Chapter 3

Defining Data

Data have a story to tell. ViSta translates that story into the languages of visualization and analysis so that we can understand it. To this end, we introduce the most fundamental aspects of data, including: ViSta's data types; the ways ViSta represents data; the methods ViSta provides for defining data; ViSta's data function; and the way ViSta stores data in your computer's file system.

3.1 Data Types

ViSta recognizes three fundamental types of data: multivariate data, table data, and matrix data. Multivariate data have observations about many variables; Table data have observations on a single variable that are classified by several other variables; Matrix data have data-elements that are organized into square matrices.

Multivariate data are data for which there are observations about many variables. The data are organized so that the columns of the data correspond to variables -- there is one column for each variable -- and the rows of the data correspond to observations -- there is one row for each observation of values for the several variables. Each observation has a label and each variable has a name and a type. The names and labels may be anything the user wishes, and are used for identification. The types may be either numeric, ordinal or category, and are used to determine what kinds of analyses are appropriate.

Table data are data which have observations on a single variable that are classified by several other variables. These are called table data because the classification variables can be thought of as the ways of a table, with the classes for a given classification being the levels of the way. In ViSta, table data must be univariate, but do not need to be balanced. There is no restriction on the number of ways (classification variables) nor on the number of levels (classes).

Matrix data are data for which the rows and columns represent the same objects. The most familiar example is a correlation matrix -- its rows and columns represent the same set of objects, whatever they are. Note that this is different than multivariate data where the rows represent observations and the columns variables. In ViSta, matrix data may consist of one or more matrices of data, all having the same number of rows and columns (i.e., each must be square). Each matrix may be symmetric or asymmetric.

3.2 Representing Data

Because of the central roll played by data, they are represented in four different ways in ViSta. These ways include *data objects*, *data icons*, *datasheets*, and *datafiles*. We define each of these representations in this section.

The most basic representation of data is as a *data object*. A data object is an abstract object which "contains" data and which lets us "do things" with the data. There are three kinds of data objects – multivariate data objects, table data objects and matrix data objects – one for each kind of data. When we define data we are really defining a data object. Once a data object is defined then the data it contains may be analyzed and visualized. It can also be modified, transformed, saved, deleted, etc. It can be displayed in a datasheet and edited with a data editor. In short, all the things

we might want to do with the data can be done, *once they are contained in a data object*. As you can see, the data object is the most basic representation of data.

Data objects are abstract objects. They aren't real objects that can be seen. However, they are so important that they are visually represented on the workmap by *data icons* (as we saw in Chapter 2). There are three different data icons, one for each type of data object.

ViSta can have any number of data objects. At any one time only one of the data objects is the focus of the data analysis and visualization process. These data are called the *"current data"*, since they are the current focus of the system. The current data are signified by the highlighted data icon and the check-marked item in the **Data** menu.

The data in a data object can be displayed in a datasheet. The datasheet has an editor for entering and editing data. Note that the data in the datasheet are a completely separate *copy* of the data in the data object. When you use the editor to make changes in the datasheet, you have not changed the data in the data object. When you close the datasheet window you have the option of discarding the changes, of updating the data in the data object or of creating a new data object. So, if you make errors or change your mind, you can close the datasheet without any effect on the data in the data object. We discuss datasheets in sections 3.3.1 and 3.3.2.

In addition to data objects, there are *datafiles*. Datafiles are discussed in section 3.5. A data object exists within ViSta, whereas a datafile exists outside of ViSta in your computer's file structure. A datafile has a particular structure, including the data itself, that ViSta uses to create a data object containing the data. When this is done the data in the data object can be analyzed and visualized by Vista.

3.3 Methods for Creating Data

Now that we have shown you how data are represented in ViSta, we can proceed to show you how you can use ViSta to create data. There are six ways for creating data (actually, for creating data objects). These ways include:

1. entering new data into a datasheet;
2. editing data that are already in a datasheet;
3. creating data from data objects or model objects;
4. importing data from text files;
5. generating simulated data; and
6. using the data function.

We discuss each of the ways in this section.

3.3.1 Entering New Data into a Datasheet

As shown in Chapter 2, data may be displayed as a datasheet. In addition to simply displaying data, the datasheet can also be used to create new data and to edit existing data. In this section we discuss creating new data by using the datasheet. We discuss editing data in the next section.

The steps for entering new data into a datasheet are:

1. Use the **File** menu's **New Data** item to create a new (empty) datasheet. In the dialog that appears you should name the new data and you must select whether you wish to create multivariate, matrix, or classification (table) data. After you use the dialog box you will see a new datasheet. It will have the minimum number of observations, variables and matrices allowed for the type of data you are creating. The datasheet has nil data values (for missing data), default variable names and types, and default observation labels. In addition, you have a new data object represented by a new data icon on the workmap and a new menu item at the end of the **Data** menu.
2. If the data are multivariate or classification data, use the **New Obs** and **New Var** datasheet buttons (or the **DataSheet** menu's **New Observations** and **New Variables** items) to give the datasheet the desired number of observations and variables. If the data are matrix data, use the **New Var** and **New Matrix** datasheet buttons (or the corresponding menu items) to give the datasheet the desired number of matrices with the desired number of variables (rows and columns). After you do this you have a datasheet filled with `nil` (missing) values.
3. At this point you have a new datasheet which contains data that are entirely missing. By following the techniques outlined in the next section you can use the datasheet editor to enter data values, to rename variables and observations, and to change variable types.
4. After you have entered your new data values, use the datasheet's close box to close the datasheet window. The new data object that was created in step 1, above, now contains your data. If the new data object contains no missing values[1] it can be used for analysis and visualization.
5. Finally, it is wise to use the **Data** menu's **Save Data** item to save the new data in a datafile. Otherwise, when you **Quit** ViSta your new data will be lost.

3.3.2 Editing Data in a Datasheet

As we saw in Chapter 2, data may be displayed as a datasheet. In addition, the datasheet has an editor that lets you enter new data and edit existing data. It is

1. A data object may contain missing values, but ViSta cannot currently process such data.

important to understand that the datasheet and the datasheet editor are two separate features. This means that when data are displayed in a datasheet the data may or may not be editable, depending on whether the editor is active. When the data editor is not active we say we say that the data are *browsable* - we can "browse through" the data but we can't edit it. When the editor is active the data are *editable* - we can edit the data as well as browse through it. Note that when data are editable nothing else can be done with the data until it's datasheet is closed. On the other hand, browsable data place no restrictions on what you can do.

The distinction between editable and browsable data interacts with other aspects of ViSta. When you create a new data object by opening a datafile with **Open Data** the data are displayed for browsing. On the other hand, when you create a new data object with **New Data** the datasheet is editable. If you already have a data object, you can open it for browsing by double-clicking the data icon or by using the **Browse Data** item in the data menu (or in the data icon's popup menu). Alternatively, a data object can be opened for editing by using the data (or popup) menu's **Edit Data** item. This item can also change browsable data to editable data.

Note the following very important fact about editable data: When you are editing the data shown in an editable datasheet, *you are not changing the data in the data object*. The data in the datasheet is a completely separate copy of the data in the data object. When you close the datasheet you have the option of updating the data in the data object. So, if you make errors or change your mind, you can close the datasheet without having any effect on the data in the data object.

3.3.2.1 The DataSheet

The datasheet displays data values and other information. As is shown in Figure 1, the datasheet is divided into four major regions by heavy lines. Editable datasheets also have two "buttons", one at the bottom-left and one on the upper-right. The four regions are:

1. The main "body" of the datasheet is in the lower-right region. It shows the data values. Each column corresponds to a variable and each row to an observation. If data values are too wide to fit into a cell they are replaced by asterisks, as in Figure 4.

5 Vars / 6 Obs	Country	MPG	Weight	Horsepow	Cylinder	New Var
	Category	Numeric	Numeric	Numeric	Ordinal	
Buick Estate Wagon	U.S.	16.90	4.36	155.00	8.00	
Ford Country Squire Wagon	U.S.	15.50	4.05	142.00	8.00	
Chevy Malibu Wagon	U.S.	19.20	3.60	125.00	8.00	
Chrysler LeBaron Wagon	U.S.	18.50	3.94	150.00	8.00	
Chevette	U.S.	30.00	2.15	68.00	4.00	
Toyota Corona	Japan	27.50	2.56	95.00	4.00	
New Obs						

Figure 1: A Datasheet

2. The lower-left region is the “label” region: It has a single column that shows the observation labels.
3. The upper-right region is the “name and type” region: It has two rows showing the name and type of each variable.
4. Finally, the upper-left region is the “information” region: It shows the number of observations and variables in the data.

For editable data the datasheet also has two “buttons”.

5. The “new observation/matrix” button is at the bottom of the labels column. The button says **New Obs** (**New Matrix** for matrix data). Clicking here gives you a new observation (matrix).
6. The “new variable” button is at the right of the variable name row. Clicking on it gives you a new variable.

3.3.2.2 The DataSheet Menu

When a datasheet is shown a **DataSheet** menu is added to the menubar. The menu appears in Figure 2. For browsable data this menu has two active items that let you adjust the **Width of Columns** and the **Number of Decimals** shown for numeric variables. When the data are editable the menu has three additional active items. Two of these permit adding **New Observations** and **New Variables** (or **New Rows and Columns** and **New Matrices** for matrix data). For multivariate data the last menu item lets you **Switch Label Variable** so that you can redefine which variable is used for labeling observations. For matrix data this item lets you **Change Matrix Names**.

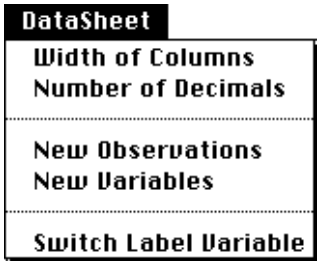

Figure 2:
The DataSheet Menu for Editable Data

3.3.2.3 The Datasheet Editor

For editable data you can use the datasheet editor. This editor allows you to select a cell of the data and to change it. You do this by simply clicking on a cell and typing.

You can move to nearby cells by using the cursor-movement (arrow) keys[2]. The tab key takes you to the beginning of the next row. The Return (Enter) key takes you to the next cell to the right. You must use your mouse to enter either the labels column or the variable names and types rows. You cannot use keys to move into these areas,

2. Currently, the Microsoft Windows version of XLisp does not permit using the arrow keys. As a temporary measure the arrow key functions are mapped onto the keys at the right end of the keyboard’s main rows: The return key acts as right-arrow; the ‘ (quote) key as down-arrow; the ; (semi-colon) key as left-arrow; and the [(left square bracket) key as up-arrow.

nor to move out of the labels column. Finally, the Macintosh's Home and End keys take you to the first and last cell of the datasheet.

When you click on a cell or move to it with another key the cell highlights, indicating it is ready for typing. When you begin to type, the cell is outlined (indicating it is accepting new information) and the old information in the cell is replaced with the typed information. While typing, you can use the Delete (Backspace) key to delete information.

You may type anything you wish in any cell, whether it be a body, label, name or type cell. No error or consistency checking is done until you try to close the datasheet window. At that time the variable types are checked and compared to each variable's contents. You will be able to close the data only if:

1. The variable types are either `Numeric`, `Ordinal`, or `Category`; and
2. `Numeric` and `Ordinal` variables have entirely numeric information.

If these conditions are not met you will see an error message and you must fix the variable types (capitalization is ignored) and/or either make all `Numeric` or `Ordinal` values numeric or change the offending variable types to `Category`.

3.3.3 Creating (subsets of) Data from Existing Data

Data (objects) may be created from existing data objects by using the **Data** menu's **Create Data** item.[3] You do this by simply choosing the menu item and then using the dialog box that appears to name the new data object. This will make an exact copy of your data.

Note that this menu item lets you subset your data. To do this, first use the **Data** menu's **List Observations**[4] and/or **List Variables** menu items to show the windows that list all of the variable names and/or observation labels. Then you select the specific variables and/or observations that you want included in the data object you will be creating. Then, when you use the **Create Data** menu item, the new data object will only contain the chosen subset of data.

Finally, if your data have missing values you may use the **Create Data** menu item to create a new data object that has no missing values. Specifically, if your data contain missing values you will be given the option to delete all observation-rows that have missing values.

---

3. Data may also be created by models with the **Model** menu's **Create Data** item, as is explained in the chapters about models.

4. **List Matrices** for matrix data, and **List Cells** for table data.

3.3.4 Importing Data From Text Files

```
Honda Accord          5 5 5
Honda Civic           5 5 4
Lincoln Continental   2 4 5
Plymouth Gran_Fury    2 1 3
Plymouth Horizon      4 3 4
Plymouth Volare       2 1 5
Pontiac Firebird      1 1 5
```

Figure 3: Importable Text Data

Data may be imported from text files (also called "ascii" or "flat ascii" files) by using the **Data** menu's **Import Data** menu item[5]. An example of such a file is shown in Figure 3. The file to be imported must contain only data. It cannot contain variable names or types. The default names and types that are generated can be changed with the datasheet editor. Observation labels may be imported as a variable and then changed to labels with the **DataSheet** menu's **Switch Label Variable** item.

The first record in the file must have one data value for each variable. Data values must be separated by white space. The number of data values in the first record determines the number of variables in the data object. In Figure 3 there are 5 data values in the first record (as determined by the white space), so there will be 5 variables in the imported data object, as can be seen from the information region of the object's datasheet shown in Figure 4 (asterisks are used when information is to large to fit in a cell). Succeeding records must have the same number of data values (note, in Figure 3, that `Gran_Fury` has an underline). A variable that has one or more values that contain one or more non-numeric values is treated as a `Category` variable, otherwise variables are numeric. Non-numeric values may be symbols (as in Figure 3, where the characters are not inside quotes) or strings (where characters are in quotes). The use of quotes permits white space inside values. For these data the first two variables will be `Category`. Missing values must be coded as nil (however, such data cannot be analyzed without removing the rows that have missing values, which may be done with the **Create Data** menu item).

5 Vars / 7 Obs	Var0	Var1	Var2	Var3	Var4
	Category	Category	Numeric	Numeric	Numeric
Obs0	HONDA	ACCORD	5.00	5.00	5.00
Obs1	HONDA	CIVIC	5.00	5.00	4.00
Obs2	LINCOLN	*****	2.00	4.00	5.00
Obs3	*****	*****	2.00	1.00	3.00
Obs4	*****	HORIZON	4.00	3.00	4.00
Obs5	*****	VOLARE	2.00	1.00	5.00
Obs6	PONTIAC	FIREBIRD	1.00	1.00	5.00

Figure 4: Datasheet of Imported Data

5. Currently, only multivariate data may be imported, although table data may be imported as multivariate data and then converted to table data if the first variable is `numeric` and the remaining variables are `category`. Matrix data may not be imported as of this writing.

3.3.5 Generating Simulated Data

This feature is not yet supported and will be documented at a later time.

3.4 The Data Function

As was pointed on in Chapter 1, one of the philosophical foundations of ViSta is that the user never has to see the underlying language, but that the language is always there for the sophisticated user. Thus, the several ways of creating data that we discussed in section 3.3 all create data by using the `data` function, a function in the underlying ViDAL language. This function actually creates the data object.

If you wish, you can type the function yourself, rather than have ViSta generate it for you. You can do this by typing directly in the listener window, or you can use a text editor to prepare a file that contains the function. Then, when you evaluate the function, it will create a data object. It can create any of the three types of data objects – multivariate, matrix or table data objects. The specifics of the `data` function for each type of data object are explained in the each of the next three subsections.

```
(data "HealthClub"
:title
"Data from a Health Club"
:variables
'("Weight" "Waist" "Situps")
:data '(
191 36 162
189 37 110
193 38 101
162 35 105
189 35 155
182 36 101
211 38 101
167 34 125
176 31 200
154 33 251
169 34 120
166 33 210
154 34 215
193 36  70
202 37 210
176 37  60
157 32 230
156 33 225
138 33 110))
```

Figure 5: Data function for multivariate data

3.4.1 Multivariate Data

An example of a `data` function for creating multivariate data is shown in Figure 5. The `data` function has one initial required argument and two required keyword arguments. The initial required argument is a string that is used to name the data. In the Figure, this is "HealthClub". Any characters may be used in this string, including spaces. In addition to the name, you must also use the `:variables` and `:data` keywords, following each with a list of elements. The `:variables` keyword specifies the names of the variables and (indirectly) the number of variables. It is followed by a list of character strings which are the variable names. The `:data` keyword specifies the list of data values. If n is the number of variables, the first n elements of the list are the values for the first observation (row) of the data, the next n elements are the values for the second observation, etc. The total number of elements of the data list must be an exact integer multiple of the number of variables.

There are several optional keywords which may be used with the `data` function, one of which, the `:title` keyword, is shown in the example. This keyword is followed by a character string to specify the data-object's title, which is used in various windows. When not specified, the data title is `Untitled Data Object`. The `:types` keyword, which is followed by a list of character strings, one for each variable, specifies the type of each variable. The type may be "category", "ordinal", or "numeric". If this keyword is not used, all variables are assumed to be numeric. The `:labels` keyword, which is followed by a list of character strings, specifies the label for each observation (row of the data). If not used, the observation labels are `Obs0`, `Obs1`, etc.

3.4.1.1 Matrix Data

An example of a `data` function for creating a matrix data object is shown in Figure 6. A matrix data object is a data object whose data consist of one or more matrices of data. Matrix data objects are defined in exactly the same way as Multivariate data objects, except that the `:matrices` keyword must be used. This keyword, which is followed by a list of character strings, specifies that the data are matrix data, and specifies the names (and, indirectly, the number) of the matrices. The `:matrices` keyword is required for matrix data, and must not be used for other types of data. The `:shapes` keyword, which is followed by a list of character strings, one for each matrix, specifies the shape of each matrix. The shapes may be "symmetric", or "asymmetric". If shapes are not specified, then all matrices are assumed to be symmetric. In the example shown in Figure 6, both the `:shapes` and the `:types` keywords are not needed, because they both specify characteristics of the data which are assumed by default. If the data consists of several matrices, then the `:data` list consists of all the values for the first matrix, followed by all the values for the second matrix, etc.

```
(data        "FlyMiles"
:title       "Flying Mileages between 10 Cities"
:variables '("atlanta" "chicago" "denver" "houston"
             "los angeles" "miami" "new york"
             "san francisco" "seattle" "wash.d.c.")
:labels    '("atlanta" "chicago" "denver" "houston"
             "los angeles" "miami" "new york"
             "san francisco" "seattle" "wash.d.c.")
:matrices  '("Mileages")
:data      '(
   0  587 1212  701 1936  604  748 2139 2182  543
 587    0  920  940 1745 1188  713 1858 1737  597
1212  920    0  879  831 1726 1631  949 1021 1494
 701  940  879    0 1374  968 1420 1645 1831 1220
1936 1745  831 1374    0 2339 2451  347  959 2300
 604 1188 1726  968 2339    0 1092 2594 2734  923
 748  713 1631 1420 2451 1092    0 2571 2408  205
2139 1858  949 1645  347 2594 2571    0  678 2442
2182 1737 1021 1831  959 2734 2408  678    0 2329
 543  597 1494 1220 2300  923  205 2442 2329    0
))
```

Figure 6: Data function to define matrix data.

3.4.1.2 Table Data

Table data objects are defined in the same way as previous data objects except that only one variable can be specified by the `:variables` keyword (i.e., the data must be univariate), the `:data` keyword must be followed by a list of lists (rather than a list of values), and the `:classes` and `:ways` keywords must be used.

An example is shown in Figure 7. The :ways keyword is followed by a list of character strings that are used to name the ways of the data. In the example the ways are "Lab" and "Sample". The arguments of the :classes keyword specify the names and the number of levels of each way of the table. For one-way data the :classes keyword is followed by a list of character strings. For multi-way data it is followed by a list of lists of character strings. The number of lists must correspond to the number of ways. In the two-way example shown in Figure 7, the :classes keyword is followed by a list of two lists. Since the first list has six elements and the second list has two, these data form a 6x2 table. Finally, the :data keyword must be followed by a list of lists of values. Each sub-list is a cell of the design. The list of lists allows varying numbers of data-elements per cell. In the example, there are 4 elements in each list. Thus, these data are balanced (same number of observations in each cell) 6x2 two-way data that are replicated 4 times.

```
(data "Eggs2"
      :variables '("FatContent")
      :ways '("Lab" "Sample")
      :classes '(("A Lab" "B Lab" "C Lab"
                          "D Lab" "E Lab" "F Lab")
                 ("1st Sample" "2nd Sample"))
      :data '(

;         Sample 1                  Sample 2
   (0.62 0.55 0.80 0.68)     (0.34 0.24 0.76 0.65);  Lab A
   (0.30 0.40 0.39 0.40)     (0.33 0.43 0.29 0.18);  Lab B
   (0.46 0.38 0.37 0.42)     (0.27 0.37 0.45 0.54);  Lab C
   (0.18 0.47 0.40 0.37)     (0.53 0.32 0.31 0.43);  Lab D
   (0.35 0.39 0.42 0.36)     (0.37 0.33 0.20 0.41);  Lab E
   (0.37 0.43 0.18 0.20)     (0.28 0.36 0.26 0.06);  Lab F
))
```

Figure 7: Data function to define two-way table data.

A second example appears in Figure 8. In this example the :classes list has only one list of four classes, hence the data are one way, and the way has four levels. Note that the :ways keyword specifies one way. The data consist of four sub-lists of different lengths. Since each sublist specifies all of the multiple observations for each cell of the table, these data are unbalanced.

```
(data        "Singers"
:title       "Singer's Heights"
:variables '("Height")
:types     '("Numeric")
:ways      '("Part")
:classes   '(("Sopranos" "Altos" "Tenors" "Basses"))
:data      '(
(64 62 66 65 60 61 65 66 65 63 67 65 62 65 68 65 63 65
 62 65 66 62 65 63 65 66 65 62 65 66 65 61 65 66 65 62)
(65 62 68 67 67 63 67 66 63 72 62 61 66 64 60 61 66 66
 66 62 70 65 64 63 65 69 61 66 65 61 63 64 67 66 68)
(69 72 71 66 76 74 71 66 68 67 70 65 72 70 68 73 66 68
 67 64)
(72 70 72 69 73 71 72 68 68 71 66 68 71 73 73 70 68 70
 75 68 71 70 74 70 75 75 69 72 71 70 71 68 70 75 72 66
 72 70 69)))
```

Figure 8: Data function for one-way unbalanced table data

3.5 Datafiles

Everything that we have discussed in this chapter up to this point has concerned the way that data are treated within ViSta. The final topic, that of datafiles, concerns how data are stored outside of ViSta in your computer's file system, and how they are placed there and retrieved for later use.

This is very straight-forward. To create a datafile you simply use the **Data** menu's **Save Data** item. A dialog box will appear that lets you name the file and locate where you wish to save it. The file's name must end with the `.lsp` extension, which will be added automatically if you don't provide it. Type in a proper name (details determined by your computer system), find the place you wish to save the data, and click OK. A datafile has now been created and your data have been saved in it.

The data may be loaded in at a later time by using the **File** menu's **Open Data** or **Load Data** menu items. These two items do the same thing, except that **Open Data** does a **Browse Data** after the data are loaded.

You can, if you wish, look at the file with a program editor. It is an ascii file that can be transported between operating systems. When you look at the file you will discover it contains a `data` function for defining your data.

Finally, you can prepare a file with a program editor and save it with the `.lsp` extension. If this file's outer-most function is the `data` function, and if the function is used properly, then the file is a proper datafile which may be loaded into ViSta. (The `data` function may have other functions nested inside it).