

# *Transformations*

Forrest W. Young & Pedro Valero

THE L.L. THURSTONE  
PSYCHOMETRIC LABORATORY  
UNIVERSITY OF NORTH CAROLINA  
CB 3270 DAVIE HALL, CHAPEL HILL N.C., USA 27599-3270

VISUAL STATISTICS PROJECT  
[WWW.VISUALSTATS.ORG](http://WWW.VISUALSTATS.ORG)  
REPORT 2000-3  
JANUARY 2000



# Transformations

Forrest W. Young & Pedro Valero

## **Abstract**

This report presents the set of functions for transforming variables that are available to the ViSta user. Transformations are capable of processing the data so they are more amenable to the assumptions of common statistical analysis. The data can include missing values and the transformed data will deal with them appropriately.

These transformations generally work directly, producing a new data object that includes the new transformed variables. There are numerous pre-defined transformations, as well as the capability for user-defined transformations.

In addition, some of the transformations are visual, providing visualizations that allow the user to dynamically choose the parameters of the underlying transformation function. These visualizations include a scatterplot-matrix of the transformed variables, a histogram, a line plot indicating the relation between original and transformed variables, and a slider that controls the transformation.



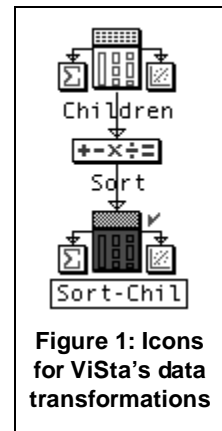
## 1 Introduction

Very often the researcher needs to transform the collected data before proceeding with its analysis. For example, many common statistical procedures, such as regression or ANOVA, assume that the univariate distribution of continuous variables follows approximately the normal distribution, but it is often the case that collected data have asymmetric, non-normal variables. This can be a severe limitation for the analyst, because even though the statistical packages will still produce a result, it will be misleading since the tests and confidence intervals are based on the assumptions not satisfied by the data.

Unfortunately, statistical procedures based on these assumption are well known for their simplicity and elegance and, even though there are alternatives available, they are the methods of choice in a large range of scenarios. Transformations of the original variables can be the solution to this problem because they can make the data more symmetric. Furthermore, these transformations are very useful for exploratory data analysis because they can reveal hidden relationships in the data.

The need for transformations also arises when the researcher must take several steps before arriving at a dataset amenable for applying the desired statistical analysis. ViSta has transformations that are directly accessible from menus which cover the needs of a large number of users. It also has an open-ended transformation which can be defined as desired by the user.

Most of the commands in the menu transformations work in the following way. With a data object already selected, the transformation produces a new data object containing the transformed data. The data objects are represented on the WorkMap by data icons, which are connected together through another icon symbolizing the transformation. The series of three icons is shown in Figure 1. Sometimes, the user is presented with a dialog that requests further arguments needed for carrying out the transformation.



ViSta's transformations are grouped into three categories.

1. **Pre-Defined Transformations.** These are the familiar transformations commonly available in all statistical systems. They require no programming and are not visual.
2. **User-Defined Transformations:** These transformations are defined by programs written by the user. The programming environment for defining transformations is discussed in this report. The programming language is discussed elsewhere.
3. **Visual Transformations:** Two transformations deserve special mention because they have visualizations which help the user select the best member of a family of transformations.

We discuss the transformations in the remainder of this report. Examples are provided for some of the transformations to illustrate their statistical application.

## 2 Pre-Defined Transformations

The pre-defined transformations include sorting, ranking, normalizing and rounding data; computing normal scores, absolute values, reciprocals, exponentials, logarithms and trigonometric functions of data; and computing matrices of correlations, covariances and distances, as well as transposing and orthogonalizing matrices. We review these in this section.

### 2.1 Sort-Permute.

This command sorts a data object according to the values of a variable. It permutes the rows according to this sorting, so the identity of each case is not lost. The Sort and Permute menu item's dialog box (Figure 2) supports ascending (small to large, or alphabetical) or descending sorts. The observation labels, as well as variables, may be sorted.

Sorting is useful in a variety of situations. For example, checking for data entry errors can be done by sorting and looking for values which don't make sense. Figure 3a shows a small simulated example with two variables: gender of subjects (Gender) and number of pregnancies (Children). Figure 3b shows the sorted data and easily reveals an error.

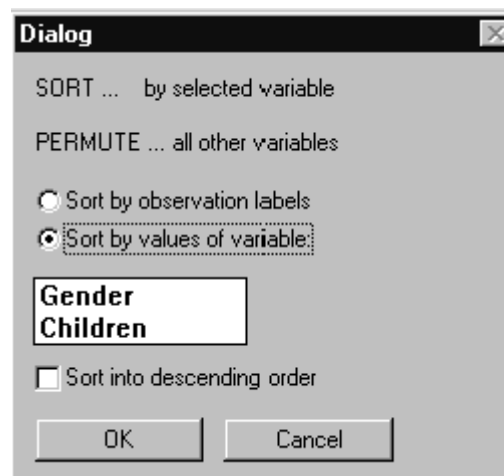


Figure 2: Sort-Permute Dialog Box

### 2.2 Normal Scores

The Normal Scores transformation creates a new dataset in which the values of each numeric variable in the current dataset are converted into normal scores. Ordinal and category variables are not included in the new dataset. Normal scores are the normalized z-scores that would be obtained under the assumption that the variable is normally distributed, given the variable's mean and standard deviation.

2 Vars 10 Obs	Gender	Children
	Category	Numeric
Obs1	F	2.00
Obs2	F	0.00
Obs3	F	1.00
Obs4	M	0.00
Obs5	M	0.00
Obs6	F	1.00
Obs7	M	0.00
Obs8	F	2.00
Obs9	M	0.00
Obs10	M	1.00

Figure 3a: Data Before Sorting

2 Vars 10 Obs	Gender	Children
	Category	Numeric
Obs1	F	2.00
Obs2	F	0.00
Obs3	F	1.00
Obs6	F	1.00
Obs8	F	2.00
Obs4	M	0.00
Obs5	M	0.00
Obs7	M	0.00
Obs9	M	0.00
Obs10	M	1.00

Figure 3b: Data After Sorting

## 2.3 Ranks

Ranks are values between 1 and N (the number of observations). Ranks start at 1, for the smallest value of a variable, and proceed to N for the largest. Ties in the values of a variable are converted into means of appropriate ranks. The Ranks transformation creates a new dataset in which the values of each numeric or ordinal variable in the current dataset are converted into ranking numbers. Category variables are not included in the new dataset. Figure 4a shows the result of ranking the data in Figure 3b. Figure 4b shows the workmap, indicating the series of transformations that we have made.

1 Vars	Children
10 Obs	Numeric
Obs1	9.50
Obs2	3.00
Obs3	7.00
Obs6	7.00
Obs8	9.50
Obs4	3.00
Obs5	3.00
Obs7	3.00
Obs9	3.00
Obs10	7.00

Figure 4a: The Data in Figure 3b after using the rank transformation.

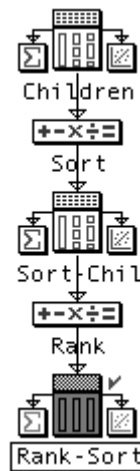


Figure 4b: The WorkMap

## 2.4 Absolute Value

The Absolute Value transformation creates a new dataset from the current dataset in which the values of the numeric variables in the new dataset are the absolute values of those in the current dataset. Ordinal and category variables are not placed into the new dataset.

## 2.5 Rounding

The Rounding transformation creates a new dataset from the current dataset in which the values of the numeric variables in the current dataset are rounded. Rounding means converting decimal numbers to the nearest integer. Ordinal and category variables are not placed into the new dataset.

Figure 5 shows the dialog box associated with this transformation. Four methods of rounding are available. 'Round' rounds to the nearest integer (so 0.4 rounds to 0 and -0.4 to 0). 'Ceiling' rounds upwards to the nearest integer (so 0.4 rounds to 1 and -0.4 rounds to 0). 'Floor' rounds downwards (so 0.4 rounds to 0 and -0.4 rounds to -1) and truncate removes the decimals (so both 0.4 and -0.4 truncate to 0).

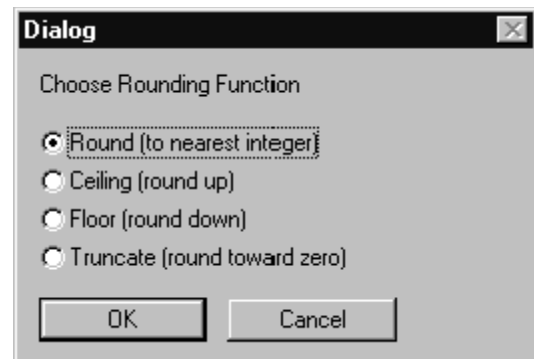


Figure 5: Dialog Box for Rounding

## 2.6 Reciprocal

The Reciprocal transformation creates a new dataset from the current dataset in which the values of the numeric variables in the new dataset are the reciprocals of those in the current dataset. The reciprocal of the number A is  $1/A$ . You should be aware of the values of your variables. If a variable has a value equal to zero, reciprocal will cause an error. Ordinal and category variables are not placed in the new dataset.

## 2.7 Exponential

The Exponential transformation creates a new dataset from the current dataset in which the values of the numeric variables in the new dataset are the exponentials of those in the current dataset. A choice of several specific exponential transformations is provided. It should be remembered that many of the transformations in this menu correspond to transformations available in the Box-Cox transformations, which provides a visualization that helps to choose the transformation that makes variables more amenable for statistical analysis.

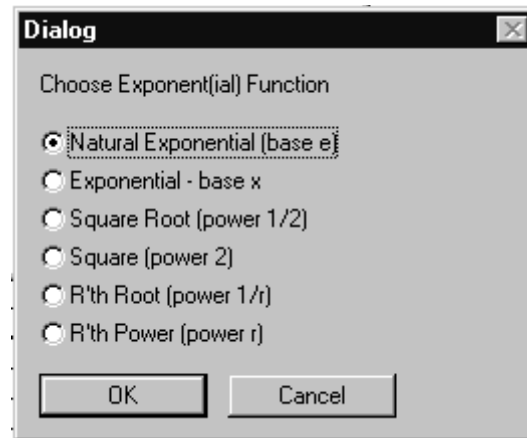


Figure 6: Dialog Box for Exponential Transformation

The dialog box associated with the command menu Exponential is shown in Figure 6. Ordinal and category variables are not placed into the new dataset. Some of the options in this dialog need an additional argument. These are: the Exponential - base x (you need to introduce the base; default is 10), the R'th Root and the R'th Power (both needs the value of r). The value for the additional argument is obtained in a subsequent dialog box.

## 2.8 Logarithm

The Logarithm transformation creates a new dataset from the current dataset in which the values of the numeric variables in the new dataset are the logarithms of those in the current dataset. A choice of natural (base e) or base X logarithms is provided, where the value of X is specified in an additional dialog box. Natural logarithms are also part of BoxCox transformations. You should be aware of values in your variables equal to zero because they will cause an error. Also, negative values should be avoided because they will produce values that are not acceptable for further analysis. Ordinal and category variables are not placed into the new dataset.

## 2.9 Trigonometric

The Trigonometric transformation creates a new dataset from the current dataset in which the values of the numeric variables in the new dataset are trigonometric transformations of those in the current dataset. A choice of the familiar trigonometric functions is provided in an additional dialog box. Ordinal and category variables are not placed into the new dataset.

## 2.10 Transpose

The Transpose Data transformation creates a new dataset from the current dataset in which the observations (rows) of the current dataset become the variables (columns) of the new one, and the variables (columns) of the current dataset become the observations (rows) of the new one.



## 2.11 Correlations and Covariances

The Correlations transformation creates a new dataset from the numeric variables in the current dataset. The values in the new dataset are the correlations or the covariances between the numeric variables in the current dataset. The diagonal either contains ones for correlations or variances of the variables for covariances. The new dataset has the same number of observations as it has variables. The labels of the observations are constructed from the variable names. Ordinal and category variables are not used in the calculations.

This transformation cannot be used with data that have missing values. If your data do have missing values you have the following alternatives to computing this transformation.

1. Use the **DATA** menu's **CREATE-CONVERT DATA...** item to remove rows with missing data values in the data. This means that each case with missing values will be deleted from the new data object and correlations or covariances can be computed using the Correlations or Covariances command. This method results in what is called 'Listwise' correlations or covariances in many statistical packages.
2. Use the **DATA** menu's **IMPUTE MISSING DATA...** item to compute a missing data model object. Then, the **MODEL** menu's **CREATE DATA** can be used to obtain Listwise, Pairwise or Maximum Likelihood correlations and covariances (explained in the chapter on Missing Data Imputation).

## 2.12 Distances

The Distances transformation creates a new dataset from the numeric variables in the current dataset. The values in the new dataset are the Euclidean distances between the rows of the numeric variables in the current dataset. The diagonal contains zeros. The new dataset has the same number of variables as it has observations. The labels of the variables are constructed from the observation names. Ordinal and category variables are not used in the calculations. No missing values are allowed in the data. See the preceding section for what to do if you do have missing values. If you wish to compute distances between variables, you should transpose your data first. See the description of the **TRANSPOSE** item given above. Figure 7 shows the top portion of the matrix of distances for the file cereals.lsp in the Data folder. The original data have 10 numeric variables and 76 observations. The distances have 76 rows and columns.

76 Obs	100% Bran	100% Natu	All-Bran	All-Bran	Almond De	Apple Cin	Biotic 4	Bran
Distances: 100% Bran	0.00	193.57	336.04	55.34	292.63	219.83	206.49	1
Distances: 100% Natural	193.57	0.00	312.17	243.74	251.29	179.45	200.21	1
Distances: All-Bran	336.04	312.17	0.00	122.28	329.19	265.70	235.84	2
Distances: All-Bran with Extra Fiber	55.34	243.74	122.28	0.00	342.13	270.32	255.99	2
Distances: Almond Delight	292.63	219.83	329.19	342.13	0.00	73.87	103.53	1
Distances: Apple Cinnamon Cheerios	219.83	179.45	265.70	270.32	73.87	0.00	47.56	1
Distances: Biotic 4	206.49	200.21	235.84	255.99	103.53	47.56	0.00	1
Distances: Bran Chex	171.65	189.54	205.35	177.75	127.64	62.39	48.41	1
Distances: Bran Flakes	122.44	206.50	140.90	161.96	152.39	125.49	98.75	1
Distances: Cap'n Crunch	266.12	229.56	292.39	314.16	42.64	54.17	66.97	1
Distances: Cheerios	240.96	278.20	221.12	277.41	139.32	116.04	82.97	1
Distances: Cinnamon Toast Crunch	253.68	218.31	284.20	302.23	48.16	48.45	56.11	1
Distances: Clusters	180.16	151.47	249.65	233.34	121.82	53.32	75.15	1
Distances: Cocoa Puffs	234.38	185.51	260.01	284.98	59.73	15.51	58.27	1
Distances: Corn Chex	229.23	208.62	260.90	341.51	84.68	110.51	104.76	1
Distances: Corn Flakes	294.76	294.76	260.62	335.22	97.90	116.63	107.59	1
Distances: Corn Pops	266.44	189.16	347.16	320.23	112.08	103.02	145.77	1
Distances: Corn Chexia	174.78	181.42	270.56	275.35	69.20	5.32	50.94	1
Distances: Frolics Nat Bran	127.14	108.14	204.07	180.31	121.40	48.48	44.71	1

Figure 7: Top portion of distance matrix computed from the cereals.lsp data

## 2.13 Normalize

The Normalize Data transformation creates a new dataset from the current dataset in which the values of the numeric variables in the current dataset are normalized to have a specified mean and standard deviation. Ordinal and category variables are not placed into the new dataset.

The default values change the means of the transformed variables to 0 and the standard deviations to 1, as shown in Figure 8. The resulting transformed data are often called standard-scores or z scores. To change the means to 0 and not to change the standard deviations is sometimes called centering, and the result is called deviation, difference or centered scores.

Normalize data are useful for carrying out comparisons between variables measured on different scales. Figure 9 shows part of a dataset concerning the 20 American cities which are rated as the best places to live in when you are retired. The variables in this dataset are on very different scales, as can be seen. Population, for example, is the number of people, whereas “Over65” is a percentage scale. In order to compare these variables, we transform them to z-scores, which are shown in Figure 10. Since Z-scores usually range between -3 and +3, Looking at this figure we can point to Las Vegas as a very large city in comparison to the other top 20 retirement cities, which are mostly smaller towns. We also see that Chapel Hill, NC has very expensive housing, in the context of these 20 retirement locations.

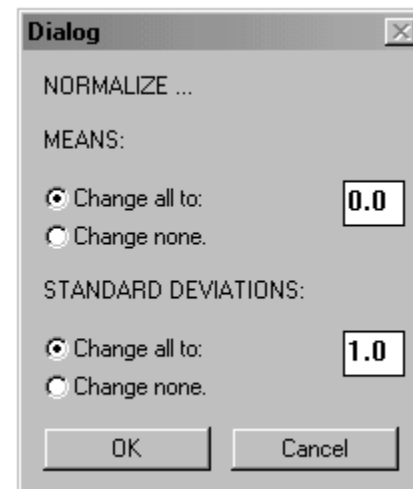


Figure 8: Dialog Box for Normalize

11 Vars	Populatio	Over65	BigCity D	Cost of L	Rent	Home Cost
20 Obs	Numeric	Numeric	Numeric	Numeric	Numeric	Numeric
Prescott AZ	28211.00	22.50	90.00	6.00	800.00	115000.00
Fairhope Ala	9000.00	23.30	20.00	-5.00	600.00	80000.00
Mt Dora FL	7500.00	27.00	25.00	-6.00	600.00	70000.00
Las Vegas NV	920000.00	13.00	180.00	5.00	1150.00	115000.00
Chapel Hill NC	41524.00	13.00	8.00	4.00	900.00	180000.00

Figure 9: A portion of the retirement data.

11 Vars	Populatio	Over65	BigCity D	Cost of L	Rent	Home Cost
20 Obs	Numeric	Numeric	Numeric	Numeric	Numeric	Numeric
Prescott AZ	-0.21	0.01	0.20	0.42	0.15	0.37
Fairhope Ala	-0.30	0.09	-1.01	-1.00	-0.69	-0.66
Mt Dora FL	-0.31	0.47	-0.93	-1.13	-0.69	-0.96
Las Vegas NV	4.22	-0.96	1.75	0.29	1.61	0.37
Chapel Hill NC	-0.14	-0.96	-1.22	0.16	0.57	2.28

Figure 10: Retirement Data after Normalization

## 2.14 Dummy code

The Dummy Code transformation creates a new dataset from the current dataset in which the values of the categorical variables of the old dataset are transformed into numeric variables in the new dataset. This results in a dataset with the number of variables equal to the number of different values in each of the categorical variables in the old dataset. The new variables are binary, having only values of 0 and 1. The new variables have a 1 when the new variable corresponds to the category value of the variable in the old dataset and a zero otherwise. The type of the variables in the new dataset is Numeric. Missing values are also converted to 0/1 values.

This transformation allows using categorical variables in computations that require a numerical variable. The main example is using a regression program to analyze data with independent variables that are categorical. Dummy-coding these variables allows for this analysis with the only limitation that  $k-1$  categories (where  $k$  is the number of categories) should be used in order to avoid exact collinearity. The command for imputation of missing data is another example where using dummy variables can be used to take advantage of information contained in categorical variables (again, at least a dummy variable by each original variable should be left unselected before using this command). Ordinal and Numeric variables are not placed into the new dataset.

## 3 User-Defined Transformations

---

The **TRANSFORM** menu's **USER DEFINED ...** menu item lets you define your own transformation program. There are two tools available to you for writing your program: The **LISTENER** and the **LISPEDIT** program editor. With these tools you can enter statements in ViSta's two programming languages: **LISP** and **VIVA**. Your program can transform variables contained in other data objects, or it can create new variables (index variables, random variables, etc). In this section we very briefly review the Listener and the Editor, and data objects and variables. Then we turn to the programming languages..

### 3.1 The Listener and the Editor

The menu item gives you the choice of using the **LISTENER** window to enter the calculations interactively, or of using the **LispEdit** program editor to create a script which can be run with the editor and/or saved for later use. The script can be run by using **LispEdit**'s **EVAL** menu item, or if it is placed in a file, it can be loaded and run using **ViSta**'s **LOAD EDIT** menu item.

The **LISTENER** is good for trying out individual statements, whereas the **EDITOR** is best for creating programs (groups of statements). The **LISTENER** gives you instant results, the **EDITOR** is for longer term development. Often it is best to use both tools simultaneously, since it can be useful to use the **LISTENER** to test statements of a program and the **EDITOR** to construct, test and save the program. These tools are discussed elsewhere.

## 3.2 Data Objects and Variables

It is important to note that the data objects which have been defined during the analysis session, and the variables in these data objects, are available to be used in your program (i.e., are bound in the local environment). Consequently, when the Listener is chosen, it informs you of the data objects and variables currently available. These objects and variables are represented by the following symbols:

- **\$** represents the current data object (the workmap's highlighted icon).
- **\$vars** a list of names of all variables in **\$**.
- **\$data** a list of names for all the data objects that have been defined.

All of the current data object's variables (as listed by **\$vars**) are available for use in your program. At any time you can change the active data object by typing its name. Then its variables become available to you. You can determine which variables are in the object by typing **\$vars**. In addition

- **\$data.vars** gives a list of two-level symbol-names of all variables in all data objects. These names are available to you at all times, regardless of which data object is the current data object. The names are made from the data object name and the variable name.

## 3.3 ViVa: ViSta's Interactive Variable Algebra language

Of ViSta's two programming languages, ViVa is a special purpose language specifically designed to transform ViSta variables and to create new ViSta data objects containing the transformed variables. Writing a ViVa expression is like writing an algebraic statement. It is familiar and easy to use (but in comparison to Lisp, limited in scope). You use the Listener to write ViVa expressions and to see results.

ViVa transforms variables and to creates new data objects containing the transformed variables. ViVa has a familiar algebraic syntax that is easy to use. It is an interactive language which is used at the listener. Note that when using ViVa, variable names cannot contain embedded characters interpreted as mathematical operators (i.e.,  $-$   $+$   $=$ , etc). In particular, the common Lisp convention of variable names with dashes, such as **a-variable-name**, is not allowed in ViVa.

ViVa can be used in the following 3 ways:

1. At the Listener, type a ViVa algebra-like expression enclosed in braces. It is evaluated and the value returned. If the expression involves assignment to a variable, the variable is bound to the expression value. For example:

```
> {A = 2 + (3*4) }
14
>
```

Here, the user has typed the ViVa expression  $A=2+(3*4)$  enclosed in braces. ViSta responds with 14, the appropriate value, using the standard rules of operator precedence.

Note that any Lisp function may be used in ViVa expressions, though in standard algebraic syntax. For example:

```
> {a=3*iseq(4)}
(0 3 6 9)
>
```

where `iseq(4)` is the ViVa equivalent of Lisp's `(iseq 4)`, which generates the sequence of numbers (0 1 2 3). Notice that a new Lisp variable `A` has been created:

```
> a
(0 3 6 9)
>
```

Note that a ViVa expression may be a compound expression:

```
> { ( x=1,y=2,print(x+y),sin(pi/2) ) }
3
1.0
> x
1
> y
2
>
```

2. At the Listener, type a left brace and a return. Lisp's `>` prompt is replaced by ViVa's `?` prompt. Then you can type a series of ViVa expressions, each being evaluated when you type a return. Finally, type a right brace to return to regular Lisp. Notice that functions with multiple arguments have their arguments separated by commas. Also, notice that vector and matrix algebra are supported. Here is an example interaction:

```
? a=2+3
5
? a           ;a new value for A has been defined
5
? b=sqrt(a)
2.23606797749979
? a=iseq(4)   ;yet another value for A
(0 1 2 3)
? b=sqrt(a)   ;vector arithmetic is supported
(0.0 1.0 1.4142135623730951 1.7320508075688772)
? }          ;the value of the last expression is returned
(0.0 1.0 1.4142135623730951 1.7320508075688772)
```

Lisp's matrix language may be used with ViVa. Here is an example:

```
?a=matrix ( list(2,2), list(1, 2 ,3 ,4))
#2A((1 2) (3 4))
?b=matrix ( list(2,3), list(1,2,3,4,5,6))
#2A((1 2 3) (4 5 6))
?c=matmult(a,b)
#2A((9.0 12.0 15.0) (19.0 26.0 33.0))
```

3. Enter a brace-enclosed ViVa statement in the middle of lisp. It is evaluated and returns its value to Lisp:

```
>(list 1 3 {sqrt(25)} 7)
(1 3 5.0 7)
>
```

Here is a real example of ViVa embedded in Lisp. In this example we calculate overall grades that students receive in a course on the basis of the raw points they have received in the course. Note that the grade is a categorical variable and that it is computed from numeric variables.

```
(datacalc midgrades (hmkw1 hmkw2 hmkw3 hmkwsum mid1 midpts grade)
  (progn
    {homeworksum = homework1 + homework2 + homework3}
    {midpts = homeworksum + midterm1}
    (let ((grades (mapcar #'(lambda (pts)
      (cond ((> pts 190) "A")
            ((< 170 pts 190) "B")
            ((< 150 pts 170) "C")
            ((< 140 pts 150) "C-")
            ((< 0 pts 150) "D"))
      midpts)))
      (list homework1 homework2 homework3
        homeworksum midterm1 midpts grades)))
    :title "DataCalc example"
    :types (combine (repeat "Numeric" 6) "Category")
    :about "Psych 30 Grade determined from points by DataCalc"
    :labels (mapcar #'(lambda (i) (format nil "Student~d" i))
      (iseq (length homework1)))))
```

**About ViVa, C, Lisp and Parcil:** ViVa statements conform to a subset of the syntax for the C programming language. Specifically, ViVa supports all syntax defined in section 18.1 of the original Kernighan and Ritchie book, plus all C numerical syntax including floats and radix syntax (i.e. 0xnnn, 0bnnn, and 0onnn). ViVa supports multiple array subscripts, and also allows strings to be delimited using single quotes as well as double quotes (but you must use the same type to close the string as you did to open it).

ViVa uses Parcil, which parses expressions in the C programming language into Lisp. ViVa then takes the parsed statements and creates an interactive environment in which they are evaluated. Parcil is copyright (c) 1992 by Erann Gat, and is used under the terms of the GNU General Public License as published by the Free Software Foundation.

### 3.4 The Lisp DataCalc Macro: ViSta's Data Calculator<sup>1</sup>

Lisp is a general purpose language which is very general and powerful, but which is difficult to learn. It can be entered from the listener or from files via the editor. The general nature of Lisp is way beyond the scope of this book: In this section we only discuss DataCalc, the Lisp macro which transforms data objects into new data objects. Lisp is described in Luke Tierney's book on Lisp-Stat, available from Wiley.

**The DataCalc macro:** You can use any Lisp statements to define your program, but the **DataCalc** macro expression is specifically designed to create a new data object containing variables derived from variables in existing data objects. It can also create a new data object with variables computed independently from existing variables. The statement's syntax is:

```
(datacalc name (variables) <in-data>
  body)
```

**DATA CALC** creates a new data object **name**, containing **variables** calculated according to the instructions in **body**. The calculations can use variables in **in-data**, an optional argument specifying an existing data object. The arguments are as follows:

<b>name</b>	is a symbol that becomes the name of the new data object. A symbol is represented by characters without single or double quotes and with no embedded blanks.
<b>variables</b>	is an unquoted list of symbols specifying the names of the variables placed in the new data object.
<b>in-data</b>	is an optional symbol argument specifying the input dataobject. Some, all or none of the variables in <b>in-data</b> may be used in the body of <b>DataCalc</b> . When omitted it is assumed that the current dataobject is to be used, if there is one.
<b>body</b>	is a program of any Lisp statements which calculates values for the <b>variables</b> . The program must return information which satisfies these conditions: <ol style="list-style-type: none"> <li>1. the returned information must be a list of lists</li> <li>2. The number of lists must equal the number of <b>variables</b></li> <li>3. Each list must have the same number of elements.</li> </ol>

For example, if you **OPEN DATA** named **P309799.lsp** (found in the **data** folder), you will have a ViSta data object on your WorkMap named **P309799**. The data object contains variables **SATMATH** and **SATVERB**. The following expression defines a new data object named **SAT**, calculates a variable **SUM** as the sum of **SATMATH** and **SATVERB**, and another variable **DIF** as the difference between **SATMATH** and **SATVERB**, and puts the two original and two new variables in the new dataobject. Since **P309799** is the current data object it is not necessary to specify it.

```
(DataCalc sat (sum dif satmath satverb)
  (let ((sum (+ satmath satverb))
        (dif (- satmath satverb)))
    (list sum dif satmath satverb)))
```

---

1. Note that this section involves Lisp and may be skipped without loss of continuity.

As a second example, you can simulate SAT scores and total them with the following code. Notice that there is no input data object since the “variables” are generated by the random-number generator.

```
(DataCalc simulated-sat (total sat-math sat-verb)
  (let ((sat-math (+ 500 (* 100 (normal-rand 50))))
        (sat-verb (+ 500 (* 100 (normal-rand 50)))))
    (list (+ sat-math sat-verb) sat-math sat-verb)))
```

The body of **DataCalc** can be followed by optional keyword arguments. The keywords are **:types** **:labels** **:freq** and **:about** (and any other keywords that are used with the data function). These arguments have the following meaning:

- :types** must be followed by a list of strings "numeric", "ordinal" or "category" (case ignored), specifying whether the variables being created in the new data object are numeric, ordinal or categorical (all Numeric by default).
- :labels** must be followed by a list of strings specifying observation names ("Obs1", "Obs2", etc., by default).
- :freq** followed by **t** (true) to specify that the values of the numeric variables are frequencies.
- :about** followed by a string of information about the data. This information will be used by the "About These Data" menu item.

Here is another version of the example of calculating grades from student points.

```
(DataCalc midgrades (homework1 homework2 homework3
  homeworksum midterm1 midpts grade)
  (let* ((homeworksum (+ homework1 homework2 homework3))
        (midpts (+ homeworksum midterm1)))
    (list homework1
      homework2
      homework3
      homeworksum
      midterm1
      midpts
      (mapcar #'(lambda (pts)
        (cond ((> pts 190) "A")
              ((< 170 pts 190) "B")
              ((< 150 pts 170) "C")
              ((< 140 pts 150) "C-")
              ((< 0 pts 150) "D"))
        midpts)))
    :types (combine (repeat "Numeric" 6) "Category")))
```



## 4 Visual Transformations

---

There are two visual transformations: The Box-Cox and Folded-Power transformations. We discuss them in this section.

### 4.1 Box-Cox transformations

The Box Cox family of transformations create a new dataset with some of the original variables transformed by a function which has an argument whose value is determined by a slider. As a consequence of manipulating the slider, the user modifies the argument and thus determines which function will be used to transform the variables.

This process will be exemplified with data on the forty largest countries in the world (according to 1990 population figures) that include the following variables:

<b>LifeExpec</b>	the country's life expectancy at birth
<b>LifeExpecMal</b>	life expectancy for males
<b>LifeExpecFem</b>	life expectancy for females
<b>PeopleTV</b>	the number of people per television set
<b>PeoplePhy</b>	the number of people per physician.

These data were taken from the World Almanac and Book of Facts (1993) by A. J. Rossman to be used as an example about correlation and transformations in a context easy to understand. They are included in the file **tele.lsp**.

The data have missing values that could be dealt with by using the **DATA** menu's **IMPUTE MISSING DATA** item. However, as this command is based on multiple regression, and as this technique requires data that are normally distributed, the need for transformation of the data should be examined in advance.

Preliminary examination of this data reveals a strong positive asymmetry in the variables **PeopleTV** and **PeoplePhy**. A set of transformations capable of dealing with this kind of problem is the Box-Cox family of transformations (Emerson, 1990). The formula for the box-cox family of transformations is:

$$f(y) = \begin{cases} y^p & (p > 0) \\ \ln(p) & (p = 0) \\ y^{-p} & (p < 0) \end{cases} \quad (\text{EQ 1})$$

Values of  $p$  in formula 1 only make sense normally in the range of -1 to 2. In this range the produce approximations to many well-known transformations. For example, a value of  $p=2$  gives the square transformation,  $p=0.5$  is the square root,  $p=0$  is the log transformation, and  $p=-1$  is the inverse transformation. Values of  $p$  below 1 are known to improve the distribution of positively asymmetric distributed variables. Values of  $p$  above 1, in turn, will benefit negatively asymmetric distributed variables. You should be aware of values in your variables equal or minus zero because logarithm of zero or negative values will cause an error.

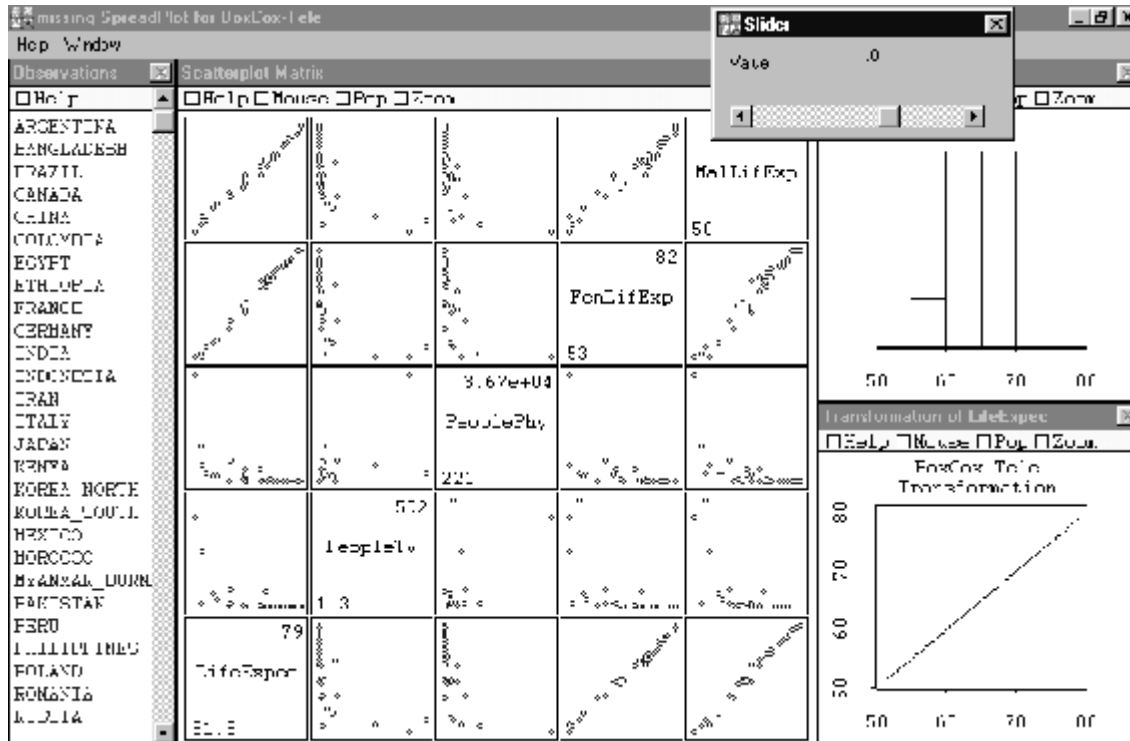


Figure 11: Spreadplot for Box-Cox Transformation

Generally, the analyst will try successive p-values looking for the plots of transformed data that conform to the intended shape. In particular, he/she will search for symmetric histograms for individual variables and closely linear and homoskedastic scatterplots for pairs of variables.

Univariate changes are not only useful per se, but also because of their effects on the multivariate distribution of the data. These effects are 1) to make non-linear relationships more linear, 2) to promote constant variability across groups and 3) to improve additivity of a response in relation to two or more factors (Emerson, 1991). All these consequences should be examined in the data simultaneously in order to get a complete view of the results of the transformation process.

The visualization for Box Cox transformations is shown in Figure 11. It includes a scatterplot matrix of the variables, a histogram of the variable currently selected (by clicking on the diagonal of the scatterplot matrix), a slider that controls the transformation parameter, a line plot of the original versus transformed data, and a list of the labels of observations. Initially, the transformations are all linear and the first variable in the dataset is the selected variable.

The slider in the spreadplot in the figure is linked to the currently selected variable. Hence, pointing and clicking the mouse on the arrows of this slider applies the Box-Cox formula to this variable using the p-value shown on the slider. Recall that transformations with  $p > 1$  make left-skewed distributions more symmetric and transformations with  $p < 1$  have the same effect

with right-skewed distributions.  $p=1$  means no transformation at all. The histogram in the Box-Cox spreadplot allows the assessment of this effect because it is connected to the transformation of the variable currently selected. A data object for the transformed data is created when the BoxCox menu item is used, and the data object is updated automatically. Thus, it is possible to check the transformed values anytime.

The scatterplot matrix, which gives us information on the linearity and homoskedasticity of the bivariate relationships among variables, varies as changes are made in the slider. Finally, the line-plot named "power of the transformation" gives us information on the relationship between original and transformed data. It, too, changes as the slider is moved.

The scatterplot matrix reveals several non-linear and non-homoskedastic relationships. For example, the relation between PeopleTV and LifeExpec seems clearly non-linear. Not surprisingly, the relationship between LifeExpec and its counterparts, LifeExpecMal and LifeExpecFem, is clearly linear. Histograms for PeopleTV and PeoplePhy are shown separately in figure 10 and show positive asymmetry. BoxCox transformations with  $p < 1$  will probably reduce this asymmetry and will bring the data closer to the assumptions of univariate and multivariate normality.

Successive values of  $p$  can be tested until identifying a more satisfactory shape of the plots. The visualization in Figure 12 shows the set of transformations that the data analyst liked the

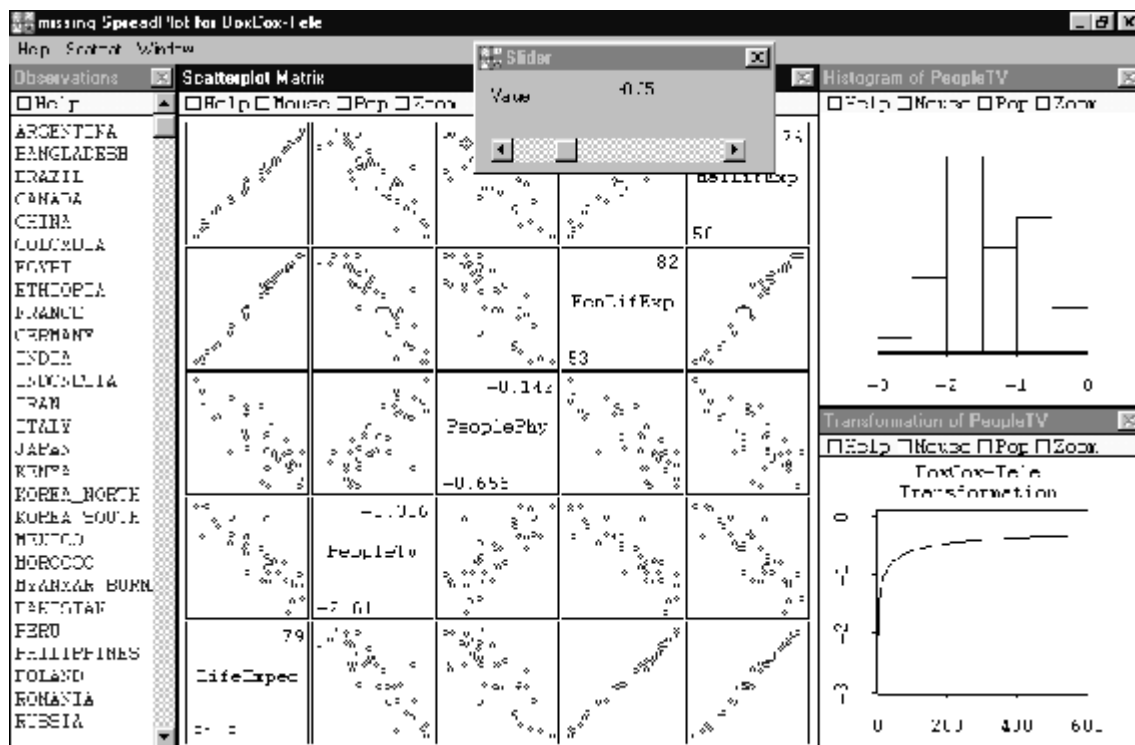


Figure 12: Box-Cox transformation spreadplot after applying the transformations to the variables. Note the linearity of the scatterplots.

best. Values of  $p$  were -0.30 for PeopleTV and -0.35 for PeoplePhy. The other variables were

left untouched (hence  $p=1$ ).

The examination of the matrix of scatterplots in figure reveals linear relationships between the transformed variables and the rest of variables. Other interesting aspects of the data, like the bivariate outlier corresponding to North Korea, can also be appreciated more easily.

BoxCox transformations are a powerful resource for the data analyst. However, the transformations will not always attain the desired distributional characteristics. This case is generally due to extreme values that do not fit in the distribution of the rest of values even after transformation. Special treatment of these values can be necessary in this case.

Another problem with transformations is that interpretation of the new data can be more difficult than that of the original, untransformed data. For example, working with logarithms of the variables instead of the raw variables can be unnatural for some users. These problems should not restrain the user from using these transformations in order to solve the more critical obstacle of lack of agreement with assumptions. For a discussion of these issues and others related see Emerson (1991).

## 4.2 Folder Power Transformations

The family of Folded Power Transformations may be well suited to examining counted fractions or proportions when some of the values in the variables to be transformed are close to 0 and others are close to 1. ViSta offers a visualization for carrying out these transformations that parallels the visualization for BoxCox transformations. Figure 13 shows the spreadplot for Folded Power Transformations. The data in the example are the variables **LogDose** and **PropKill** in the file `Dose.lsp`. **PropKill** is the proportion of insects dead after five hours' exposure to gaseous carbon disulphide at various concentrations and **LogDose** is the logarithm of the concentrations. These data were gathered by Bliss (1935) and reported by Dobson (1990) to exemplify the application of dose response models to binary data. The file `Dose.lsp` also includes the actual number of insects and the number dead at the different concentrations (not used in this example).

Proportions or counted fractions are special because of the barriers imposed on the extremes, 0 and 1, of the data. The shape of the histogram for this kind of data has typically a higher shape than gaussian in the middle and narrower at the extremes. This results in bivariate relationships that are basically linear in the center but curvilinear in both extremes, such as shown in the scatterplot of Figure 13. Therefore, we need transformations of proportions that are linear in the center of the data but folds data in both tails. Alternatively, BoxCox transformations can be used with proportions when all the values are near one of the extremes, 0 or 1, because, in this case, a transformation that works on both sides is not needed.

The family of Folded Power Transformations has the properties required. The formulae for these transformations are (Emerson, 1991):

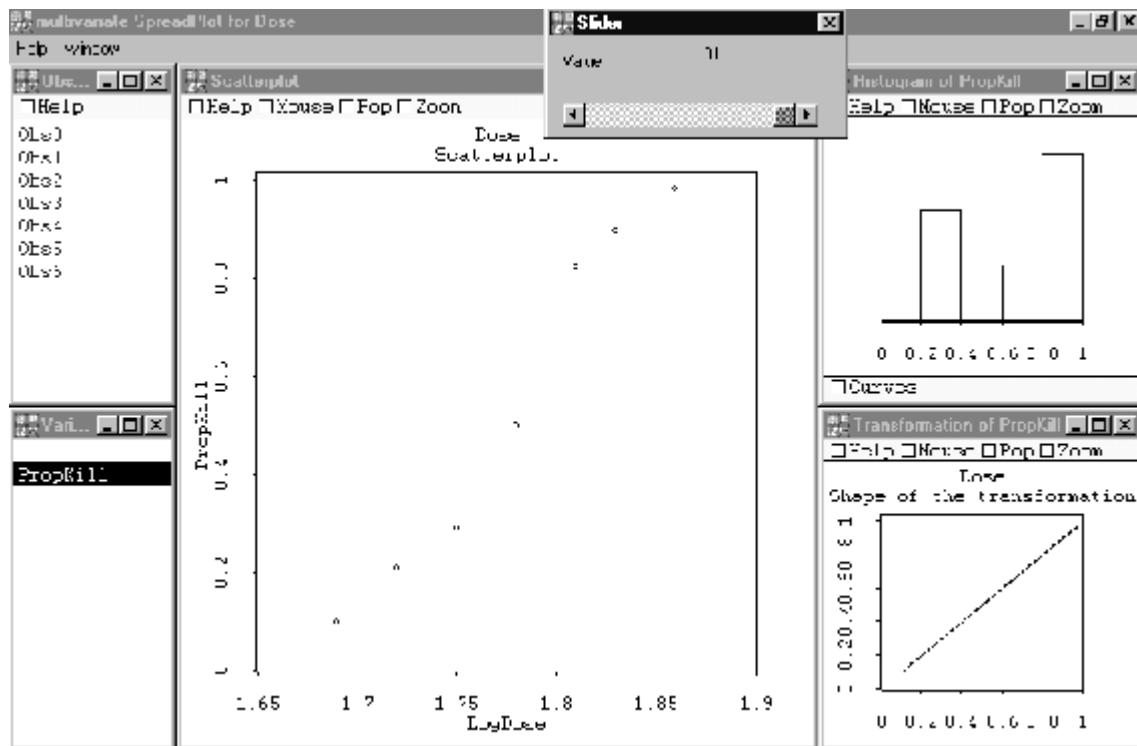


Figure 13: Folded Power Transformation Visualization

$$T_p(y) = \begin{cases} \frac{(2y)^p - (2-2y)^p}{2p}, & p \neq 0 \\ \frac{\log_e y - \log_e(1-y)}{2}, & p = 0 \end{cases} \quad (\text{EQ 2})$$

Some (statistically) interesting values of  $p$  are:

1: plurality

0.5 folded root square

0.41 Arcsin

0.14 Probit

0 Logistic

Dobson (1990) also discusses the complementary log log transformation, which is given by the formula:

$$T(y) = \log(\log(1-y)) \quad (\text{EQ 3})$$

This transformation is produced in ViSta using values in the slider below 0. On the opposite

side, values in the slider above 1 produce an identity (no) transformation. This is because folded power transformations with values of  $p$  above 1 are rarely used and so the user has the option of not performing any transformation on a particular variable. Also, if any of the values of the variable include values above 1 or below 0 the folded power transformation can not be applied. Notice that there is a list of variables in the spreadplot for Folded-Power transformations. Variables that include values above 1 or below 0 are made invisible in this list so they can not be chosen for transforming. Finally, when the values in the variable include values of proportions equal to 0 or 1, both the logistic transformation and the complementary log log transformation are not applicable but the rest are. In this case, the function to compute the transformation will output the same result as would be produced using  $p=0.01$ .

Figure 13 shows the visualization for  $p=-0.01$ . This value of  $p$  results in ViSta in the complementary log log transformation. Previously to this transformation the last observation of the data has been excluded because its value is 1. The scatterplot shows that this transformation linearizes the relationship remarkably well. Dobson (1990) shows that this transformation is the best among others that can be fit using the program GLIM.