3

Data

The word **data** is the plural of the Latin *datum*, meaning a given, or that which we take for granted and use as the basis of our calculations. This meaning is carried in the French word for statistical data, *données*. We ordinarily think of data as derived from measurements from a machine, survey, census, test, rating, or questionnaire — most frequently numerical. In a more general sense, however, data are symbolic representations of observations or thoughts about the world. As we have seen, we do not need to begin with numerals to create a graphic. Text strings, symbols, shapes, pictures, graphs themselves, can all be graphed.

Older graphing systems required data to be stored in **flat files** containing row-by-column arrays of numerical observations on a set of variables. Spreadsheet graphics systems also arrange data this way. In these systems, the structure of the data determines the types of displays possible: line charts, bar charts, pie charts, or scatterplots. More recent object-oriented graphics systems assume their data are in a **data source**. And the most flexible systems provide an object-oriented interface to the data source which makes no assumptions about the underlying structure of the data. The graphing system itself has a **view** of the data that it uses for its initial mapping. This view may have no simple relation to the actual organization of the data source. In addition, the view may change from moment to moment because the system may be fed by a **streaming** data source. A well-designed graphics system must be able to handle these situations in order to avoid static views that can misrepresent the underlying data.

This chapter outlines three types of data: empirical, abstract, and meta. These types are distinguished by their sources and function. Empirical data are collected from observations of the real world. Abstract data arise from formal models. And metadata are data on data. Treinish (1993) provides further details on these types and elucidates them within the general context of graphics production. Fortner (1995) discusses empirical and abstract scientific data. We will first summarize data functions and then discuss these three types of data. We will conclude with a brief section on the emerging field of data mining and its relationship to the models needed to support a graphical system.

3.1 *Data Functions*

Table 3.1 lists some functions we will use in this chapter to create variables from datasets. We use the convention that the `typewriter` letters refer to column names (domain names) of a relation. The results of these functions are function variables. As we mentioned in the last chapter, we could apply these functions to the contents of an object-oriented database as long as we devised a relational interface to insure that related sets are indexed properly. In any case, the actual referencing scheme we need is independent of the physical or formal organization of the data themselves.

Table 3.1 **Data Functions**

Empirical Data	*Abstract Data*	*Metadata*
col(*source*(), *name*(), *unit*(), *weight*()) *map(source*(), *id*()) *stream(source*(), *id*()) *image(source*()) *sample*(x, *n*) *reshape*(x_1,...,x_n , "<*index*>")	*iter*(*from*,*to*,*step*) *mesh*(*min*, *max*, *n*) *count*(*n*) *proportion*(*n*) *percent*(*n*) *constant*(*c, n*) *string*("<*string*>", *n*) *rand*(*n*)	*meta*(*source*(), *name*())

The empirical functions operate on columns of observed data. The *col*() function links a variable to a column in a data source. If this function is not used in a specification, then the names of variables are assumed to be the same as the names of fields in the data source. The *unit*() argument assigns measurement units. The *weight*() argument maps a column to case weights used for statistical calculations. The *map*() function is for maps, the *stream*() function is for streaming data sources, and the *image*() function is for image sources.

The *sample*() function implements sampling. Sampling methods include *sample.srs* (simple random), *sample.jackknife* (Tukey, 1958) and *sample.boot* (Efron & Tibshirani, 1993). The *reshape*() function reshapes a matrix or table of columns (corresponding to an *n*-ary relation) into a single variable. Let $\mathbf{X}_{m\times n}$ be a matrix produced by concatenating the columns (x_1,...,x_n). Let i and j be row and column indices respectively of the matrix $\mathbf{X}$ ($i = 1, \ldots, m$ and $j = 1, \ldots, n$). Let k be the row index of the variable x output by the *reshape*() function. The *reshape*() functions compute the index k as follows:

$$\begin{aligned}
&\textit{reshape.rect}(): && k = n \cdot (i-1) + j \\
&\textit{reshape.tri}(): && k = i \cdot (i-1)/2 + j : (i \geq j) \\
&\textit{reshape.low}(): && k = (i-1) \cdot (i-2)/2 + j : (i > j) \\
&\textit{reshape.diag}(): && k = i : (i = j)
\end{aligned}$$

Note the name of the function is what it converts *from*, not *to*. The *reshape.rect*() function unwraps the rows of a rectangular matrix into a single column. The *reshape.tri*() function unravels the lower triangular half (including the diagonal) of a square matrix into a single column. The *reshape.low*() function does the same for the triangular half excluding the diagonal. And the *reshape.diag*() function places the diagonal of a square matrix into a column.

The <*index*> argument to the *reshape*() function has several alternatives:

value:	the value of the entries
rowindex:	the row index
colindex:	the column index
rowname:	the row name
colname:	the column name

The *reshape*() functions leave one with the impression that a matrix algebra package could provide many (but not all) of the functions needed for structuring data at the first stage of a graphics system. Many of the graphics we draw in practice depend on a matrix data model. Notable exceptions, however, are geographic, physical, and mathematical objects that must depend on different data organizations.

The abstract functions create columns. They operate across rows and most take an argument (*n*) for the number of rows; if the argument is omitted and we are processing a real dataset, *n* is the number of rows in the dataset. The *iter*() function is a simple iterator for creating arithmetic series. The series *iter*(1, 10, 1) contains a sequence of the integers 1 through 10, for example. The *mesh*() functions computes a 1D, 2D, or 3D mesh. The *count*(), *proportion*(), and *percent*() functions are iterators as well. The *count*(10) function, for example, is equivalent to *iter*(1, 10, 1). A comparable *proportion*(10) iterator would be equivalent to *iter*(.1, 1.0, .1). Finally, the *constant*() and *string*() iterators supply *n* instances of an item and the *rand*() iterator generates independent random numbers. The *rand*() iterator has methods such as *rand.uniform*() and *rand.normal*(). We will occasionally use <string> as shorthand for the *string*("<*string*>") function, *e.g.*,

```
DATA: s = string("Hello world")
DATA: s = "Hello world"
```

The *meta*() function associates metadata with rows of datasets. We have included an atomistic example to suggest the association of a single metadata item with a row of a dataset. Each index in the *source*() of the *meta*() function might point to a video image, a Web address in the form of a Universal Resource Locator (URL), or some other item of metadata.

Finally, there is a system variable that may be used in place of user variables. It is an identity variable, notated **1**, which is an identity element for the cross and nest operators in graphics algebra. It is discussed in Section 5.1.3.3.

3.2 *Empirical Data*

Empirical data arise from our observations of the world. The Greek term *ἐμπειρία*, the source of the English word *empirical*, means experience or acquaintance with some thing. Among the many prevailing views of the role of empirical data in modern science, there are two opposing extremes. On the one hand, the **realist** assumes data are manifestations of latent phenomena. In this view, data are pointers to universal, underlying truths. On the other hand, the **nominalist** assumes data *are* what they describe. From this latter point of view, as the philosopher Ludwig Feuerbach noted, "you are what you eat." We use the historical realist-nominalist terminology to emphasize that these differing perspectives, in one form or another, have origins in Medieval theological controversies and their classical antecedents in Greek philosophy.

Many working scientists adopt some form of a realist position, particularly those in the natural sciences. Even social scientists, despite the myriad particulars of their field, have often endorsed a similar position. Watson (1924) and Hull (1943), for example, believed that behavioral data could be explained best through universal quantitative laws. More recently, Webb *et al.* (1981) promoted "unobtrusive" data collection methods that point to or help us triangulate on basic truths about behavior.

Alternatively, operationalists like Skinner (1969) have argued that behavioral data are best understood by avoiding unobservables and inferences. Skinner even rejected statistical modeling, arguing that smoothing or aggregation obscures details that could help falsify theories. The graphics in Skinner (1961), for example, are the cumulative traces over time by a stylus connected to a lever or button pressed by a pigeon ("untouched by human hands"). Figure 3.1 shows an example of this type of graphic. The horizontal axis marks the linear movement of the stylus over time. The vertical axis marks the increment of the stylus following each bar press. The vertical trace lines are due to resetting of the stylus to keep it on a single page. Skinner argued that these graphics are sufficient representations of responses to schedules of reinforcement. He contended that smoothing, interpolation, aggregation, or model fitting only hide the associations that the scientist needs to see in order to refine theory.

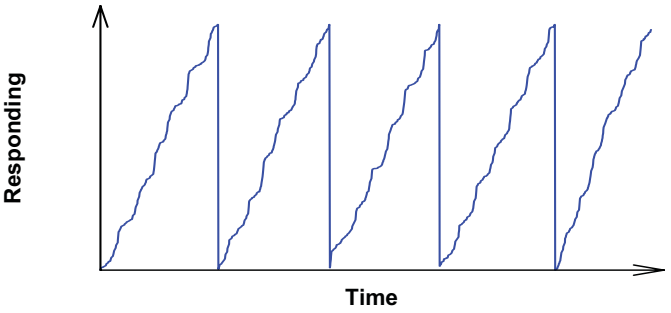


Figure 3.1 *Cumulative record*

Skinner's cumulative-record graphic would appear to short-circuit the diagram presented in Figure 2.2. It would seem that the physical production of such a graphic obviates the need for data functions, variables, indeed, a computer. In fact, however, Skinner's hard-wired device performs our data functions implicitly. Each pen movement involves a sequential indexing of the bar press events and time increments. Skinner's variables are Time and Cumulative Response Count.

Computer data acquisition systems now perform the functions originally designed into mechanical lab equipment like Skinner's. In doing so, they enable alternative data organization and views not always anticipated by the scientist collecting the data. Consider Figure 3.1. It is visually indistinguishable from several of the cumulative records in Ferster and Skinner (1957). However, if we had Skinner's raw data we could use modern statistical time-series methods (unavailable to Skinner) to show that the data underlying Figure 3.1 were not produced by a real organism (human, mouse, *or* pigeon) in an operant conditioning chamber. They were generated by a computer from a stochastic equation. This detective work would involve producing a different graphic from the same data. We can sometimes image-scan older graphics to retrieve data, but there is no substitute for having the original observations and devising our own analyses.

The following examples illustrate some methods for referencing and organizing raw data to produce variables that can be graphed. The first example shows how variables are not always derived from columns in files. The second shows how they may be derived from repeated random samples.

3.2.1 Reshaping Data

Since any matrix is transposable, we can convert row data into column data and graph them when appropriate (*e.g.*, Jobson, 1992, p. 428). Or, as Figure 3.2 shows, we can construct a graphic based on a subset of a matrix.

A correlation matrix is a symmetric matrix of the correlations among a set of variables. If our received data are in a correlation matrix and we wish to graph the correlations, we have to restructure them into one variable using the *reshape.low*() function. This takes the lower-triangular elements of a symmetric matrix and strings them into one variable.

```
DATA: r = reshape.low(pounding, sinking, shaking, nauseous, stiff,
          faint, vomit, bowels, urine, "value")
ELEMENT: point.dodge.asymmetric(position.bin.dot(r))
```

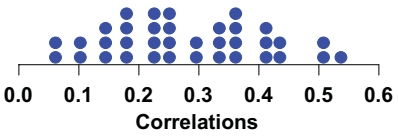


Figure 3.2 *Dot plot of correlations from correlation matrix*

These correlations are based on recalled experiences of symptoms of combat stress (1=yes, 0=no) among soldiers who experience battle in World War II. These data fit a Guttman scale (Stouffer *et al*., 1950). A Guttman scale is an ordinal relation that Guttman called a **simplex**. Presence of a given symptom implies presence of all milder symptoms and absence of a given symptom implies absence of all stronger symptoms. Such a relation implies that the matrix of correlations among symptoms should have all positive elements that follow a banded structure, with the largest correlations near the diagonal and the smallest near the corner opposite the diagonal.

We can display this structure by reshaping the correlation matrix in a different way. This time, we will create three variables — a row index, a column index, and the value of the correlation corresponding to each combination of these indices. Then we will plot the entire correlation matrix using color to represent the value of each correlation. This method works well in this example because all the correlations (as expected for a simplex) are positive.

Figure 3.3 reveals this structure. We have used the *reshape*() data functions to derive indices for plotting the rows and columns of the correlation matrix directly and a *polygon* graph within binning (see Chapter 8) to represent the correlations as colored rectangles. Notice that the colors get warmer as we approach the diagonal.

```
DATA: row = reshape.tri(pounding, sinking, shaking, nauseous,
      stiff, faint, vomit, bowels, urine, "rowname")
DATA: col = reshape.tri(pounding, sinking, shaking, nauseous,
      stiff, faint, vomit, bowels, urine, "colname")
DATA: r = reshape.tri(pounding, sinking, shaking, nauseous, stiff,
      faint, vomit, bowels, urine, "value")
ELEMENT: polygon(position(bin.rect(col*row)), color.hue(r))
```

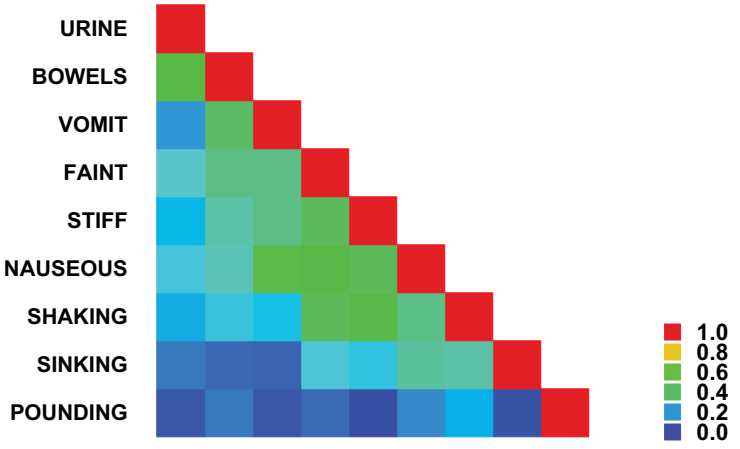


Figure 3.3 *Correlation matrix of combat symptoms*

3.2.2 Bootstrapping

Efron and Tibshirani (1993) discuss a procedure for repeatedly computing a statistic using random samples (with replacement) from a dataset. This procedure, called **bootstrapping**, offers a way to compute confidence intervals when conventional methods are inefficient or unavailable. A bootstrap sample is a sample with replacement from a dataset. Usually, the size of the sample is the same size as the dataset, so that some cases are sampled more than once. Figure 3.4 shows an example of a histogram of a dataset and a histogram of means from 1,000 bootstrap samples of that dataset. The variable mean is indexed to a set of bootstrapped means, each of which was computed from the original dataset. This operation is equivalent to computing the mean of a set of values by making a single pass through a column of a database, repeating this database query 1,000 times, and then assembling all the means into a column of 1,000 values. This computationally intensive operation is not one calculated to endear a user to a database administrator. Although automated boostrapping is not available in database systems, some statistical packages that have data management subsystems do perform bootstrapping.

ELEMENT: *interval*(*position*(*summary.count*(*bin.rect*(military, *dim*(1)))))

DATA: mean = *sample.boot*(military, 1000, "mean")
ELEMENT: *interval*(*position*(*summary.count*(*bin.rect*(mean, *dim*(1)))))

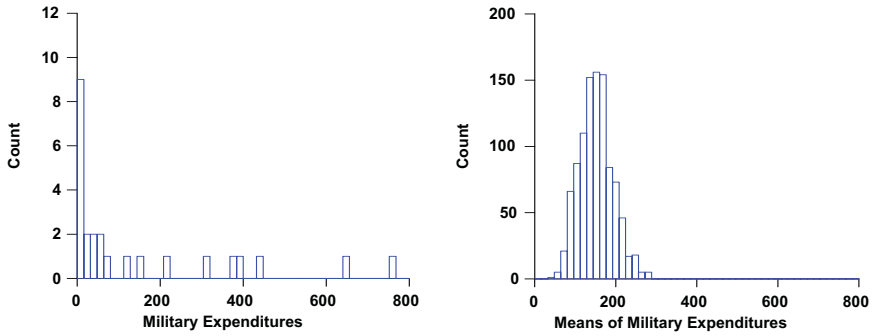


Figure 3.4 *Bootstrapped means*

Two salient contrasts are apparent in Figure 3.4. The first is that the histogram of bootstrapped means is less positively skewed. The second is that the variation in the bootstrapped means is smaller. The standard deviation of the values in the left histogram is 214.1 and the standard deviation of the mean values in the right histogram is 42.3. These differ by a factor of approximately 5, which is the square root of the sample size, 25. The bootstrapping results are consistent with the standard statistical theory for the sampling distribution of means. Nevertheless, the distribution of means is still positively skewed. If the sample size were larger than 25, the means would look more normally distributed.

3.3 Abstract Data

Abstract data functions are most often used to create variables consisting of series, lattices, and other indexing schemes that we use to arrange observed data values. There are many generating functions for series that are useful in the data manipulation needed for statistical graphics. Knuth (1968) discusses some of these functions. Sometimes we construct graphics using abstract data that are determined by a mathematical rule or function. We will first present some simple examples of the former and then one example of the latter.

3.3.1 Time Series

Time series datasets do not always include a column for time. Even when they do, we sometimes wish to plot against the index of the measurement rather than against time. Figure 3.5 contains 256 instantaneous firing rate measurements of a single cat retinal ganglion cell during a 7 second interval (Levine *et al*., 1987). The points are connected by a *line* graph. Most time series packages automatically create this index variable for equally spaced time series; some go further and code the series in calendar time. In this example, we have used the *iter* data function to create a time index running from 1 to 256.

We have chosen an aspect ratio for this plot that reveals a low-frequency component of roughly one-and-a-half cycles over the duration of the series. This component is due to the respiration of the cat, according to Michael Levine (personal communication). Firing rate tends to increase with more oxygen in the blood.

DATA: case = *iter*(1, 256, 1)
ELEMENT: *line*(*position*(case*rate))

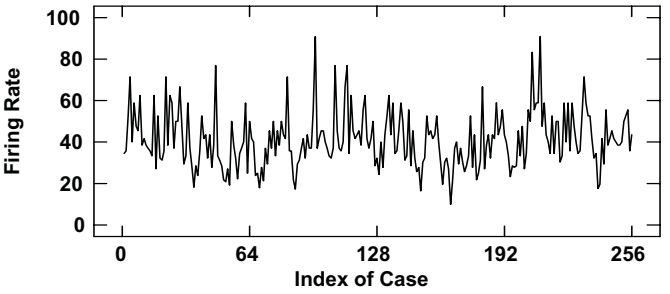


Figure 3.5 *Firing rate of cat retinal cell ganglion*

3.3.2 *Counts*

Bar charts of counts often need to be constructed when there is no explicit count variable in the data. Figure 3.6 shows a bar chart constructed from the countries data used in Chapter 8. The *constant*() implicit function fills a column with ones. The *summary.count* statistical function (see Chapter 7) tallies the counts in all combinations of **gov** and **urban** values. Notice that we did not use the *bin* statistical function as we did for histograms. The data are already classified into distinct groups by the categorical variables. In some histogram examples and bar charts of counts, we omit the constant variable. In the GPL language, we assume the *summary.count* function constructs it if it is missing.

```
DATA: z = constant(1)
COORD: rect(dim(1, 2, 3))
ELEMENT: interval(position(summary.count(gov*urban*z)))
```

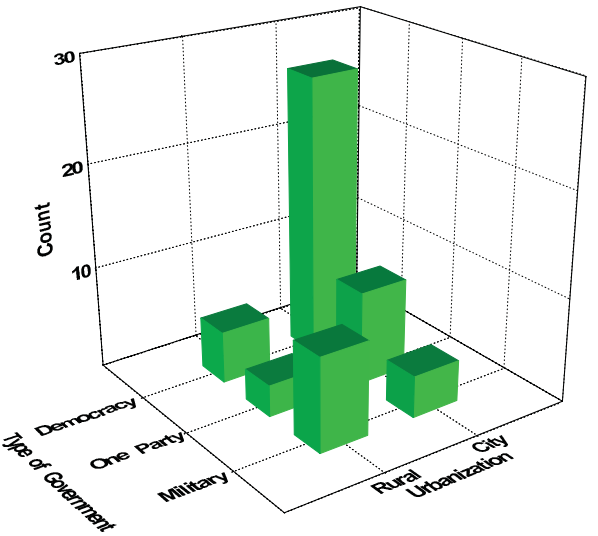


Figure 3.6 *Count bar chart*

3.3.3 *Mathematical Functions*

Data for a graphing system may exist only as an abstraction. This happened in a system developed for automatically graphing interesting subsets of mathematical functions (Wilkinson, 1993a). The data for this system are expressed by an equation. The system examines the equation to determine bounded, periodic, and asymptotic behavior and then chooses a suitable range and domain for graphing the function. The system then renders the function based on its analysis.

Figure 3.7 shows an example of such a plot using a polar arc-tangent function. The system determined that the behavior outside of $[-5 < x,y < +5]$ in the domain could be inferred relatively accurately by looking at the behavior in the visibly graphed domain. It scaled the range to $[0 < f(x, y) < .9]$ to encompass the interesting behavior there as well.

```
DATA: x, y = mesh(min(–5, –5), max(5, 5))
TRANS: z = (x^2+y^2)/sqrt(x^2+y^2
COORD: rect(dim(1, 2, 3))
ELEMENT: surface(position(x*y*z))
```

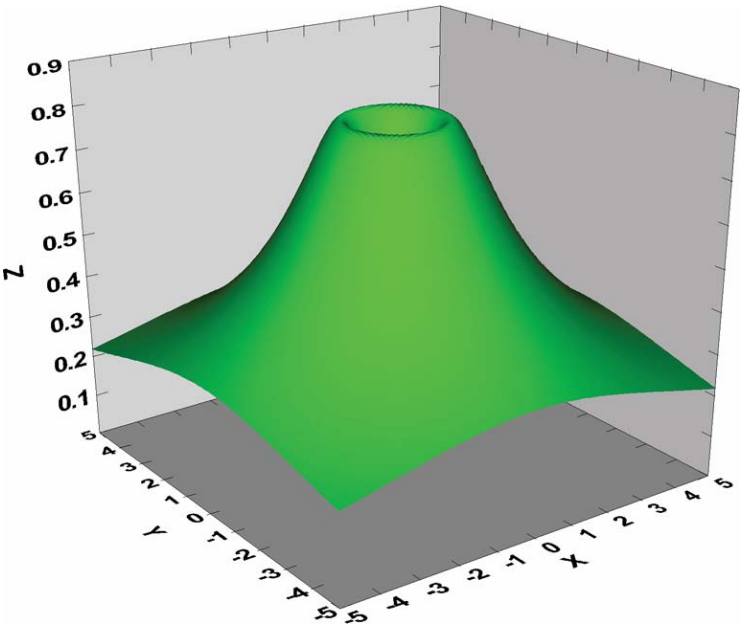


Figure 3.7 *Automated function plot*

It is easy to miss the point in the example because of the existence of symbolic and numeric mathematics systems which plot functions by generating a regular or irregular mesh of data points within a specified range and domain. The searchable dataset in the system described here, by contrast, is theoretically infinite. It varies depending on the function being examined. For some functions, the system spends most of its time examining values densely in a local neighborhood and for others, it ranges far-and-wide looking for global behavior. Thus, abstract data are not static values generated by a theoretical function and residing in memory or a file. They are instead produced by a mathematical system or simulation algorithm that is capable of changing its search space dynamically. Abstract datasets are not massive. They are infinite.

3.4 Metadata

Because our thoughts and observations are structured by the situation in which we make them, data always include **metadata**. These are the facts about the setting and circumstances under which we made our observations. Metadata are data about data. Our countries data, for example, include information about the statistical reporting methods for each country, the reliability and biases in the reports, types of missing data, the collection of these reports through an agency of the United Nations, the encoding into a computer file, the distribution of the file via electronic and printed methods, as well as numerous other details. Metadata affect the way we select data for and interpret data in our graphics. They can influence our display formally through imputation models (Rubin, 1987) or informally through written annotations attached to the graphic.

Several fields of study make use of metadata in different ways. Database metadata define descriptions of fields, columns, domains, and sometimes constraints. A primary advantage of this technique is that applications can discover the structure of information contained in a database at run time. Web applications encode metadata in hidden tags used within the page to provide keywords or other summary information about what a web page contains. This information is specifically hunted by search engines for classifying web pages. In geographic systems, information regarding area, perimeter, topography, or demography is considered metadata and stored separately from the polygons and images. In statistical graphics, the ancillary statistics calculated to produce a graphic can be considered metadata relevant to the graphic. For example, a box plot produces inner and outer outliers, hinge points, fence points and a median. All of this information can be structured in a way that parallels the geometry of the resulting graphic as metadata.

Combining contextual metadata with the graphics grammar can enable us to generate graphics automatically from user queries. Assuming table- and column-level metadata are available to describe logical, metric, and statistical aspects of variables, we can generate grammar rules to produce one or more graphics relevant to the query.

3.5 Data Mining

Recent developments in data warehousing, data mining, and knowledge discovery in databases (KDD) have blurred the distinctions between graphics, statistics, and data management. Indeed, some proponents of KDD have claimed that it will replace graphical and statistical analysis packages in the future. These claims, however extreme, must be examined in terms of models rather than commercial products. It is easy to mimic functionality and wrap old designs in new interfaces in commercial software, so product comparisons are generally not good ways to evaluate such claims.

It is difficult to discuss this field without invoking its acronyms. A popular one is OLAP, or on-line analytical processing. OLAP is intended to describe a computing environment that provides multiple views of data in real time (or, at least, responsive-to-my-needs time). These views are composites of text and graphics that can be manipulated by the user to develop alternative views. There are two main approaches to implementing an OLAP system: Multidimensional OLAP (MOLAP), and Relational OLAP (ROLAP). We will discuss these first and then conclude with a brief introduction to visual query of databases.

3.5.1 MOLAP

MOLAP depends on a basic object called a **data cube**. This object is a multidimensional array. Each dimension of this array is a set of sets representing a content domain such as time or geography. The dimensions are scaled categorically (region of country, state, quarter of year, week of quarter, etc.) so that the whole object defines a multidimensional array of cells. The cells of this array contain aggregated measures of variables such as income or gender. The data cube *is* the data for a MOLAP. In fact, this cube is usually prepared before the user begins to explore patterns in the data, sometimes in overnight processing runs on massive relational databases.

Exploring relations in the cube involves several operations. The popular names for these are **drill-down**, **drill-up,** and **drill-through**. Drill-down involves splitting an aggregation into subsets. For example, data for regions on a geographic dimension may be split into data for states. Drill-up is the reverse of drill-down; the user aggregates subsets on a dimension. Drill-through involves subsets of crossings of sets. The user might investigate statistics within a particular state and time period, for example.

These operations work well with statistics like counts, means, proportions, and standard deviations. Simple aggregations over subclasses can be computed by operating on sums, sums of squares, and other terms that are combined in linear functions to produce basic summary statistics. They do not work properly with statistics like the median, mode, and percentiles because the aggregate of these statistics is not the statistic of their aggregates. The median of medians is not the median of the aggregate, for example. A telling indicator of this fact is the lack of medians and modes from the menus of MOLAP front-ends.

It is unnecessary to provide examples to illustrate the importance of these alternative summary statistics. Every statistics student is introduced to the differences between the mean and median within the first few weeks of the first course in high school or college. One can find few governmental or commercial summaries based on mean income, mean housing prices, or mean cholesterol in the population. The median is used on these variables because of the extreme skewness in their distributions or because of the prevalence of outli-

ers. And for marketing data, the mean cannot substitute for the modal choice over alternatives in a study of preferences.

Few of the graphics in this book and in other important applications can be computed from a data cube. There are several reasons for this situation. First, many of the statistics presented in Chapter 7 require raw data for their computation. These statistics give graphs the ability to represent sampling error within subclassifications. Second, spatial and time series statistics require raw data because the distribution of errors across space and time is not independent of space and time. Third, aggregation often involves weighting of subclassifications by measures other than sample size. This has emerged as a focal problem in the governmental deliberations over census taking versus sampling. Glymour *et al*. (1996) discuss other factors that disqualify the data cube from serious consideration as an informative database exploration model.

A recent exception to this evaluation is a model presented in Sarawagi *et al*. (1998). While the system they describe is not designed for producing general graphics, it does incorporate a statistical model at the cube-building phase that allows later exploration of the cube to be guided by robust statistical procedures. Outlier detection based on these models is used to drive the coloring of surprising cells in the cube so that ordinary users are drawn to investigate anomalies further. Their model could be used as one approach toward adapting cubes to more general graphical displays.

3.5.2 ROLAP

ROLAP places an interactive data-view at the front-end or client-side of a relational database. Some ROLAPs are based on a data cube model and can be disqualified from serious consideration for reasons similar to the ones we have given in the previous section. A more sophisticated ROLAP model has emerged recently, however. It is possible, through several technologies, to give statistical algorithms access to raw data through the relational model in real time. This is an approach we advocate in this book.

One method is to use extended Structured Query Language (SQL) to create relations that can be presented to other language clients (Java, *C++*, etc.) for use as variables. This approach requires either storage of the data on the client side (the preferable arrangement) or slow, case-by-case processing across a communications link (network or Internet). Nevertheless, this method provides access to the raw data and, if an extension to SQL is used, allows the kind of data reshapings that are featured in this and the next chapter.

Another method has become available with the advent of platform-independent languages like Java and the encapsulation offered by software **components**. These components permit remote invocation of methods that ordinarily would be an internal part of an executing program. In this way, statistical modules can be sent to servers or remote sites to process data locally and return summaries and other statistical objects. This approach is more

promising than the rigid aggregations offered by structures such as data cubes. In the most elegant form of this architecture, applications can request remote components to provide information about their data-handling methods and take suitable action depending on the returned information. In this form, component architecture can achieve the real promise of distributed computing: design and execution that are independent of site, operating system, or language.

3.5.3 Visual Query of Databases

In a MOLAP or ROLAP, graphical displays are driven by the relational structure of a database. Queries through a language such as SQL yield tables that can be graphed. Researchers are beginning to develop approaches that reverse this dependency: they are designing search methods that are driven by visual query languages rather than relational languages. The original impetus for these methods came from Geographic Information Systems (GIS), where spatial relations rather than variable relations necessarily govern the search structure.

Papantonakis and King (1995) devised a graphical query language called GQL, which functions like SQL but operates on graphical objects rather than relational variables. Derthick, Kolojejchick, and Roth (1997) have extended this model to allow dynamic exploration rather than static queries. In their system, graphics are linked views into a database; manipulating graphical objects immediately updates data objects and updating data objects immediately updates graphical objects. Many of the graphical methods for accomplishing these actions are derived from the work of the original Bell Labs statistics group (see Cleveland and McGill, 1988).

Chi *et al.* (1997) have used a spreadsheet metaphor to organize graphical exploration. Their implementation is a prototype, but it can integrate multiple data sources in a single display. This allows exploration through dynamically linked views. This work resembles the trellis graphic of Becker and Cleveland (1996) but is designed to be a controller as well as a display.

The graphics algebra presented in this book is related to this research but comes from the opposite direction — derived from the structure of graphics rather than the structure of data. We can expect to see convergence in database query methods and graphic displays in coming years. Graphics will evolve beyond passive displays and will begin to play a role in the organization of data. This trend is driven by the need to get beyond static query to real-time interaction.

3.6 Sequel

Now that we have data assembled into a reference system, we need to link data to theoretical constructs. The next chapter covers variables and variable sets (varsets). These are the entities that graphs describe.