundancy. One challenge in handling postal address text is to make sure any redundant or ambiguous aspects are consistent and complete – e.g. to ensure that street addresses and postal codes are consistent, and that the street name is distinctive (e.g., "100 Pine, San Francisco" vs. "100 Pine Street, San Francisco".). Another challenge is that of *deduplication*: identifying duplicate entries in a mailing list that differ in spelling but not in the actual recipient. This involves not only canonicalizing the postal address, but also deciding whether two distinct addressees (e.g. "J. Lee" and "John Li") at the same address are actually the same person. This is a fairly mature area, with commercial offerings including Trillium, QAS, and others [Piatetsky-Shapiro, 2008], and a number of approaches in the research literature (e.g., [Singla and Domingos, 2006], [Bhattacharya and Getoor, 2007], [Dong et al., 2005]). The U.S. Bureau of the Census provides a survey article on the topic [Winkler, 2006].

- **Identifiers** or *keys* are another special case of categorical data, which are used to uniquely name objects or properties. In some cases identifiers are completely arbitrary and have no semantics beyond being uniquely assigned to a single object. However, in many cases identifiers have domain-specific structure that provides some information: this is true of telephone numbers, UPC product codes, United States Social Security numbers, Internet Protocol addresses, and so on. One challenge in data cleaning is to detect the reuse of an identifier across distinct objects; this is a violation of the definition of an identifier that requires resolution. Although identifiers should by definition uniquely identify an object in some set, they may be repeatedly stored within other data items as a form of *reference* to the object being identified. For example, a table of taxpayers may have a unique tax ID per object, but a table of tax payment records may have many entries per taxpayer, each of which contains the tax ID of the payer to facilitate linking the payment with information about the payer. *Referential integrity* is the property of ensuring that all references of this form contain values that actually appear in the set of objects to which they refer. Identifying referential integrity failures is an example of finding *inconsistencies* across data items in a data set. More general integrity failures can be defined using the relational database theory of functional dependencies. Even when such dependencies (which include referential integrity as a subclass) are not enforced, they can be "mined" from data, even in the presence of failures [Huhtala et al., 1999].

This list of data types and cleaning tasks is not exhaustive, but it does cover many of the problems that have been a focus in the research literature and product offerings for data cleaning.

Note that typical database data contains a mixture of attributes from all of these data types, often within a single database table. Common practice today is to treat these different attributes separately using separate techniques. There are certainly settings where the techniques can complement each other, although this is relatively unusual in current practice.

1.4 Emphasis and Outline

The focus of this report is on post-hoc data cleaning techniques for quantitative attributes. We begin in Section 2 by considering outlier detection mechanisms for single quantitative attributes based on robust statistics. Given that background, in Section 3 we discuss techniques for handling multiple quantitative attributes together, and in Section 4 we discuss quantitative data in timeseries. Section 5 briefly presents alternative methodologies to the robust estimators presented previously, based on the idea of resampling. In Section 6, we shift from the identification of outlying values, to outlying counts or frequencies: data values that are repeated more frequently than normal. In Section 7 we turn our attention to softer issues in the design of interfaces for both data entry and the exploratory side of data cleaning. Throughout, our focus is on data cleaning techniques that are technically accessible and feasibly implemented by database professionals; we also provide algorithms and implementation guidelines and discuss how to integrate them with modern relational databases. While keeping this focus, we also provide references to additional foundational and recent methods from statistics and data mining that may be of interest for further reading.

2 Univariate Outliers: One Attribute at a Time

The simplest case to consider – and one of the most useful – is to analyze the set of values that appear in a single column of a database table. Many sources of dirty quantitative data are discoverable by examining one column at a time, including common cases of mistyping and the use of extreme default values to achieve spurious integrity on numeric columns.

This single-attribute, or *univariate*, case provides an opportunity to introduce basic statistical concepts in a relatively intuitive setting. The structure of this section also sets the tone for the next two sections to follow: we develop a notion of outliers based on some intuitive statistical properties, and then describe analogs to those properties that can remain "robust" even when significant errors are injected into a large fraction of the data.

2.1 A Note on Data, Distributions, and Outliers

Database practitioners tend to think of data in terms of a collection of information intentionally input into a computer for record-keeping purposes. They are often interested in *descriptive statistics* of the data, but they tend to make few explicit assumptions about why or how the values in the data came to be.

By contrast, statisticians tend to think of a collection of data as a *sample* of some data-generating process. They often try to use the data to find a statistical description or *model* of that process that is simple, but fits the data well. For example, one standard statistical model is *linear regression*, which in the simple two-dimensional case attempts to "fit a line" to the data (Figure 1). Given such a model, they can describe the likelihood (or the "surprise") of a data point with respect to that model in terms of probabilities. They can also use the model to provide probabilities for values that have not been measured: this is useful when data is missing (e.g., in data *imputation*), or has yet to be generated (e.g., in *forecasting*). Model-based approaches are often called *parametric*, since the parameters of a mathematical model describe the data concisely. Approaches that do not require a model are sometimes called *nonparametric* or *model-free*.

Outlier detection mechanisms have been developed from both these points of view, and model-free approaches that may appeal to database practitioners often have natural analogs in statistical models, which can be used to provide additional probabilistic tools.

In practice, most database practitioners do not implicitly view data in a strictly model-free fashion. They often use summary statistics like means and standard deviations when analyzing data sets for outliers. As we discuss shortly in Section 2.2, this assumes a model based on the *normal* distribution, which is the foundation for much of modern statistics.

We will see below that outlier detection techniques can be attacked using both model-free and model-based approaches. We will move between the two fairly often during our discussion, bringing up the distinctions as we go.

2.2 Characterizing a Set of Values: Center and Dispersion

It can be difficult to define the notion of an outlier crisply. Given a set of values, most data analysts have an intuitive sense of when some of the values are "far enough" from "average" that they deserve extra scrutiny. There are various ways to make this notion concrete, which rest on defining specific metrics for the *center* of the set of values (what is "average") and the *dispersion* of the set (which determines what is "far" from average, in a relative sense).

The center, or core, of a set of values is some "typical" value that may or may not appear in the set. The most familiar center metric is the *mean* (average) of the values, which typically is not one of the values in the set[1]. We will discuss other choices of center metrics in the remainder of this report.

[1] In statistical terminology, the average of the set of data values is called a *sample mean*; the true mean is defined with respect to the statistical distribution of the model generating those data values. Similarly below, when we speak of the standard deviation, it is the sample standard deviation we are discussing.

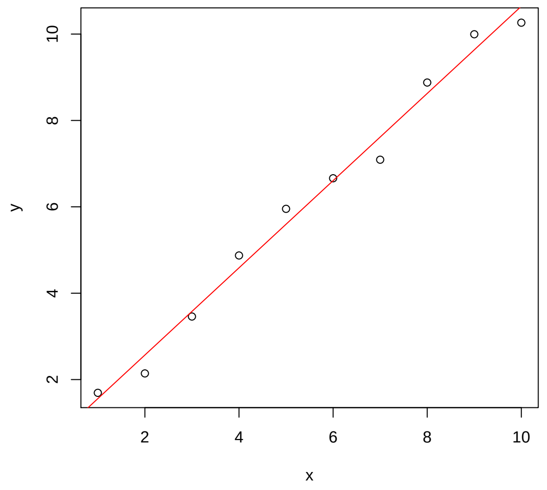


Figure 1: Linear regression applied to a simple two-dimensional dataset.

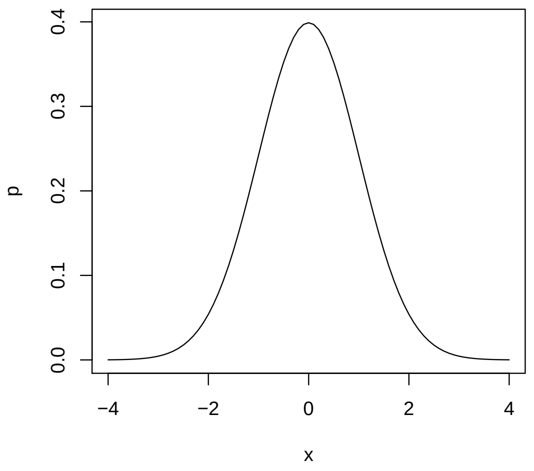


Figure 2: Probability density function for a normal ("Gaussian") distribution with mean 0 and standard deviation 1. The y axis shows the probability of each x value; the area under the curve sums to 1.0 (100%).

The dispersion, or spread, of values around the center gives a sense of what kinds of deviation from the center are common. The most familiar metric of dispersion is the *standard deviation*, or the *variance*, which is equal to the standard deviation squared. Again, we will discuss other metrics of dispersion below.

Our "center/dispersion" intuition about outliers defines one of the most familiar ideas in statistics: the *normal distribution*, sometimes called a *Gaussian distribution*, and familiarly known as the *bell curve* (Figure 2.) Normal distributions are at the heart of many statistical techniques, especially those that focus on measuring the variation of errors. The normal distribution is defined by a *mean* value μ and a standard deviation σ, and has the probability density function

$$p(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

Plotting that equation yields a characteristic bell curve like that of Figure 2, where the vertical axis roughly shows the probability of each value on the horizontal axis occurring[2]. While the normal distribution is not a good model for all data, it is a workhorse of modern statistics due both to certain clean mathematical properties, and the fact (based on the famous Central Limit Theorem of statistics) that it arises when many small, independent effects are added together – a very common occurrence.

Beyond the center and dispersion, a third class of metrics that is often discussed is the *skew* of the values, which describes how symmetrically the data is dispersed around the center. In very skewed data, one side of the center has a much longer "tail" than the other. We will return

[2]Technically, a continuous probability density function like the normal should be interpreted in terms of the *area* under the curve in a *range* on the horizontal axis – i.e., the probability that a sample of the distribution falls within that range.

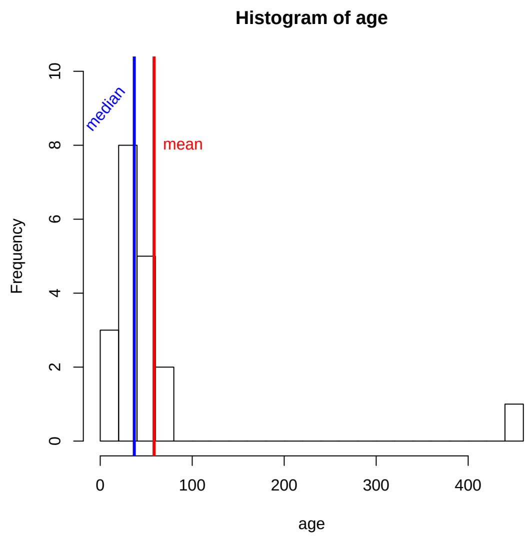


Figure 3: Histogram for example ages; median and mean points are labeled.

to the notion of skew in Section 6.

2.3 Intuition on Outliers and Robust Statistics

Our motivation in defining metrics for the center and dispersion of a data set is to identify *outliers*: values that are "far away" from the center of the data, where distance is measured in terms of the dispersion. For example, a typical definition of an outlier is any value that is more than 2 standard deviations from the mean. But this definition raises a basic question. If the mean and standard deviation are themselves computed over the entire data set, then aren't they "dirtied" to some degree by the very outliers we are using them to detect?

To get some quick intuition on this problem, consider the following set of numbers, corresponding to the ages of employees in a U.S. company:

12 13 14 21 22 26 33 35 36 37 39 42 45 47 54 57 61 68 450

A histogram of the data is shown in Figure 3.

Our knowledge about U.S. child labor laws and human life expectancy might suggest to us that the first three and last one of these numbers are errors. But an automatic procedure does not have the benefit of this knowledge. The mean of these numbers is approximately 59, and the standard deviation approximately 96. So a simple procedure that flags values more than 2 standard deviations from the mean would exclude data outside the range $[96-2*59, 96+2*59] = [-22, 214]$. It would *not* flag the first three values as outliers. This effect is called "masking": the magnitude of one outlier shifts the center and spread metrics enough to mask other outliers. This is a critical problem in data cleaning, where values can easily be off by orders of magnitude due to data entry errors (pressing a key twice instead of once), incorrect use of units, and other effects.

Robust Statistics is a subfield that considers the effect of corrupted data values on distributions, and develops estimators that are robust to such corruptions. Robust estimators can capture important properties of data distributions in a way that is stable in the face of many corruptions of the data, even when these corruptions result in arbitrarily bad values. When the percentage of corruptions in a data set exceeds a threshold called the *breakdown point* of an estimator, the estimator can produce arbitrarily erroneous results. Note that the mean described above can break down with a single bad value. However, the breakdown points for robust estimators can have high breakdown points – as high as 50% of the data. (When more than half the data is corrupted, it is not possible to distinguish true data from outliers.) A main theme in the remainder of this report will be the use of robust estimators for detecting outliers.

2.4 Robust Centers

To avoid the masking effect we saw with computing means and standard deviations, we turn our attention to robust estimators. We begin with robust metrics of the center or "core" of the data.

The *median* of a data set is that item for which half the values are smaller, and half are larger. (For data sets with an even number of values $2n$, the median is defined to be the average of the two "middle" values – those two values for which $n-1$ are smaller and $n-1$ are larger.) The median is an optimally robust statistic: it has a breakdown point of 50%, meaning that more than half of the data have to be corrupted before the median is shifted by an arbitrary amount. Returning to our previous example of employee ages, the median value was 37, which seems a lot more representative than the mean value of 96. Now, consider what happens when we "repair" the data by getting the true values of the outliers (12, 13, 14, 450), which should have been (21,31,41,45) respectively. The resulting set of ages is:

21 21 22 26 31 33 35 36 37 39 41 42 45 45 47 54 57 61 68

The new mean is about 40, and the new median is 39. Clearly the median was more stable with respect to the corruptions. To understand this, notice that the choice of the median element was affected by the *positions* of the outliers in the sort order (whether they were lower or higher than the median before and after being repaired). It was *not* affected by the values of the outliers. In particular suppose that the original value 450 had instead been 450,000,000 – the behavior of the median before and after data repair would not have changed, whereas the mean would have grown enormously before repair[3].

Another popular robust center metric is the *trimmed mean*. The k% trimmed mean is computed by discarding the lowest and highest k% of the values, and computing the mean of the remaining values[4]. The breakdown point of the k% trimmed mean is k%: when more than k% of the values are set to be arbitrarily bad, then one of those values can affect the trimmed

[3]The robust estimators we consider in this report are all based on position or *rank* in this way; they are referred to as L-estimators in the Robust Statistics literature. Another popular class of robust statistics are called M-estimators, which are based on model-based maxiumum-likelihood techniques not unlike our linear regression discussion in Section 2.1. These are somewhat less intuitive for most database professionals, and less natural to implement over SQL databases, but widely studied and used in the statistics community. There are a number of other classes of robust estimators as well. The interested reader is referred to textbooks on Robust Statistics for a more thorough discussion of these estimators [Rousseeuw and Leroy, 1987, Hampel et al., 1986, Huber, 1981].

[4]When k% of the dataset's size is not an integer, the trimmed mean is often computed by finding the two nearest trimmed means (the next lower and higher values of k that produce integers) and averaging the result. For example, the 15% trimmed mean of 10 numbers is computing by averaging the 10% trimmed mean and the 20% trimmed mean. Alternatively, one can round to an integer number of values to trim.

mean arbitrarily. A variant of the trimmed mean is the *winsorized* mean, in which the extreme values are not dropped, they are instead set to be the value of the lowest (or highest) included value. Observe that trimming or winsorizing with $k \approx 50\%$ is equivalent to computing the median: only the median value remains after trimming or winsorizing, and the resulting mean is equal to the median value.

Looking at our original "dirty" age data, the 10% trimmed mean is a bit more than 39.4, and the 10% winsorized mean is a bit more than 39.5.

It should be clear that these center metrics are more robust to outliers than the mean. A more difficult question is to decide which of these metrics to choose in a given scenario. As a rule of thumb, one can remember that (a) the median is the "safest" center metric, due to its 50% breakdown point, but (b) smaller-k trimmed/winsorized means are computed from more data than the median, and hence when they do not break down they are more tightly fit to the data.

As regards the choice between trimming and winsorization, winsorizing does put more weight on the edges of the distribution. This makes it a better choice for normally-distributed (bell-curve) datasets, whereas skewed datasets with "long tails" are more safely handled by trimming, since winsorizing can amplify the influence of unwinsorized outlying values in the tails.

2.4.1 Special Cases: Rates and Indexes

In any discussion of center metrics, there are a few special cases to keep in mind. We discuss two important ones here.

Given a collection of rates or (normalized) indexes, it can be misleading to use traditional center metrics. For example, suppose the rate of inflation of a currency over a series of years is:

$$1.03 \quad 1.05 \quad 1.01 \quad 1.03 \quad 1.06$$

Given an object worth 10 units of the currency originally, its final value would be:

$$10 * 1.03 * 1.05 * 1.01 * 1.03 * 1.06 = 11.926 \text{ units}$$

Now, in computing a center metric μ for numbers like rates, it would be natural to pick one that would lead to the same answer if it were substituted in for the actual rates above:

$$10 * \mu * \mu * \mu * \mu * \mu = 10\mu^5 = 11.926$$

This is not true of the traditional (arithmetic) mean, as one can verify on the example. The *geometric mean* is defined by this desired property: given n numbers $k_1 \ldots k_n$, the geometric mean is defined as the nth root of the product of the numbers:

$$\sqrt[n]{\left(\prod_{i=1}^{n} k_i\right)}$$

In practice, the geometric mean is not greatly affected by outlier values on the high end. However it is sensitive to values close to 0, which can pull the geometric mean down arbitrarily; as a result, the geometric mean is not a robust statistic in general.

In some computations of rates, the *harmonic mean* is the appropriate center metric. The canonical example is when computing average speeds over a fixed distance. Supposing you take a trip, traveling 50 kilometers at 10 kph, and 50 kilometers at 50 kph; you will travel 100 kilometers in 6 hours (5 hours at 10kph, one hour at 50 kph). One would like the "average" speed μ of the 100k trip to be computed as 100k/6hr = 16.67kph. The general form of this computation on values $k_1 \dots k_n$ is the harmonic mean:

$$\frac{n}{\sum_{i=1}^{n} \frac{1}{k_i}}$$

i.e. the reciprocal of the average of reciprocals of the speeds ($\frac{2}{1/10+1/50}$ in our example). Harmonic means are also useful for representing "average" sample sizes across experiments. In general, they are appropriate when the numbers being aggregated have weights associated with them; indeed, the harmonic mean is equivalent to a weighted arithmetic mean with each value's weight being the reciprocal of the value. Like geometric means, harmonic means are sensitive to values close to 0, and are not robust.

In order to make the geometric and harmonic means more robust, trimming can be used. Winsorization does not translate directly to these measures, as the weight of the values being "substituted in" depends on the value. Instead, different substitution rules have been proposed. For geometric means, some practitioners propose substituting in the value 1 (100%), or some value that represents 1/2 of the smallest measurable value, depending on the application. Clearly these approaches can affect estimations significantly, and need to be chosen carefully – they are not well suited to automatic data cleaning scenarios.

In general, a useful rule to know is that for any set of numbers, the harmonic mean is always less than or equal to the geometric mean, which in turn is always less than or equal to the arithmetic mean. So in the absence of domain information, it can be instructive to compute all three (or a robust version of all three).

2.5 Robust Dispersion

Having established some robust metrics for the "center" or "core" of a distribution, we also would like robust metrics for the "dispersion" or "spread" of the distribution as well.

Recall that the traditional standard deviation is defined in terms of each value's distance from the (arithmetic) mean:

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(k_i - \mu)^2}.$$

where μ is the arithmetic mean of the values $k_1 \dots k_n$.

When using the median as a center metric, a good robust metric of dispersion is the *Median Absolute Deviation* or *MAD*, which is a robust analogy to the standard deviation: it measures the median distance of all the values from the median value:

$$\underset{i}{\mathrm{MAD}}\{k_i\} = \underset{i}{\mathrm{median}}\{|k_i - \underset{j}{\mathrm{median}}\{k_j\}|\}$$

When using the trimmed (resp. winsorized) mean as the center metric, the natural dispersion metric is the trimmed (winsorized) standard deviation, which is simply the standard deviation of the trimmed (winsorized) data.

2.6 Putting It All Together: Robust Univariate Outliers

Having defined robust definitions of center and dispersion metrics, we can now fairly naturally come up with robust definitions of an *outlier*: a value that is too far away (as defined by the robust dispersion metric) from the center (as defined by the robust center metric).

The median and MAD lead to a robust outlier detection technique known as *Hampel X84* which is considered quiet reliable in the face of many outliers because it can be shown to have an ideal breakdown point of 50%. A simple version of Hampel X84 labels as outliers any data points that are more than $1.4826x$ MADs away from the median, where x is the number of standard deviations away from the mean one would have used in the absence of outliers[5]. For example, we used 2 standard deviations from the mean in Section 2.3, so we would want to look $1.4826 * 2 = 2.9652$ MADs from the median. Recall the example data:

$$12 \quad 13 \quad 14 \quad 21 \quad 22 \quad 26 \quad 33 \quad 35 \quad 36 \quad 37 \quad 39 \quad 42 \quad 45 \quad 47 \quad 54 \quad 57 \quad 61 \quad 68 \quad 450$$

The median is 39, and the MAD is 15. So the Hampel x84 outliers are numbers outside the range $[39 - 2.9652 * 15, 39 + 2.9652 * 15.] = [-5.478, 83.478]$. This still does not pick up the first three outliers. But if we compare it to the mean/standard-deviation range of Section 2.3, which was $[-22, 214]$, the robust methods cover many fewer unlikely ages. To emphasize this, consider the case of one standard deviation, corresponding to 1.4826 MADs. The non-outlier range using Hampel X84 is $[39 - 1.4826 * 15, 39 + 1.4826 * 15] = [16.761, 61.239]$, while the mean/standard-deviation range is $[37, 155]$. In this case, both techniques catch the outliers, but the robust estimators flag only one non-outlier incorrectly (68), while the mean/standard-deviation approach flags six non-outliers incorrectly $(21, 22, 26, 33, 35, 36)$.

Another approach is to trim or winsorize the data and compute means and standard deviations on the result. To briefly consider trimming our example, note that we have 19 values, so trimming the bottom and top value corresponds to approximately a 5% trim. The trimmed mean and standard deviation that result are approximately 38.24 and 16.05. Two standard deviations from the mean gives us the range $[6.14, 70.34]$; one standard deviation from the mean gives us the range $[22.19, 54.29]$.

2.7 Database Implementations of Order Statistics

Relational databases traditionally offer data *aggregation* facilities in the SQL language. The built-in SQL aggregation functions include:

- `MAX`, the maximum value in a column,
- `MIN`, the minimum value in a column,
- `COUNT`, the count of rows,
- `SUM`, the sum of the values in a column,
- `AVG`, the (sample) mean of all values in a column,
- `STDEVP`, the (sample) standard deviation of all values in a column,
- `VARP`, the (sample) variance of all values in a column,
- `STDEV`, the (sample) standard deviation of a random sample of values in a column,
- `VAR`, the (sample) variance of a random sample of values in a column

[5]The constant 1.4826 is used because for a normal distribution, one standard deviation from the mean is about 1.4826 MADs.

Obviously these aggregation functions allow outlier detection via traditional metrics such as the mean and standard deviation. Unfortunately, standard SQL offers no robust statistics as aggregate functions built into the standard language. In this section we discuss various approaches for implementing robust estimators in a relational database.

2.7.1 Median in SQL

We begin by discussing "vanilla" SQL expressions to compute the median. Later we will consider using somewhat less standard extensions of SQL to improve performance.

The basic idea of the median can be expressed declaratively in SQL by its definition: that value for which the number of values that are lower is the same as the number of values that are higher. For a column `c` of table `T`, a simple version is as follows:

```
-- A naive median query
SELECT c AS median
  FROM T
 WHERE (SELECT COUNT(*) from T AS T1 WHERE T1.c < T.c)
     = (SELECT COUNT(*) from T AS T2 WHERE T2.c > T.c)
```

Unfortunately that definition requires there to be an odd number of rows in the table, and it is likely to be slow: the straightforward execution strategy rescans `T` two times *for every row of* `T`, an $O(n^2)$ algorithm that is infeasible on a fair-sized dataset. A more thorough approach [Rozenshtein et al., 1997] gathers distinct values by using SQL's `GROUP BY` syntax, and (by joining `T` with itself) for each distinct value of `c` computes the number of rows with values less or greater, accounting for the case with an even number of rows by returning the lower of the two "middle" values:

```
-- a general-purpose median query
SELECT c as median
FROM T x, T y
GROUP BY x.c
HAVING
   SUM(CASE WHEN y.c <= x.c THEN 1 ELSE 0 END) >= (COUNT(*)+1)/2
AND
   SUM(CASE WHEN y.c >= x.c THEN 1 ELSE 0 END) >= (COUNT(*)/2)+1
```

This approach is also somewhat more efficient than the previous, since the naive execution strategy scans `T` only once per row of `T`. However, this is still an $O(n^2)$ algorithm, and remains infeasible on even modest-sized tables.

2.7.2 Sort-Based Schemes using SQL

The obvious algorithm for any order statistic is to sort the table by column `c`, count the rows, and identify the values at the appropriate position. Clearly this can be done by code running outside the database, issuing multiple queries. Psuedo-code for median might look like the following:

```
// find the median of a column
// first find the number of rows
cnt = exec_sql("SELECT count(*) FROM T");
```

```
if (cnt % 2 == 1)
  // odd number of rows: find the middle value
  results = exec_sql("SELECT c FROM T
                      ORDER BY c
                      LIMIT 1
                     OFFSET " + cnt/2);
  median = results.next();
else
  // even number: average the two middle values
  median = exec_sql("SELECT c FROM T
                    ORDER BY c
                    LIMIT 2
                    OFFSET " + cnt/2);
  median = (results.next() + results.next())/2;
```

In some database systems, including recent editions of Microsoft SQL Server, this can be achieved in a single SQL statement that results in roughly the same execution:

```
SELECT MEDIAN(c)
  FROM T
```

While this latter query looks simple, the performance will be about as bad as the previous pseudocode, due to the need to perform a sort of the table to compute the median. Any sorting-based approach takes $O(n \log n)$ operations. In practice, for a table larger than main memory, sorting requires at least 2 disk passes of the table; for tables larger than the square of the size of available memory, it requires yet more passes [Ramakrishnan and Gehrke, 2002, pp. 100-101].

2.7.3 One-Pass Approximation and User-Defined Aggregates

For very large tables and/or scenarios where efficiency is critical, there are algorithms in the database research literature that can, in a single pass of a massive data set, compute *approximate* values for the median or any other quantile, using limited memory. The approximation is in terms of the rank order: rather than returning the desired value (median, quantile, etc.), it will return some value from the data set that is within ϵN ranks of the desired value. The two most-cited algorithms [Manku et al., 1998, Greenwald and Khanna, 2001] differ slightly in their implementation and guarantees, but share the same general approach. Both work by scanning the column of data and storing copies of the values in memory along with a weight per value; during the scan, certain rules are used to discard some of the values in memory and update the weights of others. At the end of the scan, the surviving values and weights are used to produce an estimate of the median (or quantiles).

Again, these algorithms can be implemented by code running outside the database engine to manage the discard and weighting process. Unfortunately that approach will transfer every value from the database table over to the program running the algorithm which can be very inefficient, particularly if the median-finding program is running across a network from the server. To avoid this, and provide better software modularity, some modern databases allow *user-defined aggregate* (UDA) functions to be registered with the database. Once a UDA is registered, it can be invoked conveniently from SQL and executed within the server during query processing. To register a UDA, one must write three "stored procedures" or "user-defined functions": one to initialize any data structures needed during aggregation, one to process the next tuple in the scan and update the data structures, and one to take the final state of the data structures to produce a single answer. This approach, pioneered in the open-source Postgres database system [Stonebraker, 1986], is a natural fit to the approximation algorithms mentioned above, and is a sensible choice for a scalable outlier detection scheme in a modern database system. Having registered a UDA called `my_approx_median`, it can be invoked in a query naturally:

```
SELECT my_approx_median(c)
  FROM T
```

Another important practical advantage of the one-pass approximate schemes is queries like the following, that compute medians over multiple columns:

```
SELECT my_approx_median(c1),
       my_approx_median(c2),
       ...
       my_approx_median(cn)
FROM T
```

Using the one-pass approximate approach, all the medians can be computed simultaneously in a single pass of the table. By contrast, the exact median algorithms (including the built-in `median` aggregate) require sorting the table repeatedly, once per column. Hence even for moderately large tables, the approximate schemes are very attractive.

2.7.4 From Medians to Other Robust Estimators

While the previous discussion focused on the median, all the same techniques apply quite naturally for finding any desired quantile – including the approximation techniques. Given that fact, it is fairly easy to see how to compute the other robust estimators above.

For example, consider computing trimmed means. In this example, assume we use a one-pass approximation approach based on a UDA called `my_approx_quantile` that takes two arguments: the desired percentile and the column being aggregated. Then the 5% trimmed mean and standard deviation can be computed in SQL as:

```
-- 5% trimming
SELECT AVG(c), STDDEVP(c)
  FROM T,
       (SELECT my_approx_quantile(5,c) AS q5,
               my_approx_quantile,(95,c) AS q95
          FROM T AS T2) AS Quants
 WHERE T.c > Quants.q5
   AND T.c < Quants.q95
```

The 5% winsorized mean and standard deviation can be computed by a similar query:

```
-- 5% winsorization
SELECT AVG(CASE WHEN T.c < Quants.q5 THEN Quants.q5
                WHEN T.c > Quants.q95 THEN Quants.q95
           ELSE T.c),
       STDDEVP(CASE WHEN T.c < Quants.q5 THEN Quants.q5
                    WHEN T.c > Quants.q95 THEN Quants.q95
               ELSE T.c)
  FROM T,
       (SELECT my_approx_quantile(5,c) AS q5,
               my_approx_quantile,(95,c) AS q95
          FROM T AS T2) AS Quants
```

Finally, given that we have functions for the median, we would like to calculate the MAD as well. For an exact median, this is straightforward:

```
   SELECT median(abs(T.c - T2.median))
     FROM T,
          (SELECT median(c) AS median
             FROM T) AS T2
```

It is possible to replace the median aggregate in this query with an approximation algorithm expressed as a UDF; the error bounds for the resulting MAD appear to be an open research question. However, note that even with a one-pass approximation of the median, this query requires two passes: one to compute the median, and a second to compute absolute deviations from the median. An interesting open question is whether there is a direct one-pass, limited-memory approximation algorithm for the MAD.

2.8 Non-Normal Distributions

Much of our discussion was built on intuitions based in normal distributions. Of course in practice, not all data sets are normally distributed. In many cases, the outlier detection schemes we consider will work reasonably well even when the distribution is not normally distributed. However, it is useful to be aware of two commonly occurring cases of distributions that are not normal:

- **Multimodal Distributions:** In some cases, a data set appears to have many "peaks"; such distributions are typically referred as being *multimodal.* In some cases these distributions can be described via superimposed "bell curves", known as *mixtures of Gaussians.*
- **Zipfian Distributions:** In many settings with data that varies in popularity, a large fraction of the data is condensed into a small fraction of values, with the remainder of the data spread across a "long tail" of rare values. The Zipfian distribution has this quality; we discuss it in more detail in Section 6.2.

In applying data cleaning methods, it can be useful to understand whether one's data is more or less based in a normal distribution or not. For univariate data, the standard way to do this is to plot a histogram of the data, and overlay it with a normal curve. A Q-Q plot [Barnett, 1975] can also be used to "eyeball" a data set for normality, as we illustrate in Section 7.1. In addition, there are a variety of formal statistical tests for normality [Chakravarti and Roy, 1967, C.E. and W., 1949, Frieden, 2004, Shapiro and Wilk, 1965, D'Agostino and Pearson, 1974]. However, at least one source suggests that these tests are "less useful than you'd guess" [Software, 2008], and cites D'Agostino and Stephens who are also equivocal, saying

> Attempting to make final recommendations [for normality tests] is an unwelcome and near impossible task involving the imposition of personal judgements. . . . A detailed

graphical analysis involving normal probability plotting should always accompany a formal test of normality [D'Agostino and Pearson, 1974, pp. 405-406].

Note also that outliers can have significant effects on some normality tests; Hartigan and Hartigan's *dip* statistic is a commonly-cited robust normality test [Hartigan, 1985].

Suppose one decides via some combination of looking at plots (D'Agostino and Stephen's "detailed graphical analysis") and examining statistical tests that a data set is in fact not unimodal. What is to be done at that point to remove outliers? Four approaches are natural:

1. *Use outlier tests based on the normality assumption.* One option that is easy to adopt is simply to ignore the non-normality in the data, and use the outlier detection tests we describe in the subsequent sections that are based in normal assumptions. For multimodal distributions, these tests will continue to identify outliers that are at the extremes of the distrbution. They will not identify outliers that fall "between the bells" of multiple normal curves. They also may incorrectly label data in the edges of the outermost "bells" as being outliers, when they would not be so labeled if each bell were examined alone. However, that kind of concern is always present in outlier detection depending on the setting of threshholds: how much of the data at the "edges" is truly outlying? The answer to this question is always domain-specific and judged by an analyst. With respect to Zipfian distributions, normality assumptions will tend to arbitrarily pick points in the "long tail" as outlying, even though in many cases those points are no more likely to be outliers than points in the tail that are closer to the mean. An analyst's interpretation of these outlying points may be clouded as well, since by definition the "strange" or "unpopular" values are many, and likely to surprise even people familiar with the domain. Knowing that the tail is indeed long can help the analyst exercise caution in that setting.

2. *Model the data and look for outliers in the residuals.* Another approach is to choose a model other than a normal distribution to model the data. For example, if one believes that data is Zipfian, one can try to fit a Zipfian distribution to the data. Given such a model, the distribution of the data can be compared to the model; the differences between the empirical data and the model are called *residuals*. Residuals of well-chosen models are quite often normally distributed, so standard outlier detection techniques can be applied to the set of residuals, rather than the data – points with outlying residuals represent potential outliers from a model-based point of view. The various techniques in this paper can be applied to residuals quite naturally.

3. *Data partitioning* schemes can either manually or automatically partition a data set into subsets, in hopes that the subsets are more statistically homogeneous and more likely to be normal. Subsequently, outlier detection techniques can be run within each subset, and the subsets themselves examined to see if an entire partition is outlying. This approach is particularly natural for multimodal distributions. Unfortunately Zipfian distributions are "self-similar" – their subsets are also Zipfian. Data partitioning can be done using a variety of manual, automatic and semi-automatic schemes. The standard manual scheme for partitioning database data is to use *OLAP* or so-called *data cube* tools to help an analyst manually partition data into sensible "bins" by "drilling down" along multiple attributes. Sarawagi introduced semi-automatic techniques for identifying "interesting" regions in the data cube, which may help the analyst decide how to partition the data [Sarawagi, 2001]. Johnson and Dasu introduce an interesting multidimensional clustering technique called *data spheres* that build on the notion of quantiles in multiple dimensions [Johnson

and Dasu, 1998]. The standard fully-automated scheme for partitioning data statistically is called *clustering*, and there are a wide range of techniques in the literature on efficient and scalable clustering schemes [Berkhin, 2002].

4. *Non-parametric outlier detection.* Another option available for data that deviates significantly from normality is to use non-parametric "model-free" approaches to outlier detection. We survey a number of these techniques in Section 3.3. These techniques have drawbacks and fragilities of their own, which we also discuss.

Clearly, handling outlier detection with non-normal data presents a number of subtleties. The most important conclusion from this discussion is that in *all* settings, outlier detection should be a human-driven process, with an analyst using their judgment and domain knowledge to validate information provided by algorithms. In particular, if an analyst is convinced (e.g., via graphical tests) that a data set is not normal, they need to be particularly careful to choose outlier approaches that accomodate their non-normality. Many experienced data analysts agree that automated techniques should be accompanied by data visualizations before conclusions are drawn. In fact, the choice and parameterization of data cleaning methods is often aided by visualization-driven insights, and it is useful to consider the use of visualizations, analyses, and data cleaning transformations to be integrated into an iterative process [Raman and Hellerstein, 2001].

3 Multivariate Outliers

In the previous section, we considered methods to detect outliers in a set of numbers, as one might find in a column of a database table. In statistical nomenclature, this is the *univariate* (or *scalar*) setting. Univariate techniques can work quite well in many cases. However, there is often a good deal more information available when one considers multiple columns at a time – the *multivariate* case.

As an example, consider a table of economic indicators for various countries, which has a column for average household income, and another column for average household expenditures. In general, incomes across countries may range very widely, as would expenditures. However, one would expect that income and expenditures are positively correlated: the higher the income in a country, the higher the expenditures. So a row for a country with a low per-capita average income but a high per-capita average expenditure is very likely an error, *even though* the numbers taken individually may be well within the normal range. Multivariate outlier detection can flag these kinds of outliers as suspicious.

Multivariate techniques are analogous in some ways to ideas we saw in the univariate case, but the combinations of variables make things both more complicated to define and interpret, and more time-consuming to compute. In this section we extend our basic measures of center and dispersion to the multivariate setting, and discuss some corresponding robust metrics and outlier detection techniques.

3.1 Intuition: Multivariate Normal Distributions

In the univariate case, we based our intuition on the mean and standard deviation of a set of data, leading to the normal or "bell curve" shown in Figure 2.

Extending this in a simple way to two dimensions, one might define some (x, y) pair as the mean, and a single standard deviation number to define the dispersion around the mean. This

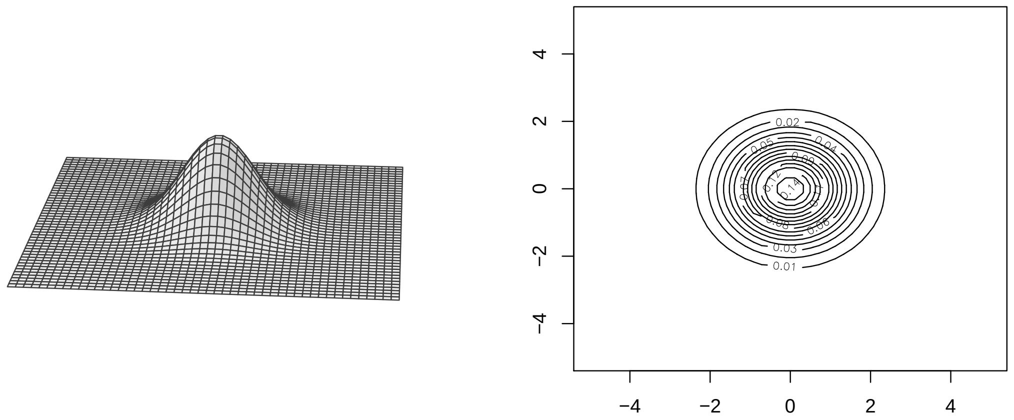


Figure 4: A bivariate normal distribution with no correlation between the variables. The left side shows a three-dimensional plot of the probability density function; the right side shows a contour plot (looking "down" at the bump) of equi-probable (x, y) pairs.

leads to the three-dimensional plot on the left of Figure 4, where the "horizontal" x and y axes represent possible values of the two fields in the pair, and the "vertical" z axis represents the probability of those two variables taking on that value. Note that all points lying on a circle about the center have the same probability, as shown by the equiprobability contour plot on the right of Figure 4.

The problem with this extension is the use of a single number for the dispersion around the center. That single number can only define distributions that are symmetric about the center as in Figure 4, which is not enough to capture x/y correlations like the one we saw between income and expenditures. Those correlations should generate equi-probable *ellipses* around the center as in Figure 5, rather than circles. To capture these correlations more generally, a multivariate normal distribution is defined by a multivariate mean and a two-dimensional *covariance* matrix Σ, which can be thought of as a generalization of the univariate variance to multiple variables. Recall that the *variance* of a single variable is the standard deviation squared. The covariance matrix essentially captures the variance of each variable individually, *and* the correlation (covariance) between pairs of variables[6].

For our purposes in data cleaning, this discussion leads us to a baseline (though non-robust) definition of "center" and "dispersion" on which we can build. The center is the multivariate mean (i.e., the mean along each dimension), and the dispersion is defined by the covariance matrix.

The final question to resolve is how we use the covariance matrix to measure how far any particular point is from the mean. In the univariate case, we care how far the point is from the mean in either direction, relative to the dispersion, as measured by standard deviation. In the multivariate case, we care how far the point is from the mean in all directions, relative to the dispersion as measured by the covariance matrix. The standard metric is known as the

[6]Formally, given n variables $X_1, \ldots, X_n$, an element Σ_{ij} in the multivariate covariance matrix Σ is defined to be $(X_i - \mu_i)(X_j - \mu_j)$. Note that the diagonal of the covariance matrix has $i = j$, and corresponds to a univariate variances for each variable.

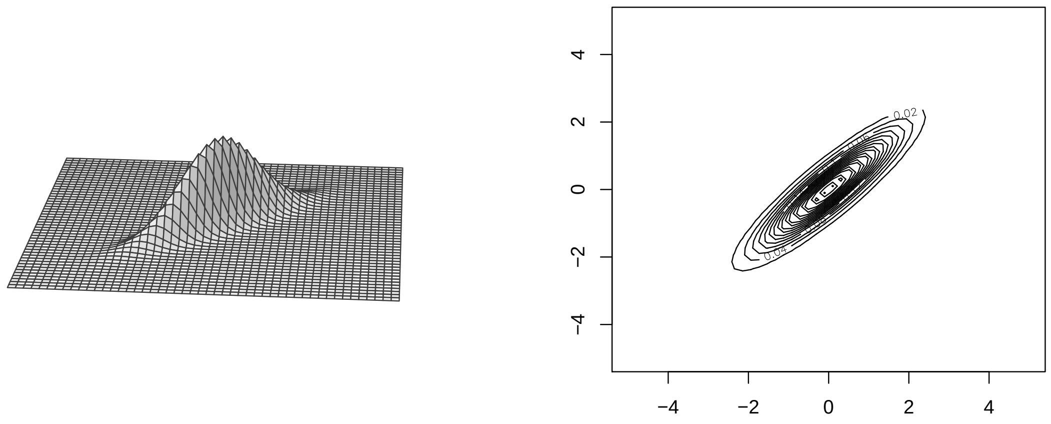


Figure 5: The probability density function of a bivariate normal distribution with significant correlation, viewed in three dimensions and as equi-probable contours.

Mahalanobis depth of the point, which is defined as

$$\sqrt{(x-\mu)'(\Sigma^{-1}(x-\mu))}$$

where Σ^{-1} is the inverse of the covariance matrix. The threshold for Mahalanobis depth depends on the number of variables d being considered, and can be compared against quantiles of the chi-squared (χ^2) distribution with d degrees of freedom [Rousseeuw and Leroy, 1987, p. 224]. As in univariate data, it is common to look for outliers outside the 5th and 95th percentiles.

3.1.1 Two-dimensional Metrics in SQL

Vanilla SQL does not make it easy to manipulate matrices. In this section we derive simple queries for computing the mean and Mahalanobis distances for a two-dimensional data, and discuss extensions to deal with more dimensions.

Computing the k-dimensional mean in SQL is no harder than computing the one-dimensional mean. For our two-dimensional example, we will give the result a name, `Two_d_mean`, for reuse later.

```
CREATE VIEW Two_d_mean AS
SELECT AVG(x) AS mu_x, AVG(y) AS mu_y
  FROM T;
```

Note that `Two_d_mean` contains a single row due to the `AVG` function.

The next step is to calculate the Covariance matrix. We will represent it as a table of one row, with the cells of the 2×2 matrix stored in table attributes, ordered in row-major form:

```
CREATE VIEW Covariance AS
SELECT SUM((T.x - M.x)*(T.x - M.x)) AS xx,
       SUM((T.x - M.x)*(T.y - M.y)) AS xy,
       SUM((T.y - M.y)*(T.x - M.x)) AS yx,
       SUM((T.y - M.y)*(T.y - M.y)) AS yy
  FROM T, Two_d_mean M;
```

Conveniently, the inverse of a two-dimensional matrix is simply expressed via arithmetic:

```
CREATE VIEW InvCov AS
SELECT    (1/(xx*yy - xy*yx))*yy AS ul,
       -1*(1/(xx*yy - xy*yx))*xy AS ur,
       -1*(1/(xx*yy - xy*yx))*yx AS ll,
          (1/(xx*yy - xy*yx))*xx AS lr
  FROM Covariance;
```

Finally, given the inverted covariance matrix, we can compute Mahalanobis distances:

```
CREATE VIEW Mahalanobis AS
SELECT x, y, sqrt(((x-mu_x)*InvCov.ul + (y-mu_y)*InvCov.ll)*(x-mu_x) +
                  ((x-mu_x)*InvCov.ur + (y-mu_y)*InvCov.lr)*(y-mu_y))
  FROM T, InvCov, Two_d_mean;
```

This completes the implementation of the two-dimensional metrics for center and dispersion. Extending this to more dimensions becomes clumsy in SQL. First, the number of entries in the Covariance matrix equals the square of the number of dimensions, and the lengths of the queries above grow accordingly. Second, computing the inverse of a matrix with more than two dimensions is complex, and not easily represented in SQL: it requires a sophisticated algorithm like Gaussian elimination, LU decomposition, or the like [Strang, 2003].

Hence for more than two dimensions, when feasible it is best to export the data from the relational database to a linear algebra or statistics package to compute the Mahalanobis distances. The leading open-source statistics package is R [R Project].

3.2 Robust Multivariate Estimation

As with the univariate mean and standard deviation, the multivariate mean and covariance are not robust – they have a breakdown point of $1/n$, meaning that a single outlier can perturb them arbitrarily. It is natural to extend our univariate techniques above by defining robust multivariate centers and dispersions in some fashion, and computing distances from the center in those terms.

There are a number of techniques in the literature for "robustifying" multivariate estimators of center and dispersion, and this remains an ongoing area of research in statistics and data mining. Here we consider four classes of techniques; the first two can be implemented on large databases via techniques we have covered above.

- **Iterative deletion** of the max-Mahalanobis points. The main problem with using non-robust metrics for the center and dispersion is that of masking; a radically high-distance point may move the center and dispersion enough to make less extreme outliers look "reasonable". A simple scheme to combat this is to remove the maximum-distance point from the dataset, recompute the center and dispersion on the trimmed data, and repeat. Several rules are possible for deciding when to stop repeating; a natural one is when the center and dispersion change very little across two iterations. This technique is known to degrade with the number of variables – its breakdown point is at most $1/V$, where V is the number of variables [Rousseeuw and Leroy, 1987]. Implementing this in a database is equivalent to repeatedly computing the center and spread as described above.

- **Rescaling via robust componentwise estimators.** The idea here is to first compute a center of the data using a robust approach, and then compute the distance of points from that center in standardized units of robust dispersion. This is particularly convenient using *componentwise* robustification, where each variable is "robustified" separately using a univariate estimator. For example, to compute a center, a robust choice is the componentwise median – the point defined by the median of each variable. (Dasu and Johnson also suggest the componentwise mean, or trimmed componentwise mean, as other options [Dasu and Johnson, 2003].) Given a center metric, the data are rescaled so that each variable is measured in a standardized unit of dispersion, robustified in a manner consistent with the center computation. For example, for a componentwise median, the dispersion metric might be the MAD, whereas for the (trimmed) multivariate mean, the metric might be the (trimmed) standard deviation. Finally, the distance of each rescaled point from the center is computed using a standard distance function (e.g. standard Euclidean distances). At this stage, a threshhold on the distance of a point from the center can be used to detect outliers. Dasu and Johnson offer no analysis of the robustness of this scheme, but assert that in practice, examining data depth layers using this technique works well [Dasu and Johnson, 2003]. An attraction of componentwise methods is that the univariate techniques of the previous section can be applied on large databases.

- **Robust estimators for mean and covariance:** In a similar vein, but using more statistical machinery, there are a number of techniques that provide optimal-breakdown estimators for the center and dispersion. Given these, Mahalanobis distances from the robust center can be computed for all points using the robust dispersion. The most widely used method is called the Minimum Covariance Determinant (MCD), which computes the subset of h points from the data that minimizes the determinant of the covariance matrix [Rousseeuw and Van Driessen, 1999]. The mean and covariance of these points are used as the center and dispersion for the dataset. These measures can be shown to have an optimal 50% breakdown point for arbitrarily large data sets (particularly data sets with many more rows than columns). Other such robustification techniques include the Minimum Volume Ellipsoid (MVE) [Rousseeuw and Leroy, 1987] and the Orthogonalized Gnanadesikan-Kettering (OGK) [Maronna and Zamar, 2002] approaches. The implementation of these techniques for very large data bases is an open research problem. All of these are implemented in the open source R statistical package, as part of the `robustbase` and `rrcov` packages [Todorov, 2006].

- **Depth-based techniques.** In the univariate setting, the ordering of data points along the single variable determines the "ranks" of the points, which leads to definitions of *order statistics* like the median (the mid-ranked point) and quantiles. Depth-based techniques attempt to develop an analogy to order statistics in multiple dimensions, where the ordering of the points is not so clearly defined. To borrow a 3-dimensional analogy from Dasu and Johnson [Dasu and Johnson, 2003], consider a stack of oranges at a supermarket. One can define the "depth" of an orange in the stack as the number of other oranges one must remove before reaching the orange in question. The orange of maximum depth is considered to be the center of the stack, akin to a univariate median. Outliers may simply be defined as the outermost (depth 0) layer of oranges. Alternatively, the depth may be used to develop a multivariate version of trimming. First, the a proportion α of the lowest-depth points are trimmed from the data set. The remaining points are used to compute a trimmed mean and trimmed covariance matrix, leading to a trimmed

Mahalanobis distance metric that can be used to identify outliers.

We have given only an intuitive definition of depth. There are many formalizations in the literature that vary in their statistical properties and their computational difficulty. Dasu and Johnson describe a number of them in a relatively intuitive fashion [Dasu and Johnson, 2003]; Liu, Parelius and Singh provide more detail [Liu et al., 1999]. Depth-trimmed estimators can have breakdown points as low as $1/(d+1)$ for d variables. However, implementations for very large databases are an open research question.

3.3 Distance-Based Outliers: A Different Approach

Our discussion up to now has focused on outliers as defined by various statistical estimators. Underlying the discussion was the idea that an outlier is a data point that has low probability with respect to a model for the majority of the other points.

However, some datasets simply do not fit well into a bell-curve model, and the analyst may be unwilling or unable to suggest an alternative statistical model that fits well, and can lead to a residual-based analysis of outliers (as suggested in Section 2.8). In recent years, the Data Mining literature has developed a number of non-parametric, model-free notions of outliers that do not depend upon finding an appropriately fitting statistical distribution. The intuition with these techniques is simply to measure the pairwise distances between points, and define outliers as those points whose distance from their "neighboring" points is high.

As one example, Kollios, et al. define a data point p to be a $DB(k, D)$-outlier if at most k other points lie within a distance D [Kollios et al., 2003]; Bay and Schwabacher present an algorithm with expected running time of $O(N)$ for this outlier detection metric [Bay and Schwabacher, 2003]. A very similar metric of Knorr and Ng is based on the percentage of objects at a large distance from a point [Knorr and Ng, 1999]. Ramaswamy, et al. define an outlier as one of the top n data elements whose distance to its kth nearest neighbor is above a fixed threshold [Ramaswamy et al., 2000]. All of these metrics suffer in cases where the density of points in different clusters of data can differ widely. Breunig, et al. provide a density-based definition of an outlier center, which is defined as the reciprocal of the "density of the object's neighborhood"; this density can be defined as the average distance of the node to its k nearest neighbors [Breunig et al., 2000].

These metrics are non-parametric: they make no assumptions about probability distributions, and hence can often handle cases like those of multiple clusters described above better. On the other hand, this disconnection from probability distributions brings some handicaps to these schemes. They do not provide probabilistic measures of the "surprisingness" of individual points, and they have the unpleasant property that when given "clean" data, they will typically still identify "outliers". (This latter property was also true of the depth-based approach above that defined outliers as depth-0 items.) They also suffer from the fact that they are not scale-invariant: changing the units on any dimension can change the definition of outliers entirely. This is not true of many of the more sophisticated techniques (MCD, MVE) mentioned above. And the lack of any underlying distribution assumption means that the kind of normalized rescaling discussed in Section 3.2 is also not an option.

4 Timeseries Outliers

Timeseries data differs from traditional numeric data because of the correlations between readings across timesteps: one expects that readings will remain "close" from one timestep to the next, and the effect of a reading on later readings dampens as time passes. In addition, timeseries data sometimes contains "seasonalities": for example, toy sales tend to grow every fall until Christmas, year after year. These seasonalities can be seen as more coarse-scale correlations in time, e.g., from year to year rather than from month to month.

4.1 Intuition

Timeseries outliers are classically described as being in one of two categories:

1. *Additive outliers*, sometimes called "hiccups", are outlying values that appear in isolation: after the additive outlier, the continuation of the timeseries returns to normal behavior immediately. Additive outliers are typical of data entry errors.

2. *Innovation outliers* are incorrect values that, due to timeseries correlations, contaminate the values that follow them, until their effects dampen out some number of timesteps in the future. Innovation outliers often occur in sensors that themselves exhibit correlations over time; a typical example is a mercury thermometer measuring "room temperature" that is exposed to a localized hot flame – the mercury will cool slowly, even though the overall room temperature remains constant throughout.

Subsequent literature [Tsay, 1988] defines more complex outlier phenomena that affect the "structural" behavior of the timeseries over the long term, including *Level Shifts* and *Variance Changes*. Advanced readers may wish to consult the research papers for discussion of these topics.

One typical scheme for dealing with timeseries outliers is to "smooth" the data using *windowing*. For example, given a daily timeseries $x_1, \ldots, x_n$, the 7-day windowed average (sometimes called a *moving* or *rolling* average) at time t is defined as the unweighted mean of the last 7 days: $y_t = \frac{1}{7}(x_t + x_{t-1} + \ldots + x_{t-6})$. The resulting timeseries can be used as a point of comparison to the original data – a point x_t that is "far away" from the corresponding moving average y_t can be considered an outlier. The definition of "far away" can be based on standard deviations. Classically, the variance of a timeseries is considered to be constant, and hence can be computed over any window of non-trivial size. It is also easy to compute a windowed standard deviation, which will track changes in variance in a simple way.

The standard generalization of windowing is to place more flexible weights on the influence of past data, where the weight typically goes down smoothly and quickly with the distance in time. The typical scheme here is the *exponentially weighted moving average (EWMA)*, which assigns weights that decrease exponentially in time. It is defined (recursively) as $y_t = \lambda x_t + (1-\lambda)y_{t-1}$ for some $0 \leq \lambda \leq 1$. The EWMA is used in many settings in timeseries analysis. Exponentially Weight Moving Variance (EWMV) schemes have also been used to measure changes in variance; a typical measure is to define the moving variance as $z_t = (1-\lambda)z_{t-1} + \lambda(x_t - \mu_o)^2$, where μ_0 can be measured in a number of ways: as the mean from time 0 until the current timestep, as a EWMA, or a windowed mean [MacGregor and Harris, 1993].

As a heuristic, it can be useful to look at plots of the windowed or exponentially weighted moving variance as an outlier indicator: outliers in the base timeseries will result in temporary increase in the moving variance, which might otherwise be fairly constant.

Note that seasonal timeseries outliers are not well studied in the literature; Kaiser and Maravall present some recent results [Kaiser and Maravall, 1999].

4.2 Robust variants

As in the previous sections, the concern with using means and variances is that they can break down with even a single outlier. The statistical literature on robust timeseries methods is still a work in progress: it only began in earnest in the 1990's, and while there is still no textbook treatment of the material that is easily accessible to someone who is not an academic statistician [Maddala and Rao, 1997], there are natural extensions to timeseries that are intuitively attractive and easy to interpret.

A simple technique is to window the timeseries, and use the windowed median and MAD to detect outliers via Hampel X84. As in the prior discussion, the MAD can be computed over the whole timeseries up to that point, or over a suitably large window.

There is no discussion in the literature about applying exponential (or other) timeseries weighting schemes to robust measures like the median; it is unclear what weighting would be appropriate. However, ARIMA [Box et al., 1994] is a popular model for representing timeseries, which builds on moving averages (the "MA" in the ARIMA acronym) for representing timeseries processes and disturbances. Many traditional timeseries tasks including forecasting involve fitting an ARIMA model to data. While this report does not focus on ARIMA (or the simpler ARMA) models, we mention briefly that there has been work in recent years on integrating outlier detection into these models, including [Box et al., 1994, Findley et al., 1998]. Notably, the US Census X-12-ARIMA package contains a built-in outlier detection technique based on work by Otto and Bell [Otto and Bell, 1990]. Bianco, et al. extend this work with robust estimates and show better detection results with many outliers [Bianco et al., 2001].

5 Resampling Techniques

In most of this survey we have grounded the detection of outliers in the use of robust estimators. In this section we briefly discuss alternative approaches that use *resampling* methods to get reliable outlier detection with more traditional approaches.

The basic idea in resampling is to repeatedly take samples of a data set in a controlled fashion, compute a summary statistic over each sample, and carefully combine the samples to estimate of a property of the entire data set. Intuitively, rare outliers will appear in few or no samples, and hence will not perturb the estimators used. Given good estimators computed in this fashion, outliers can be found by their divergence from the estimated properties of the data. There are two standard resampling techniques in the literature:

- The basic *bootstrap* method is straightforward: for a dataset of size n, one draws n samples from the data set *with replacement*, and computes the summary statistic of interest to generate an observation. This sample-then-estimate process is repeated thousands of times. The resulting set of observations is called a *sampling distribution*. There is no clear rule for the number of iterations of the bootstrap that is required; typical numbers are in the thousands or millions for smallish data sets.

- The *jackknife* method works by repeatedly recomputing the summary statistic leaving out one data item at a time from the data set. This is repeated a modest number of times, perhaps a few dozen. The resulting set of observations is used as a sampling distribution.

Given a sampling distribution from one of these resampling methods, one can estimate both the mean and variance of the desired summary statistic. Different summary statistics (means, medians, etc.) have different known sampling distributions, and hence different rules for computing the mean and variance.

Resampling can be used for more complex summary statistics than simple means and variances of the original data. For example, in bivariate (two-dimensional) data, resampling can be used in conjunction with linear regression to identify outliers from the regression model [Martin and Roberts, 2006].

6 Cleaning Up Counts: Frequency Outliers

In some scenarios, the specific values in a column are less important than the number of times each value is repeated in the column – the *frequency* of the values. Frequency statistics can be important not only for quantitative attributes, but for *categorical* attributes that do not have an inherent numerical interpretation. For categorical attributes, the "names" of the data values are relevant, but the ordering of values is not. For example, in a database of animal sightings, a "species code" column may be numeric, but it is usually irrelevant whether the code number assigned to iguanas is closer to the code for tigers than it is to the code for mice. For categorical data, the statistical distribution of values is important in terms of the frequencies of occurrence of each value – the number of mice, tigers, and iguanas observed. As a result, the techniques for outlier detection in previous sections are not a natural fit for cleaning dirty data in a key column.

In this section we discuss outlier methods that focus on two extremes of frequencies: "distinct" attributes where nearly every value has frequency 1, and attributes that have high frequency "spikes" at some value.

6.1 Distinct Values and Keys

It is quite common for a data set to have *key* columns that provide a unique identifier for each row. A key can be made up of a single column (e.g., a taxpayer ID number), or of a concatenation of columns (e.g., <cityname,countrycode>). We return to multi-column "composite" keys in Section 6.1.1

In a key with perfect integrity, the frequency of each value is equal to 1, and hence the number of distinct values in the key columns should be the same as the number of rows. However, it is often the case in dirty data that a would-be key contains some duplicated values, due either to data entry errors, or repeated use of some value as a code for "unknown".

There are two scenarios when identifying dirty key columns is important. The simple scenario is one of data repair, where one has knowledge about the column(s) that form a key, and is trying to determine which entries in the key column(s) need to be cleaned. Identifying the potentially dirty rows amounts to identifying duplicated key values, and returning all rows with those key values.

The somewhat more complex scenario is "key discovery", where one is trying to discover which columns might actually be intended as keys, despite the fact that there may be some dirty data. This problem requires coming up with a metric for how "close" a column is to being a key. One intuitive measure for dirty keys is what we call the *unique row ratio*: the ratio of distinct values in the column to the total number of rows in the table. If this is close to 1.0, the column may be flagged as a potential dirty key. This is the approach used proposed by Dasu and Johnson [Dasu et al., 2002] However, this test is not robust to "frequency outliers": scenarios where there are a small number of values with very high frequency. This problem often occurs due to the spurious integrity problem mentioned in Section 1.1, which lead data-entry users to use common "dummy" values like 00000 or 12345 as a default. A more robust measure is what we call the *unique value ratio*: the ratio of "unique" values (values with frequency one) to the total number of distinct values in the key column(s).

SQL's aggregation functionality provides language features to compute basic frequency statistics fairly easily. For example, here is a query to compute the unique value ratio mentioned above:

```
SELECT UniqueCount.cnt / DistinctValues.cnt
  FROM
        -- Unique values with frequency 1
        (SELECT COUNT(Uniques.c) AS cnt
         FROM (SELECT c FROM T
                GROUP BY c HAVING COUNT(c) = 1) AS Uniques) AS UniqueCount,
        -- Number of distinct values
        (SELECT COUNT(DISTINCT c) AS cnt FROM T) AS DistinctValues;
```

Unfortunately, this query – or any other deterministic scheme for computing frequencies – can be very resource-intensive and time-consuming. The main problem arises in large tables with columns with contain many distinct values. Counting the number of distinct values requires some scheme to bring all copies of each value together – this is expressed by the `GROUP BY` and DISTINCT clauses in the SQL query above. This is typically implemented (either in a database engine or application code) by either sorting the table, or using a hashing scheme with one hash "bucket" per distinct value. In either case this requires multiple passes over the data for large tables with many distinct values.

However, there are a number of elegant one-pass approximation schemes for computing frequencies that can be implemented naturally as user-defined aggregates in a relational database. The Flajolet-Martin (FM) Sketch [Flajolet and Martin, 1985] is an important tool here, providing a one-pass approximation to the number of distinct values. The FM sketch is a bitmap that is initialized to zeros at the beginning of a scan. Each bit in the bitmap is associated with a random binary hash function chosen carefully from a family of such functions. As each value is scanned during a pass of the table, the hash function for each bit is evaluated on that value, and if it evaluates to 1 then the corresponding bit in the sketch is turned on. At the end of the scan, the resulting bitmap can be used to estimate the number of distinct values within a factor of ϵ, where ϵ is a function of the number of bits in the sketch.

FM sketches can be used directly to compute an approximate unique row ratio in a single pass. FM sketches can also be used to compute the denominator of the unique value ratio. However, the numerator is a specific form of what is called an *Iceberg* or *Heavy Hitter* Query: a query that looks for values with frequency above some threshold [Fang et al., 1998, Hershberger et al., 2005]. Unfortunately, the desired threshold for the unique value ratio is extremely low (1 row, or equivalently a $\frac{1}{N}$ fraction of the N rows in the table). The various approximation algorithms for iceberg queries [Fang et al., 1998, Manku and Motwani, 2002, Hershberger et al., 2005] do not perform well for such low thresholds.

The literature does not appear to contain a one-pass solution to key detection that is robust to frequency outliers. But there is a natural two-pass approach akin to a trimmed unique row ratio. First, an approximate iceberg technique like that of [Manku and Motwani, 2002] is used to identify the set of values S that have frequencies above some sizable threshold – say 1% of the rows (assuming a table much larger than 100 rows). Then, a second pass uses FM sketches to compute the unique value ratio of those rows that are not in the set of "iceberg" values.

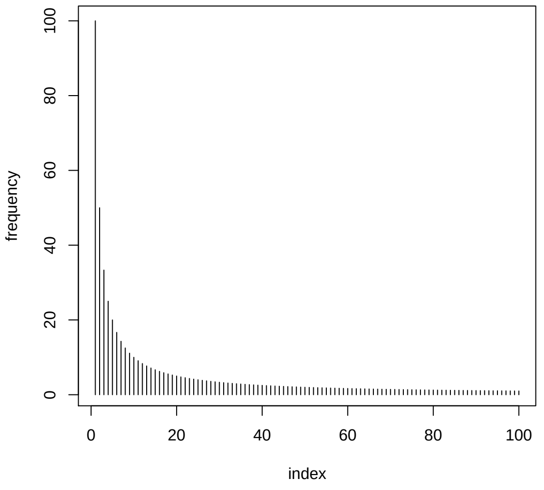


Figure 6: A Zipf Distribution, $1/x$.

To our knowledge, the question of analyzing the breakdown points of the unique value ratio or the trimmed unique row ratio are open research questions.

6.1.1 Composite Keys

We have identified a number of metrics for evaluating the likelihood of a column being a key. Any of these metrics can be applied to evaluate the likelihood of a set of columns forming a composite key as well. The problem that arises is that there are exponentially many possible sets of columns to evaluate, and even with one-pass approximations this is infeasibly slow.

Tane is an efficient algorithm for discovering approximate composite keys; it can be used with any of the metrics we describe above. The core of Tane is an efficient algorithm for deciding which combination of columns to text for "key-ness" next, based on the combinations previously tested.

It is not uncommon to uncommon to couple an algorithm like Tane with some heuristics about which combinations of columns could be keys. For example, the Bellman data cleaning system only considers composite keys with 3 or fewer columns. A recent related algorithm called Gordian [Sismanis et al., 2006] was proposed for discovering exact keys (with no duplicates in them); it seems plausible to extend Gordian to handle dirty data, but this is an open question.

6.2 Frequent Values and Zipfian Distributions

In many data sets, it is useful to identify values that have unusually high frequencies [Cormode and Muthukrishnan, 2003, Manku and Motwani, 2002, Hershberger et al., 2005]. It is often useful to report the heavy hitters in a data set – both because they are in some sense "representative" of an important fraction of the data, and because if they are indeed contaminated, they can have a significant effect on the quality of the data overall.

As noted above, heavy hitters frequently arise due to spurious integrity effects in manual data entry settings. They also arise in automatic measurement settings, when devices fall back to default readings in the face of errors or uncertainty. In these contexts, the heavy hitters often include erroneous values that should be cleaned.

Heavy hitters also become noticeable in data that follows Zipfian [Zipf, 1949] or other power-law distributions [Mitzenmacher, 2004] (Figure 6.2). In popular terminology these are sometimes referred to as “80-20” distributions, and they form the basis of “the long tail” phenomenon recently popularized in a well-known book [Anderson, 2006]. Examples of data that follow these distributions include the frequency of English words used in speech (Zipf’s original example [Zipf, 1949]), the popularity of web pages, and many other settings that involve sharp distinctions between what is “hot” and what is not, in terms of frequency. An interesting property of Zipfian distributions is that they are *self-similar*: removing the heavy hitters results in a Zipfian distribution on the remaining values. As a result, in these distributions it is difficult using the techniques discussed earlier to identify when/if the data is “clean”: frequency outliers are endemic to the definition of a Zipfian distribution, and removing them results in a new Zipfian which again has a few “hot” items.

Instead, in scenarios where the data is known to be Zipfian, a natural statistical approach is to fit a Zipfian model to the data, and study the *residuals*: the differences between observed frequencies, and what the best-fitting model predicts. Residuals often fall into a normal distribution, so robust mechanisms can be used to identify outliers among the residuals, which indicate outlying frequencies (outlying with respect to the Zipfian assumption, anyway). The approach of using models and identifying outliers among the residuals is quite common, though the standard models employed in the literature are usually based on linear regression. The interested reader is referred to [Rousseeuw and Leroy, 1987] to learn more about these model-based approaches.

In recent years there have been a number of approximate, one-pass heavy hitter algorithms that use limited memory; Muthukrishnan covers them in his recent survey article on data streams [Muthukrishnan, 2005].

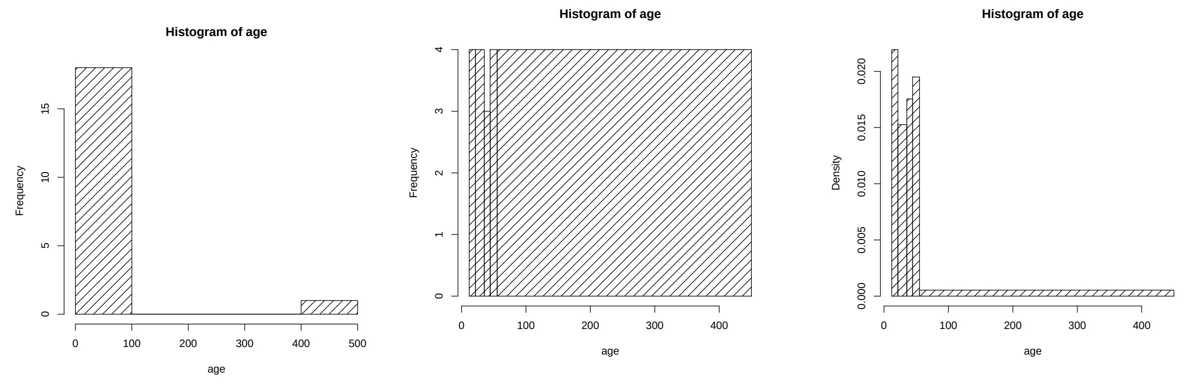
7 Notes on Interface Design

Data cleaning can be influenced greatly by the thoughtful design of computer interfaces. This is true at all stages in the “lifetime” of data, from data collection and entry, through transformation and analysis. The field of human-computer interaction with quantitative information is rich and growing; it is the subject of numerous books and research papers. Despite this fact, there is very little work on interface design for data entry and data cleaning. This section touches briefly on both data visualization and data entry. In addition to surveying a few of the leading data visualization techniques used in data cleaning, we provide a less scholarly discussion of some the key design principles in data entry that seem to be both important and largely overlooked.

7.1 Cleaning Data Via Exploratory Data Analysis

The human visual system is a sophisticated data analysis engine, and data visualization has been a rich area of research in both statistics and computer science. A few classical techniques stand out for their utility in data cleaning; we them briefly survey here.

Histograms are a natural way to visualize the density of data points across values of a single dimension. The basic idea of a histogram is to partition the data set into bins, and plot the count or probability of items in each bin. *Equi-width* histograms partition the domain of an attribute (the set of possible values) into bins, and the heights of the bars illustrate the frequencies of the



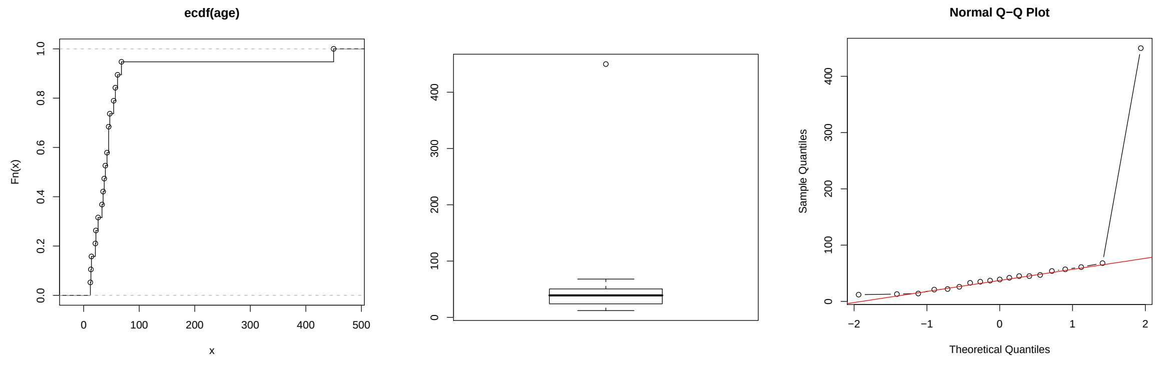
An equiwidth histogram. An equidepth histogram showing counts. An equidepth histogram showing probabilities.

Figure 7: Three five-bin histograms for the dirty age data of Section 2.3. In the rightmost, the vertical axis is normalized to probability density units, so that the area under the histogram sums to 100%.

bins. For clean data, these histograms can look very similar to probability density functions, and hence provide a good intuitive picture of the statistical properties of a data distribution. Extreme outliers can often be seen easily in equi-width histograms (bins at the far right and left, as in Figure 7, left), as can extreme frequency outliers (very tall bins). However, as seen in the leftmost chart of Figure 7, extreme outliers can also swamp more detailed display of other data. *Equi-depth* histograms construct bins of near-equal count (depth), but the bins differ in the width of their endpoints. The bin boundaries correspond to quantiles in the data, and in essence equi-depth histograms are a visual representation of quantiles. In a simple equidepth histogram (Figure 7, center), the vertical axis represents the count, so each bar is about the same height – hence the vertical axis provides minimal information. However, if the *area* of each bin in the plot is viewed as a probability mass, the heights of the bins must be adjusted so that their area is the proper proportion of the total area of the histogram (Figure 7, right), and the result captures both the quantiles (horizontally), and the probability densities (vertically).

Given their relationship to quantiles, equi-depth histograms can be good for communicating robust statistics intuition when given dirty data. For example, they focus attention on the robustly estimable aspects of the data, like the location of the median and the "inter-quartile range": the distance between the 25th and 75th percentiles. They are also less susceptible to "masking" effects that are seen in equi-width histograms. Equi-width histograms are easy to construct in a single pass of the data. Equi-depth histograms are constructed by finding quantiles, and can be done using the approaches mentioned above [Greenwald and Khanna, 2001, Manku et al., 1998]. A survey of other histogram types (typically designed for estimation, not for visualization) appears in [Ioannidis, 2003].

The *Empirical Cumulative Distribution Function* or *ECDF* (Figure 7.1, left) plots the proportion of data points that fall below each value x. The ECDF is sometimes described as an "inverse quantile": the ECDF measures the percentile rank y of each value x. The ECDF shares many properties with the equi-depth histogram normalized to probability densities: it displays both value-based information (on the horizontal axis) and quantile-based information (on the vertical.) Steep portions of the ECDF correspond to dense ranges of values – very much like the tall bins of an equi-width histogram. At the same time, order statistics like medians



Empirical Cumulative Distribution Function

Box-and-whiskers plot

Normal Q-Q plot

Figure 8: Three visualizations for the age data of Section 2.3.

and inter-quartile ranges can be read off easily via the intersection of horizontal lines from the vertical axis with the ECDF curve. The ECDF is also a nice visual aid for comparing multiple attributes with similar value domains, or for comparing a distribution to a model like a normal distribution.

A *box-and-whiskers plot* (Figure 7.1, center), sometimes called a *box plot*, is a compressed representation of a distribution that highlights key robust estimators along with potential outliers. It typically orients the data domain vertically, and places a rectangular box (with some arbitrary width) bounded by the 2nd and 3rd quartiles (the interquartile range), with the median marked in the middle. In addition, a vertical line (the "whiskers") is drawn from a point 1.5 times the box size away from either edge of the box. Any data points that are yet further from the median are plotted explicitly. Of course, a different distance than the interquartile range can be used for the whiskers. Multiple box plots are often placed side-by-side to allow for comparisons of distributions across attributes.

Quantile-Quantile (Q-Q) Plots are a visual means to compare two distributions [Barnett, 1975]. Typically, they are used to compare an empirical distribution to a model – for example, a normal Q-Q plot (Figure 7.1, right) allows one to "eyeball" an attribute for normality testing. They may also be used to compare two empirical distributions. Each axis in a Q-Q plot represents the quantiles of a distribution, with data points plotted at equi-quantile values from the two distributions. In addition, a reference line is drawn, typically between the first and third quartile points. If the two distributions are identical, the line lies at 45 deg and points fall directly on the line. Trends of the points away from the interquartile line indicate regions of the distributions that differ; outliers are easily seen as individual points that away from the line. *Probability-Probability* (P-P) plots are similar, but use the cumulative distribution points on the axes, rather than quantiles.

7.2 Interface Design

A traditional mantra of the data processing world was "Garbage In, Garbage Out" (GIGO), meaning that when given imperfect input, a program cannot be expected to produce useful output. At face value, this is as true today as it was in the 1950's when the phrase was apparently coined[7]. However, the phrase was often used as a way to blame users of data entry systems for failures in analysis, and that attitude has been obviated over the years by two factors. First, there is an acknowledgment in many communities (business, medicine, military) that if human error in controlling computers is widespread and costly, that error needs to be mitigated through design. Second, improvements in the robustness of data analysis and cleaning – including the techniques discussed in this paper – have made it possible to extract quality analyses from (somewhat) dirty data. Modern data-driven organizations are wise to embrace both of these approaches. In this section we discuss the design of data entry interfaces, presenting some high-level rules of thumb that leverage statistical and algorithmic advances discussed earlier in the report.

Data entry is repetitive, tedious and unglamorous – almost certainly the least interesting task in the information lifecycle. It is a perennial source of errors, even when undertaken by highly-motivated users of the data who might also devote enormous energy to other steps in the information lifecycle, from data gathering to analysis. When contracted out, it is typically assigned to low-paid and disengaged employees who get no personal benefit from the accuracy of the data. It is also the subject of surprisingly little research in the literature on data-centric Human-Computer Interaction, which by contrast devotes annual conferences to the final step in lifecycle: data visualization. Most of the writing about introducing quality in the data entry process comes from the MIS and business communities (e.g. Huang et al. [1999]).

Precisely because data entry is uninteresting and repetitive, it calls for interface design that is mindful of both the user experience, and the objectives of the data collection process. A balance has to be struck between the desire of the data-entry user to finish a tiresome job quickly, and the importance of precision and completeness in data collection.

7.2.1 Friction and Prediction

We have spoken repeatedly above of the *spurious integrity* problem; it is time to address it at its source. The problem arises in scenarios where a user enters incorrect data simply to get a form to accept their input. The most common reason for this behavior is the enforcement of *integrity constraints* on the data: rules that ensure completeness and consistency of data entered into the system. This includes requirements for certain fields to be non-empty, or to contain values satisfying quantitative or formatting constraints. Some integrity constraints require disparate data items to be interlinked in particular ways ("referential" integrity). Integrity constraints were invented precisely to keep a database "clean", and avoid the GIGO problem.

However, the enforcement of integrity constraints ignores two key issues that affect data entry interfaces:

1. *Integrity constraints do not prevent bad data.* Integrity constraints are typically underspecified, and even when they are rich they cannot prevent bad data altogether. For example, the requirement that a field be non-empty is not sufficient to ensure that a user provides meaningful contents. Similarly, constraining a field to a valid range of numerical values does not ensure that the values chosen are correct.

2. *Constraint enforcement leads to user frustration.* In many scenarios, users performing data entry have little incentive to ensure data quality. At the same time, they often

[7]Online etymologies of the phrase differ. Some attribute it to George Fuechsel, an IBM 305 RAMAC technician/instructor in New York, in the 1950s. Others attribute it to Wilf Hey, in articles for the magazine PC Plus in the 1980s. Connections are also made to the Gestalt Therapy psychology founder Fritz Perls, whose autobiography is entitled *In and Out the Garbage Pail*

have to enter incomplete and inaccurate data, and in many cases they lack the domain knowledge to fill in or correct that data themselves. When the system does not gracefully accomodate this lack of control and expertise, the user is left with very few options for completing their task.

These two issues, taken together, result in the prevalence of spurious integrity: the frustrated data entry user is typically able to force a large class of "garbage" into the system when frustrated, and often that is their path of least resistance for completing their job.

Recognizing the failings of this traditional approach to data integrity, we propose a few guiding principles for the design of data entry interfaces, which can be realized effectively by an integration of interface design and data analysis:

Principle 1 *Data quality should be controlled via feedback, not enforcement.* Friction *in the feedback should be inversely proportional to likelihood of the data being entered.*

This principle is a relaxation and extension of the notion of integrity constraints. First, it discards the traditional dichotomy of accepting or rejecting inputs, favoring instead a more graduated and user-centric metric of the friction a user encounters during data entry. Second, it embraces a more statistical approach to data cleanliness, that moves away from high-level boolean constraints on data (e.g., "no date can be earlier than 1/1/1970"). Instead, it favors softer constraints that compare new data values to what the system believes to be likely. The notion of likelihood can be derived from a statistical model for the data values, which can be parametric (e.g., a normal distribution with a certain mean and standard deviation) or constructed directly from the existing data in the database (e.g., friction could be proportional to the quantile of the value in the intended database column.)

One might thing this principal would require relatively exotic interface designs. But even very simple interfaces can adopt this philosophy. For example, in simple web-based forms, friction arises from each page load and form submission. Additional friction can be introduced by asking a user to fill out an additional form; e.g., to selectively confirm values that would be outliers if inserted in the database.

Principle 2 *Friction merits explanation.*

When data entry tasks take unusual amounts of effort, users can get frustrated. A frustrated user is less likely to work toward the objectives of the system; instead they may either abandon their task, or turn their attention towards completing their task by any means possible, rather than by working toward the system designer's goals.

A natural way to mitigate frustration is for the interface to provide an intuitive explanation for why the system wants the user to do additional work. This communicates the system designer's objectives to the user directly, rather than indirectly; it may make the user better appreciate the reason for their added efforts. Good explanations can also provide guidance to a user about how to best achieve the system's objectives. However, explanations of this sort must be simple and cannot distract from the task at hand. Ideally, such explanations should be an integral part of the data entry interface, to the point where the user sees no distinction between the visual effect of the task (clicking, dragging, typing) and the rendering of the explanation for the task. We will see a simple example of this below.

Principle 3 *Annotation should be easier than either omission or subversion.*

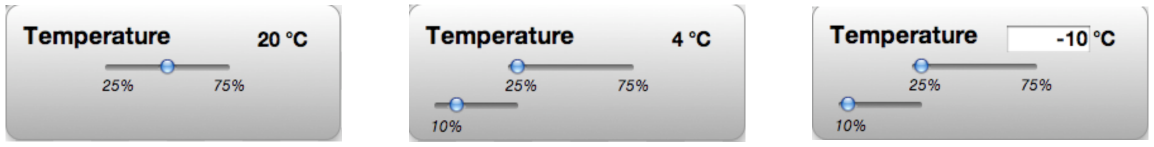


Figure 9: A data entry interface that introduces friction for outliers. Data within the first and third quartiles can be entered via the slider in the leftmost image. When the slider is pulled to the far left, a second slider appears below to allow even lower data to be entered (center image). When that slider is pulled to the far left, an arbitrarily low value may be entered textually (rightmost image).

When faced with an overly high-friction interface, users will often either "give up or give in": i.e., enter no data, or enter data that the system is willing to accept, but which does not correspond to the true data that needs to be entered. Both of these outcomes subvert the goals of the data collection process. Unfortunately, none of the principles above will prevent these results – those principles do not remove the possibility of friction. When friction is significant, users need an alternative mechanism to get as much useful information into the system as possible, and move on to the next task. The best approach here is to (a) allow the user to enter whatever data they have, regardless of integrity, and (b) provide the simplest possible affordances to collect annotations about *why* the user encountered friction in the interface. Annotations can be as simple as free text tagging, recorded speech (perhaps with subsequent speech recognition to convert to text), or even just a single bit of information from a checkbox or button-click that user was particularly frustrated.

These annotations serve a number of purposes. First, tagging of unusual or incomplete data can be very helpful to *post hoc* data cleaning processes, and data cleaning can be much more effective when suspect data is tagged, rather than mixed into the "good" data in an undifferentiated way. Second, annotations can help not only detect unusual data, but provide information for subsequent efforts to correct the data. Finally, annotations can be useful input to the redesign of both data collection protocols and data entry interfaces – in essence one is surveying users *in the moment* when the interface is frustrating their efforts.

In order to harvest annotations, the annotation process must be simpler and more attractive than the generation of missing or dirty data. Note that a certain degree of annotation can be recorded passively by the interface, by automatically tagging data with both the parameters that were used to drive the interface (including previous statistics of the database for data-driven interfaces), and with the user behavior (clickstreams, timing information, etc.) Adding such measures to an interface – and adding facilities to log these usage parameters in the backend database – can serve as a powerful way to identify problems with data and with the data collection and entry processes.

7.2.2 An Example

As an example, we sketch a simple interface for quantitative data entry. Our goal here is not to suggest that this interface is ideal – in fact, it has various flaws unrelated to the scope of this discussion. However, it serves to illustrate the principles above with very familiar and readily available interface widgets. As context, we assume a basic multi-fielded form for entering data into a database.

Consider an integer attribute like temperature, and the following simple interface inspired by the box-and-whisker plots described in Section 7.1. A “slider” is provided that allows the user to smoothly choose a numeric value from a range; the slider is parameterized with summary data from the database, so that its left and right edges are a modest distance from the median, say the first and third quartiles, akin to the box in a box-and-whisker plot. A mockup is shown on the left of Figure 9. To enter data outside the “usual” range, the user drags the slider fully to one side, which brings up another separate slider to go some distance further, akin to the whiskers in the plot. This is show in the center of Figure 9. If the user needs to go even further, they drag the second slider all the way in the same direction, bringing up a text entry field, allowing them to type in an explicit value; this value could be validated to be beyond the range of values presented by the previous sliders. This is akin to the dots in a box-and-whisker plot, and a mockup is shown on the right of Figure 9. Some salient points to note about this interface:

- The interface predicts that most data will be in the interquartile range of the distribution; it gets the quantile information from the previously-stored values in the database. The least friction occurs for data that is within the interquartile range.
- The interface integrates the user’s experience of friction with an interactive exploration of the data distribution. When the user is entering an unusual value, they are shown just how unusual it is relative to other data in the database. The friction in navigating the data entry interface is inseparably linked with a display of how unusual the item is relative to other data.
- This interface grossly violates the notion that annotation should be easier than subversion, because of its naive use of sliders. Sliders have a default value when first rendered, so the least effort results in choosing the default value. A better design would guarantee that choosing an invalid default is harder than annotating the data as being unverified. To that end, an “unknown” button could be placed next to the slider and checked by default; the slider interface would only be enabled when the “unknown” button was unchecked.

We observe that the above interface is unattractive for a number of reasons, not least that it is clumsy to use for entering typical data values, since setting a slider accurately is time-consuming. However, it illustrates within a simple and traditional interface how outlier metrics from robust statistics can be incorporated into interfaces to help meet the three design principles above relatively easily.

8 Recommendations and Additional Reading

Ideally, an organization concerned about data cleanliness will undertake a comprehensive review of the full data lifecycle. It should be understood that improvements to this lifecycle are necessarily iterative: changes in one stage often suggest additional changes at other stages, and a successful campaign to improve data quality will embrace this iterative process over significant periods of time.

Within the scope of this paper, we focus here on a staged implementation of the techniques presented above. As a matter of pragmatism, we focus on the techniques that are relatively easy to understand, explain and implement efficiently.

The first basic step toward post-hoc data cleaning is to implement one or more of the schemes for univariate outlier detection in Section 2. To that end, some simple planning questions can be asked regarding the difficulty of that task. The most basic question is to assess the size of the data on which outlier detection will run. If the data is relatively small and will easily fit in RAM, tools like the R project can be used to relatively easily implement robust estimators like the median and MAD, and outlier detection techniques like Hampel X84. Note that R has libraries for robust statistics `robustbase` and `rrcov` [Todorov, 2006].

For somewhat larger data, it may be necessary to perform outlier detection within the database, by implementing scalable algorithms for robust quantiles. The "vanilla SQL" of Section 2.7.1 is unlikely to be effective on even medium-sized data. The sort-based implementations of Section 2.7.2 are likely to perform adequately for medium-sized tables with a few quantitative columns, particularly if they are implemented as stored procedures within the database engine. However, for larger tables with more data and/or more numeric columns, it is advisable to research and implement one of the approximate quantile algorithms of Section 2.7.3, since they run in a single pass of the data regardless of data size or the number of columns being analyzed simultaneously. The original algorithm of Manku, et al. [Manku et al., 1998] is a reasonable choice.

Efficient quantile algorithms are a building block not only for robust outlier detection, but for data visualization as well. They directly enable visualizations including boxplots, equidepth histograms, and Q-Q plots. These can be of great use to the analyst performing data cleaning. Note also that they can be very useful for embedding into data entry interfaces, as illustrated in Section 7.

Once univariate outlier detection has been implemented, a next natural task is to implement a univariate "iceberg" or "heavy hitter" query to identify values with unusual numbers of copies. This implementation can be based again on an approximate quantile estimator. Heavy hitters are often best treated by enabling domain experts to "eyeball" them: typically there are only a few, and they are very obviously either reasonable or garbage.

Moving to multivariate approaches is technically challenging. First, in terms of implementation, there is no simple and efficient implementation in SQL beyond two variables at a time. Second, in educational terms, multivariate outliers can be hard to understand and explain, requiring more sophistication from the system designers and end-users. If multivariate approaches are considered important, the simplest implementation strategy for large databases appears to be the "Rescaling via robust componentwise estimators" approach of Section 3.2.

Outlier detection in timeseries data continues to be a topic of research in the Statistics literature, which merits watching over time. Simple schemes based on windowing, as described in Section 4, are reasonably attractive in this context, both for their ease of implementation, and their natural interpretation.

Finally, we emphasize the importance of data entry design, and the linkage between entry and analysis. By feeding properties of the data distribution into the data entry interface as illustrated in Section 7.2, it is possible to give instantaneous visual feedback to data-entry users about potential errors in their data. Even simple and traditional interface widgets can be used in intelligent ways to ensure that users are careful and informed when doing data entry. Richer data visualizations such as those of Section 7.1 can be more creatively incorporated into data entry interfaces as well. These kinds of feedback can help reduce errors in the data, and frustration among data-entry users. An especially important aspect of this issue is to make it easy and natural for data-entry users to annotate any data that may be dirty or incomplete, rather than require them to surreptitiously but intentionally introduce dirty data into a database.

8.1 Recommended Reading

One useful introductory resource on data cleaning for database professionals is Dasu and Johnson's book [Dasu and Johnson, 2003]. It surveys the field, and covers a number of the quantitative data cleaning issues addressed here. Another more compressed resource is the survey slide deck on Outlier Detection by Chawla and Sun.

On the statistical front, Rousseeuw and Leroy's book on Robust Statistics is recommended as an introduction to that field that should be digestable to many computer scientists, even if they have modest background in statistics [Rousseeuw and Leroy, 1987]. It is far more approachable than the other classic text in the area by Huber [Huber, 1981]. Jarrell focuses specifically on surveying multidimensional outliers [Jarrell, 1991]. Tukey's classic book *Exploratory Data Analysis* is still recommended, despite its outdated focus on pencil-and-paper exercises [Tukey, 1977].

Efficient and approximate computation of summary statistics in a single pass has been a topic of interest in the database research community over the last decade. Some of that work covers robust estimators including medians and other order statistics. Garofalakis and colleagues have excellent tutorial slides overviewing this work [Garofalakis and Gibbons, 2001, Garofalakis et al., 2002]. Muthukrishnan has written a survey article on the topic as well [Muthukrishnan, 2005].

A useful counterpoint to this report is the work of Chapman, which focuses on cleaning data in a biodiversity setting [Chapman, 2005]. Some of the focus in that document on special case data like taxonomic labels and geocoding may be of broad use beyond biological applications.

With respect to data visualization, Cleveland's *Visualizing Data* is a useful resource favored by statisticians [Cleveland, 1993]. Few's *Show Me the Numbers* is a somewhat more approachable discussion of the topic [Few, 2004]. Tufte has a number of very influential and more wide-ranging books on the history, esthetics and communicative power of various data visualization ideas [Tufte, 2001, 1990, 2006]; the strong personal sensibility behind his work has evoked a good deal of interest and controversy over the years.

Piatetsky-Shapiro has a web page with many resources on commercial and research efforts in data cleaning [Piatetsky-Shapiro, 2008].

References

Chris Anderson. *The Long Tail: Why the Future of Business is Selling Less of More.* Hyperion, 2006.

V. Barnett. Probability plotting methods and order statistics. *Applied Statistics*, 24, 1975.

S. D. Bay and M. Schwabacher. Mining distance-based outliers in near linear time with randomization and a simple pruning rule. In *Proc. ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2003.

Pavel Berkhin. Survey of clustering data mining techniques. Technical report, Accrue Software, San Jose, CA, 2002. `http://citeseer.ist.psu.edu/berkhin02survey.html`.

I. Bhattacharya and L. Getoor. Collective entity resolution in relational data. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 1(1), 2007.

A. M. Bianco, M. Garca Ben, E. J. Martnez, and V. J. Yohai. Outlier detection in regression models with arima errors using robust estimates. *Journal of Forecasting*, 20(8), 2001.

George Box, Gwilym M. Jenkins, and Gregory Reinsel. *Time Series Analysis: Forecasting & Control (3rd Edition).* Prentice Hall, 1994.

M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. Lof: Identifying density-based local outliers. In *Proc. ACM SIGMOD International Conference on Management of Data*, 2000.

Shannon C.E. and Weaver W. *The Mathematical Theory of Communication.* University of Illinois Press, Urbana and Chicago, 1949.

Laha Chakravarti and Roy. *Handbook of Methods of Applied Statistics, Volume I.* John Wiley and Sons, 1967.

A. D. Chapman. Principles and methods of data cleaning primary species and species-occurrence data, version 1.0. Technical report, Report for the Global Biodiversity Information Facility, Copenhagen, 2005. `http://www.gbif.org/prog/digit/data_quality/DataCleaning`.

William S. Cleveland. *Visualizing Data.* Hobart Press, 1993.

Graham Cormode and S. Muthukrishnan. What's hot and what's not: Tracking most frequent items dynamically. In *In Proc. ACM Principles of Database Systems (PODS)*, 2003.

R.B. D'Agostino and E.S. Pearson. Tests for depature from normality: Empirical results for the distributions of b_2 and $\sqrt{b}_1$. *Biometrika*, 60, 1974.

Tamraparni Dasu and Theodore Johnson. *Exploratory Data Mining and Data Cleaning.* Wiley, 2003.

Tamraparni Dasu, Theodore Johnson, S. Muthukrishnan, and Vladislav Shkapenyuk. Mining database structure; or, how to build a data quality browser. In *Proc. ACM SIGMOD International Conference on Management of Data*, 2002.

Xin Dong, Alon Halevy, and Jayant Madhavan. Reference reconciliation in complex information spaces. In *Proc. ACM SIGMOD international conference on Management of data*, 2005.

Min Fang, Narayanan Shivakumar, Hector Garcia-Molina, Rajeev Motwani, and Jeffrey D Ullman. Computing iceberg queries efficiently. In *Proc. International Conference on Very Large Databases (VLDB)*, 1998.

Stephen Few. *Show Me the Numbers: Designing Tables and Graphs to Enlighten.* Analytics Press, 2004.

David F. Findley, Brian C. Monsell, William R. Bell, Mark C. Otto, and Bor-Chung Chen. New capabilities and methods of the x-12-arima seasonal-adjustment program. *Journal of Business & Economic Statistics*, 16(2), 1998.

Philippe Flajolet and G. Nigel Martin. Probabilistic counting algorithms for data base applications. *Journal of Computer and Systems Sciences*, 31(2), 1985.

B. Roy Frieden. *Science from Fisher Information.* Cambridge University Press, 2004.

Minos N. Garofalakis and Phillip B. Gibbons. Approximate query processing: Taming the terabytes. In *International Conference on Very Large Data Bases (VLDB)*, 2001. Tutorial. Slides available online at `http://www.cs.berkeley.edu/~minos/Talks/vldb01-tutorial.ppt`.

Minos N. Garofalakis, Johannes Gehrke, and Rajeev Rastogi. Querying and mining data streams: You only get one look. In *ACM-SIGMOD International Conference on Management of Data*, 2002. Tutorial. Slides available online at `http://www.cs.berkeley.edu/~minos/Talks/streams-tutorial02.pdf`.

Luis Gravano, Panagiotis G. Ipeirotis, Nick Koudas, and Divesh Srivastava. Text joins for data cleansing and integration in an rdbms. In *Proc. International Conference on Data Engineering (ICDE)*, 2003.

Michael Greenwald and Sanjeev Khanna. Space-efficient online computation of quantile summaries. In *Proc. ACM SIGMOD International Conference on Management of Data*, 2001.

Frank R. Hampel, Elvezio M. Ronchetti, Peter J. Rousseeuw, and Werner A. Stahel. *Robust Statistics - The Approach Based on Influence Functions.* Wiley, 1986.

P. M. Hartigan. Computation of the dip statistic to test for unimodality. *Applied Statistics*, 34 (3), 1985.

John Hershberger, Nisheeth Shrivastava, Subhash Suri, and Csaba D. Tóth. Space complexity of hierarchical heavy hitters in multi-dimensional data streams. In *Proc. ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, 2005.

Kuan-Tsae Huang, Yang W. Lee, and Richard Y. Wang. *Quality Information and Knowledge Management.* Prentice Hall, 1999.

Peter. J. Huber. *Robust Statistics.* Wiley, 1981.

Ykä Huhtala, Juha Kärkkäinen, Pasi Porkka, and Hannu Toivonen. TANE: An efficient algorithm for discovering functional and approximate dependencies. *The Computer Journal*, 42 (2):100–111, 1999.

Yannis Ioannidis. The history of histograms (abridged). In *Proc. International Conference on Very Large Data Bases (VLDB)*, 2003.

Michele G. Jarrell. Multivariate outliers. review of the literature. In *Proc. Annual Meeting of Mid-South Educational Research Association*, 1991. `http://www.eric.ed.gov/ERICDocs/data/ericdocs2sql/content_storage_01/0000019b/80/23/61/0b.pdf`.

Theodore Johnson and Tamraparni Dasu. Comparing massive high-dimensional data sets. In *Knowledge Discovery and Data Mining*, 1998.

Regina Kaiser and Agustín Maravall. Seasonal outliers in time series. Banco de España Working Papers 9915, Banco de España, 1999. available at `http://ideas.repec.org/p/bde/wpaper/9915.html`.

E. M. Knorr and R. T. Ng. Finding intensional knowledge of distance-based outliers. In *Proc. International Conference on Very Large Data Bases*, 1999.

G. Kollios, D. Gunopulos, N. Koudas, and S. Berchtold. Efficient biased sampling for approximate clustering and outlier detection in large data sets. *IEEE Transactions on Knowledge and Data Engineering*, 15(5), 2003.

Regina Y. Liu, Jesse M. Parelius, and Kesar Singh. Multivariate analysis by data depth: descriptive statistics, graphics and inference. *Annals of Statistics*, 27(3), 1999.

J.F. MacGregor and T.J. Harris. The exponentially weighted moving variance. *Journal of Quality Technology*, 25(2), 1993.

G.S. Maddala and C.R. Rao. *Handbook of Statistics 15: Robust Inference.* Elsevier, 1997.

Gurmeet Singh Manku and Rajeev Motwani. Approximate frequency counts over data streams. In *Proc. International Conference on Very Large Data Bases (VLDB)*, 2002.

Gurmeet Singh Manku, Sridhar Rajagopalan, and Bruce G. Lindsay. Approximate medians and other quantiles in one pass and with limited memory. In *Proc. ACM SIGMOD International Conference on Management of Data*, 1998.

R. Maronna and R. Zamar. Robust estimation of location and dispersion for high-dimensional data sets. *Technometrics*, 44(4), 2002.

Michael Martin and Steven Roberts. An evaluation of bootstrap methods for outlier detection in least squares regression. *Journal of Applied Statistics*, 33(7), 2006.

M. Mitzenmacher. A brief history of generative models for power law and lognormal distributions. *Internet Mathematics*, 1(2), 2004.

S. Muthukrishnan. Data streams: Algorithms and applications. *Foundations and Trends in Theoretical Computer Science*, 1(1), 2005.

M.C. Otto and W.R. Bell. Two issues in time series outlier detection using indicator variables. In *Proc. American Statistical Association, Business and Economics Statistics Section*, 1990.

Gregory Piatetsky-Shapiro. Kdnuggets : Solutions : Data providers and data cleaning, 2008. `http://www.kdnuggets.com/solutions/data-cleaning.html`.

R Project. The R project for statistical computing, 2008. `http://www.r-project.org/`.

Raghu Ramakrishnan and Johannes Gehrke. *Database Management Systems, 4th Edition.* McGraw-Hill, 2002.

Vijayshankar Raman and Joseph M. Hellerstein. Potter's wheel: An interactive data cleaning system. In *Proc. International Conference on Very Large Data Bases (VLDB)*, 2001.

S. Ramaswamy, R. Rastogi, and K. Shim. Efcient algorithms for mining outliers from large data sets. In *Proc. ACM SIGMOD International Conference on Management of Data*, 2000.

P. J. Rousseeuw and K. Van Driessen. A fast algorithm for the minimum covariance determinant estimator. *Technometrics*, 41(3), 1999.

Peter J. Rousseeuw and Annick M. Leroy. *Robust Regression and Outlier Detection.* Wiley, 1987.

David Rozenshtein, Anatoly Abramovich, and Eugene Birger. *Optimizing Transact-SQL : Advanced Programming Techniques.* SQL Forum Press, 1997.

S. Sarawagi. User-cognizant multidimensional analysis. *The VLDB Journal*, 10(2), 2001.

S. S. Shapiro and M. B. Wilk. An analysis of variance test for normality (complete samples). *Biometrika*, 52(3 and 4), 1965.

P. Singla and P. Domingos. Entity resolution with markov logic. In *IEEE International Conference on Data Mining (ICDM)*, 2006.

Yannis Sismanis, Paul Brown, Peter J. Haas, and Berthold Reinwald. Gordian: efficient and scalable discovery of composite keys. In *Proc. International Conference on Very Large Data Bases (VLDB)*, 2006.

Graphpad Software. Normality tests, 2008. `http://www.graphpad.com/index.cfm?cmd=library.page&pageID=24&categoryID=4`.

Michael Stonebraker. Inclusion of new types in relational data base systems. In *Proc. International Conference on Data Engineering*, 1986.

Gilbert Strang. *Introduction to Linear Algebra, Third Edition.* SIAM, 2003.

Valentin Todorov. Robust location and scatter estimators for multivariate analysis. In *useR!, The* R *User Conference*, 2006. `http://www.r-project.org/user-2006/Slides/Todorov.pdf`.

Ruey S. Tsay. Outliers, level shifts, and variance changes in time series. *Journal of Forecasting*, 7(1), 1988.

Edward R. Tufte. *The Visual Display of Quantitative Information, 2nd Edition.* Graphics Press, 2001.

Edward R. Tufte. *Envisioning Information.* Graphics Press, 1990.

Edward R. Tufte. *Beautiful Evidence.* Graphics Press, 2006.

John Tukey. *Exploratory Data Analysis.* Addison-Wesley, 1977.

William E. Winkler. Overview of record linkage and current research directions. Technical report, Statistical Research Division, U.S. Census Bureau, 2006. `http://www.census.gov/srd/papers/pdf/rrs2006-02.pdf`.

George K. Zipf. *Human Behaviour and the Principle of Least-Effort*. Addison-Wesley, 1949.