DATA
For more information about SYSTAT® software products, please visit our WWW site at http://www.systat.com or contact

Marketing Department
Systat Software, Inc.
225 W. Washington Street, Ste. 425
Chicago, IL 60606
Phone: (877) 797-8280
Fax: (312) 220-0070
Email: info-usa@systat.com

Windows is a registered trademark of Microsoft Corporation.

General notice: Other product names mentioned herein are used for identification purposes only and may be trademarks of their respective companies.

The SOFTWARE and documentation are provided with RESTRICTED RIGHTS. Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subdivision (c)(1)(ii) of The Rights in Technical Data and Computer Software clause at 52.227-7013. Contractor/manufacturer is Systat Software, Inc., 225, W. Washington Street, Suite 425, Chicago, IL 60606. USA.

SYSTAT® 13 Data
Copyright © 2009 by Systat Software, Inc.
Systat Software, Inc.
225 W. Washington Street, Ste. 425
Chicago, IL 60606
All rights reserved.
Printed in the United States of America.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the publisher.

1 2 3 4 5 6 7 8 9 0 05 04 03 02 01 00
## List of Examples

ix

## 1 Working with Data

1

## 2 Data Files

3

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Editor</td>
<td>3</td>
</tr>
<tr>
<td>Opening Data Files</td>
<td>6</td>
</tr>
<tr>
<td>Import Formats</td>
<td>7</td>
</tr>
<tr>
<td>Reading Variable Names</td>
<td>7</td>
</tr>
<tr>
<td>ASCII File Import</td>
<td>8</td>
</tr>
<tr>
<td>Excel, SigmaPlot and Lotus File Import</td>
<td>8</td>
</tr>
<tr>
<td>dBASE File Import</td>
<td>9</td>
</tr>
<tr>
<td>SAS File Import</td>
<td>9</td>
</tr>
<tr>
<td>MINITAB File Import</td>
<td>10</td>
</tr>
<tr>
<td>STATISTICA File Import</td>
<td>11</td>
</tr>
<tr>
<td>JMP File Import</td>
<td>11</td>
</tr>
<tr>
<td>Import from Database with ODBC</td>
<td>11</td>
</tr>
<tr>
<td>Databases</td>
<td>13</td>
</tr>
<tr>
<td>Variable Selection</td>
<td>14</td>
</tr>
<tr>
<td>Importing Business Objects</td>
<td>16</td>
</tr>
<tr>
<td>Saving Data Files</td>
<td>24</td>
</tr>
<tr>
<td>Save Options</td>
<td>25</td>
</tr>
<tr>
<td>File Comments</td>
<td>26</td>
</tr>
<tr>
<td>Export Formats</td>
<td>26</td>
</tr>
<tr>
<td>Temporary Data Files</td>
<td>28</td>
</tr>
<tr>
<td>Creating Temporary Data Files</td>
<td>28</td>
</tr>
<tr>
<td>Accessing Temporary Data Files</td>
<td>29</td>
</tr>
</tbody>
</table>
3 Entering and Editing Data

Data Editor ................................................. 43
Data File Structure ...................................... 44
Variable Editor ............................................. 45
  Pasting Variable Properties ............................ 49
  Processing Conditions in Effect ....................... 49
Variable Properties Dialog Box ........................ 50
  To Specify Variable Properties ......................... 52
Data Entry .................................................. 55
  Editing Data .............................................. 56
  Data Editor Navigation ................................ 56
  Selecting Cells ......................................... 57
  Correcting Data Values ................................ 58
  Data Edit Bar ............................................ 59
  Changing Variable Names and Types .................. 60
  Adding Variable(s) or Case(s) ......................... 60
  Dropping Variable(s) .................................. 62
  Deleting Case(s) ........................................ 63
  Cutting/Copying and Pasting Data ...................... 63
  Right-Click Editing .................................... 66
  Find Values in Column ................................ 69
  Replace Values in Column .............................. 69
  Find Variable .......................................... 70
  Go To .................................................... 71
  Undo/Redo ............................................... 72
4 Data Transformations

Let Dialog Box .................................................. .83
If...Then Let Dialog Box ....................................... .85
    Built-In Functions ........................................ .86
    Shortcuts for Transformations ............................ .87
Operators and Functions ....................................... .88
    Arithmetic Operators .................................... .88
    Relational Operators .................................... .89
    Logical Operators ....................................... .89
Order of Expression Evaluation .............................. .91
Mathematical Functions ....................................... .93
Multivariable Functions ...................................... .94
Multicase Functions ......................................... .96
Group and Interval Functions ................................ .97
Date and Time Functions ...................................... .98
Character Functions ......................................... .102
Functions Relating to Probability Distributions .......... .109
Random Number Generators .................................. .130
Rank Dialog Box ................................................. .131
Center Dialog Box .............................................. .132
Standardize Dialog Box ....................................... .133
Trimming Data .................................................. .134
Reshaping Data .................................................. .135
    Transpose ................................................ .135
    Wrap/Unwrap ............................................. .137
By Groups Dialog Box ........................................ 190
By Groups versus Plot Groups .......................... 191
Using Commands ........................................... 192
Examples ..................................................... 195

7 BASIC Programming and Data Management 203

Data Entry and BASIC .................................... 204
  Free-Format Input ........................................ 205
  Fixed-Format Input ...................................... 213
  Entering Triangular Matrices ......................... 216
  Troubleshooting ASCII Files ......................... 218
Variable Types ........................................... 220
Data Output ............................................... 221
  Export Data to an ASCII Text File .................. 221
  Printing Data and Text ................................ 222
Conditional and Looping Statements .................. 223
  IF...THEN Statement .................................. 223
  ELSE Statement ........................................ 224
  FOR...NEXT Statement ................................ 228
  WHILE...ENDWHILE Statement ....................... 232
Subscripted Variables .................................. 233
  Dimensioning Space for New Variables -
    The DIM command .................................... 233
  Dummy Subscripted Variables - The ARRAY command 235
  Temporary Arrays - The ARRAY Command ............. 239
Functions ................................................ 245
Macro ..................................................... 247
Programming Examples .................................. 248
  Select, List, and Save Cases ......................... 248
  Subgroup Processing: BOG(), EOG(), BOF(), EOF() 252
References ................................................. 255
8 Matrix

Matrix in SYSTAT .........................................................258
Read Matrix Dialog Box .............................................258
Generate Matrix Dialog Box .........................................260
Organize Dialog Box ..................................................261
Row/Column Operations Dialog Box. .............................262
Matrix Operations Dialog Box .......................................264
Decomposition Dialog Box ..........................................267
Using Commands .......................................................269
Examples .................................................................270

Acronym & Abbreviation Expansions

Index
List of Examples

Alphabetical Intervals ................................................. 200
Calendar for the Year 2009-2010. ............................. 242
Categorizing Variables .................................................. 200
Changing the Case of Character Values ....................... 149
Coding Missing Values ............................................... 210
Collapsing Categories .................................................. 197
Computing F Statistic .................................................. 159
Computing Means of Subscripted Variables ..................... 236
Computing Means of Unsubscripted Variables: ARRAY ......... 237
Computing the Area under a Curve ............................... 147
Computing Totals within Groups .................................... 253
Concatenating Character Strings .................................... 155
Converting a Data File to a Text File ............................ 221
Converting Numbers from European to American Notation .... 153
Covariance Matrices .................................................. 217
ID Variables ......................................................... 173
IF...THEN DELETE: Using the first N Cases .................... 254
IF...THEN...ELSE Statement ................................. 225
Inputting Data at the Command Prompt ....................... 206
Inputting Fixed Format Data with Backslash (\) ............... 216
Justifying Character Values .................................... 150
Labeling Multiple Variables .................................... 197
Lagging Variables ................................................. 143
Listing All Variables and Cases ................................ 172
Listing Cases Subsets for Selected Variables ................. 173
Manipulating Matrices .......................................... 278
Matching Specific Values ....................................... 148
Matching Values ................................................... 199
Matrices with Missing Diagonals: DIAGONAL ................. 218
Matrix Algebra ..................................................... 282
Matrix Decomposition ........................................... 288
Merging with a Key Variable .................................... 39
Moore-Penrose Generalized Inverse ......................... 310
Multiline IF...THEN Statement ......................... 226
Multiline IF...THEN...ELSE Statement ................ 227
Multiple Regression and Canonical Correlation .......... 307
Nested Sort ........................................... 177
Number of cases, number of variables, and number of columns in the file .................. 162
Omitting Variable Names and Case Numbers ................ 250
Packing Two Records into One ....................... 293
Picture Format: Displaying Dates ..................... 176
Picture Format: Displaying Many Variables in One Panel .................. 175
Printing File and Temporary Variables .................. 222
Printing the First Three Cases ...................... 251
Random Subsample with Replacement ................... 241
Random Subsamples without Replacement .............. 239
Random Walks ...................................... 231
Reading an ASCII Text File: GET ...................... 206
Reading Incomplete Records: Backslash() ............. 213
Reading Multiple Cases per Record ................... 212
Reading Multiple Lines per Case ...................... 207
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reading Records of Unequal Length</td>
<td>208</td>
</tr>
<tr>
<td>Recoding and Labeling Categories and Intervals</td>
<td>195</td>
</tr>
<tr>
<td>Recoding Characters to Numbers</td>
<td>199</td>
</tr>
<tr>
<td>Recoding Numeric Variables</td>
<td>198</td>
</tr>
<tr>
<td>Ridge Regression</td>
<td>298</td>
</tr>
<tr>
<td>Saving Data as Text without Quotes: PRINT</td>
<td>223</td>
</tr>
<tr>
<td>Saving Selected Cases to a New File</td>
<td>249</td>
</tr>
<tr>
<td>Selecting a Subset of the Variables</td>
<td>37</td>
</tr>
<tr>
<td>Selecting Cases with Equal Values of Some Variables</td>
<td>179</td>
</tr>
<tr>
<td>Selecting Subset of Cases</td>
<td>179</td>
</tr>
<tr>
<td>Side-By-Side Merge</td>
<td>38</td>
</tr>
<tr>
<td>Simple Sort</td>
<td>177</td>
</tr>
<tr>
<td>Slope of the Line of Best Fit</td>
<td>145</td>
</tr>
<tr>
<td>Sorting in Ascending and Descending Order</td>
<td>178</td>
</tr>
<tr>
<td>Stacking Variables</td>
<td>161</td>
</tr>
<tr>
<td>Standardizing Age</td>
<td>158</td>
</tr>
<tr>
<td>Statistical Functions</td>
<td>285</td>
</tr>
<tr>
<td>Subgroup Means</td>
<td>254</td>
</tr>
</tbody>
</table>
Summary Statistics .............................. 145
The Sweep Function ............................ 301
Trimming Cases for Selected Variable(s) .................. 159
Unpacking Records .............................. 230
Use of NCAT Function .......................... 201
Using PRINT to Save Selected Cases ................. 250
Using Soundex to Code Names .................. 156
Using WHILE…ENDWHILE Loop .............. 233
Writing a Correlation Matrix as a Vector ............ 294
Statistical and graphical analyses require data. Formatting data for a desired analysis can often be as challenging as interpreting the output. Fortunately, SYSTAT offers several transformations and manipulations designed to facilitate working with data. In addition, SYSTAT includes BASIC programming commands globally and includes matrix algebra, providing flexibility for advanced data management, under Utilities menu.

This manual focuses on tools available on the File menu, Edit menu, Data menu, and Utilities menu, and is organized as follows:

- **Chapter 2** focuses on reading and writing data files.
- **Chapter 3** describes the Data editor. This editor serves both to view and to edit your data.
- **Chapter 4, Chapter 5** and **Chapter 6** introduce transformations and manipulations of data, including selecting cases, sorting cases, and stratifying analyses.
- **Chapter 7** discusses the reading and writing of data in unique formats using the commands of the BASIC language.
- **Chapter 8** describes SYSTAT’s matrix algebra tools.
Data Files

Leland Wilkinson, Laszlo Engleman, Michael Pechnyo, and Lou Ross

A SYSTAT data file includes not only the data, but a great deal of information on the data files and the variables. These are explained in the sequel. You can create a new SYSTAT data file by entering data in the Data editor or by importing data from another application.

Data Editor

The active file is displayed in the Data editor, where you can also enter and edit data, run transformations, and select subsets of cases.
To view the active data file in the Data editor, from the menus choose:

View
   Data Editor...

or

Click on the Data tab of the data file in the Viewspace.

To view the variable information of the variables used in the active data file click on the Variable tab in the Data editor.

or

Right-click on the Data tab in the Viewspace and select Data/Variable editor. To toggle between Data and Variable editor, <CTRL + SPACEBAR> can be used.

To open an empty Data editor for creating a new data file, from the menus choose:

File
   New
      Data...

or

Right-click the Data or Variable tab of the active data file in the Viewspace and select New from the right-click menu.
To view multiple data files in the Data editor, go to the Output Organizer, right click on the data file you want to view and select View Data. But only one data file is active at a time.

**Variable tab.** There is a Variable tab in Data editor, besides the Data tab.

The Data tab shows the data file. The Variable tab displays the Variable editor. There are two parts; one part provides information about variable names, variable type, width, display option, decimal places, variable comments, variable labels, value labels, format and category variable status. The other part shows Processing conditions like Frequency, By, Case Selection, Category, Weight, Order, and ID. Double-click on a variable name or click the Variable tab to get the Variable editor. For more information on Data editor see Chapter 3.

You can see variable comments by moving the cursor to the variable name in the Data editor. Labels as well as comments are displayed in the tool tip, in the form: labels: comments.
**Opening Data Files**

To open a data file, from the menus choose:

File
   Open
   Data...

SYSTAT data files are listed by default, and these files are read only. Descriptions of these data files are available in the Data Files Appendix of *Getting Started*. These are also available in the right-click of Data editor tab in File Comments dialog and variable description in the Comments box of the Variable Properties dialog. To open a file of a different type, choose All Files from the Files of type drop-down list. The available types include: SPSS, SAS, Excel, SigmaPlot, Lotus, dBase, ASCII, MINITAB, STATISTICA, STATA, JMP, DIF, StatView and ArcView. To open a database of another type, use the Database capture facility.
**Import Formats**

SYSTAT can open data files saved in the following formats:

- SYSTAT, FASTAT, and MYSTAT (*.SYS, *.SYD, *.SYZ)
- SIGMAPLOT (*.JNB)
- Excel (*.XLS)
- SPSS (*.SAV)
- SAS (*.SD2, *.SAS7BDAT, *.XPT, *.TPT)
- MINITAB (*.MTW)
- STATISTICA (*.STA)
- STATA (*.DTA)
- JMP (*.JMP)
- dBASE (*.DBF)
- ASCII text (*.TXT, *.DAT, *.CSV)
- ArcView (*.SHP)
- LOTUS (*.WK1, *.WK2, *.WKS)
- DIF files (*.DIF)
- STATVIEW (*.SVD)

To import any S-PLUS files you must employ SYSTAT's command language.

In addition, using ODBC import, SYSTAT can import data from any database with a corresponding driver installed on your computer.

**Reading Variable Names**

Variable name restrictions vary from product to product. To successfully import data from another application, variable names undergo any necessary modifications to yield valid, unique SYSTAT names. SYSTAT reads the ASCII file and scans the first 16 cases of each variable to determine if it is a string or numeric type of variables. If it encounters even one string data then the variable is designated a string variable and a $ character is appended to the variable, (if not already available at the end of the variable name). If the variable is designated a numeric variable, then the variable is scanned to check that no illegal characters are present in the name (e.g., $ is illegal for
a numeric variable name) and substitutes them with underscore (_). If there are no variable names in the source file and if the first row contains numeric information, SYSTAT creates the variable names C1, C2,... Cn and C1$, C2$, ..... for string information.

Blank cells. SYSTAT considers blank cells to be numeric. Therefore, blank cells in the first row of a spreadsheet file are considered to be missing numeric values. This means that the first row is interpreted as data rather than as variable names.

Variable names. The length of the variable name can go up to 256 characters.

Subscripted variables. Variable names followed explicitly by subscripts (i), where i ranges from 1 to n, are subscripted when converted. String variable names can also be subscripted. For example- RATS$(1), RATS$(2).

ASCII File Import

An ASCII text file is created in a word processor or text editor and contains text and numbers with no special characters or formatting. Before you try to read an ASCII file, check the following:

- Each case is written as one line.
- The variable names are written at the top of the file and are separated by spaces and/or a comma.
- The values of the variables are separated by one or more spaces (blanks) and/or a comma.
- Missing numerical data are flagged by a period (.) and missing character data are marked by a space.

Excel, SigmaPlot and Lotus File Import

The following general rules apply to reading of Excel, SigmaPlot and Lotus files:

- You can import Excel, SigmaPlot and Lotus worksheets only if they are saved in row and column order. Although Lotus can read files stored in other orders, row and column is the default.
- If an Excel file contains multiple worksheets, SYSTAT numbers the sheets in the order in which they appear in the file. You can access any sheet by specifying the sheet number. When you import an Excel file (*.XLS, *.XLSX) that has multiple
sheets, a dialog appears, that contains the names of the sheets with their sheet numbers. You can select the sheet you want from the drop down list. If a SigmaPlot notebook (*.JNB) contains multiple worksheets, then only the first sheet will be imported.

- Any leading blank rows and columns are skipped.
- Numeric results of Lotus formulae are imported, with a warning issued to indicate that the imported data may not be accurate. If numbers were changed and the spreadsheet was saved before recalculating, the values saved may not reflect the changes that were made.
- Excel formulae cannot be imported.
- Variables displayed in SigmaPlot using date/time formats are displayed in SYSTAT as numeric variables, which can be changed to date/time format by changing the variable properties.

**dBASE File Import**

The following general rules apply to reading of dBASE files:

- For dBASE III and IV, SYSTAT changes logical fields to character fields. The character used for the dBASE logical value is retained.
- SYSTAT skips memo fields, issuing a warning message when this occurs.
- Date fields are converted to numeric values. For example, if your dBASE file contained a value for November 2, 1987, it would be stored as 19871102. SYSTAT converts this to 0.19871102E + 8.

**SAS File Import**

SYSTAT reads data and transport files created by version 6 (release 6.08 to 9.2) of SAS. However, the SAS procedure used to create a transport file determines SYSTAT’s ability to read that file. Transport files created using PROC COPY, with the XPORT engine specified in the LIBNAME statement, can be imported. On the other hand, SAS transport files created using PROC CPORT cannot be imported. If a SAS transport file consists of multiple SAS data sets, SYSTAT gives a sheet number to each of the data sets. You can access the data sets by specifying the sheet number.
The following rules apply to the treatment of variables when importing SAS files:

**Variable types**
- Variables displayed in SAS using numeric formats are displayed in SYSTAT as numeric variables.
- Variables displayed in SAS using string formats appear as string variables. SAS variables declared as strings are imported as string variables, even if all values are numeric.
- Variables displayed in SAS using date, time, or date/time formats are displayed in SYSTAT in a numeric format.

**Variable values**
- Missing values in SAS files are supported in SYSTAT.
- SYSTAT does not import variable and value labels from SAS files.

**MINITAB File Import**

SYSTAT reads data created by versions 8-13 of MINITAB. Although MINITAB project files can contain multiple MINITAB worksheets along with other data, SYSTAT does not extract separate worksheets from these project files.

The following rules apply to the treatment of variables when importing MINITAB files:
- Variables displayed in MINITAB using numeric formats are displayed in SYSTAT as numeric variables.
- Variables displayed in MINITAB using date/time formats are displayed in SYSTAT as numeric variables, which can be changed to date/time format by changing the variable properties.
- Missing values in MINITAB files are supported in SYSTAT.
**STATISTICA File Import**

SYSTAT reads data created by version 5 of STATISTICA. The following rules apply to the treatment of variables when importing a STATISTICA file:

- Variables displayed in STATISTICA using numeric formats are displayed in SYSTAT as numeric variables.
- Variables having character values in STATISTICA are labeled and displayed as numeric values in SYSTAT.
- Variables displayed in STATISTICA using date/time formats are displayed in the SYSTAT numeric format. However, on changing the variable properties from numeric to date/time format, SYSTAT displays date/time with the chosen format.
- Missing values in STATISTICA files are supported in SYSTAT.

**JMP File Import**

SYSTAT reads data created by version 3.2 to 5 of JMP. The following rules are applied to the treatment of variables when importing a JMP file:

- Variables displayed in JMP using numeric formats are displayed in SYSTAT as numeric variables.
- Variables displayed in JMP using date/time formats are displayed in SYSTAT in numeric format. However, on changing the variable properties from numeric to date/time, SYSTAT displays date/time with the chosen format.
- Missing values in JMP files are supported in SYSTAT.

**Import from Database with ODBC**

Open Database Capture or Connectivity (ODBC) provides a method for reading data into SYSTAT from any database format for which a driver is installed on your system. ODBC import provides the ability to access data stored in formats which SYSTAT’s standard import cannot recognize. In addition, you can also use ODBC import as an alternative to the standard import by using the appropriate driver. Because ODBC allows selection of variables and cases to import, you have greater control over the resulting data set.
ODBC import involves translating an import request from SYSTAT to the Structured Query Language (SQL). The ODBC driver translates the SQL to a format which a database type can process and returns the requested data. ODBC drivers included with SYSTAT allow importing data in the following forms:

- MS-Access
- Btrieve
- DB2 Common Server (IBM)
- Clipper and FoxPro files
- INFORMIX
- INFORMIX 9
- OpenIngres
- OpenIngres 2
- Oracle
- Paradox tables
- PROGRESS
- SQL Server (Microsoft)
- SQLBase
- Sybase
- Text files

You define the data source, database, and variables to import using the Database Capture dialog box. This dialog box has two tabs: Databases and Variables.
Data Files

Databases

To open the Database Capture dialog box, from the menus choose:

File
  Database Capture...

Use the Databases tab to define the data source, select a database, and select a table in that database.

Data source(s). Click the box to view all data sources that have been set up for Windows using the ODBC Data Source Administrator. Select the source corresponding to the data to be imported.

Database. If no database is associated with the selected data source, SYSTAT prompts you to select a database. After selecting a database, the database name and path appear on the Databases tab.

Tables. Click the Tables box to view all tables available in the selected database. Select the table to import into SYSTAT.

Clicking OK imports the entire selected table into SYSTAT. To import a subset of variables from the selected table, use the Variables tab.
Chapter 2

Database Selection

If you have not set up the selected data source using the ODBC Data Source Administrator, the Select Database dialog prompts you for a database of the selected type to open.

![Select Database dialog box]

The list of file types contains only the selected data source type and All Files. Selecting All Files allows you to view all of the files in the current directory, but you can only select a database that corresponds to the current data source. You can, however, select a file with a nonstandard extension as long as the file is consistent with the selected data source.

Passwords. If you select a database that requires a password, SYSTAT prompts you for the password. After successfully entering the password, you can access the database.

Variable Selection

Select the variables and cases to import using the Variables tab of the Database Capture dialog box.
Initially, ODBC selects all variables in the table for importing. Move variables to the Excluded variable(s) list to omit them from the imported data; only the variables in the Selected variable(s) list appear in the final data set.

The variable list consists of the variable names found in the database. During ODBC import, SYSTAT modifies these names as needed to yield valid variables in the resulting data set. Possible changes include:

- Removal of characters that are not letters, numbers, or underscores.
- Addition of a terminal dollar sign for string variables.
- Numbering of names that would otherwise be identical.

Use the SQL query area to submit custom database information requests, including statements to select cases.

**Querying Databases Using SQL**

Use the SQL query area of the Variables tab to create and submit custom database queries. SQL is the standard language used to access information in a database. A complete discussion of SQL is beyond the scope of this manual. To gain an understanding of SQL, we recommend Date and Darwen (1997), Melton and Simon (1993), and Bowman, Emerson, and Darnovsky (1996). Here, we will only discuss case selection and ordered imports using SQL.
Initially, the SQL Query area contains the word 'where'. The WHERE clause in SQL defines cases to import. Follow WHERE with conditions specifying values or ranges of values to import using =, <>, <, >, <=, or >=. If a condition uses any character values, they must be enclosed in single quotation marks. Use AND or OR to combine conditions. For example, to import cases that have an AGE value below 30, specify

WHERE AGE<30

To further restrict the import to males only, specify

WHERE AGE<30 AND GENDER='MALE'

**Importing Business Objects**

Business Objects is business intelligence platform organization, which supports pre-defined reports, ad-hoc reporting, dashboards, extraction, transform, and load operations necessary when building data warehouse. The BusinessObjects “universe” is a “semantic layer” which sits between the business end user and the complexities of the underlying database model. End users force the “universe” to access all the databases to which they have been given permission.

Our goal would be to create an integration within the desktop version of SYSTAT which gives the end user the option of using a “BusinessObjects Universe” as a data source (similar to the other choices such as ODBC, Excel, etc.). When the “BO Universe” choice is selected within the SYSTAT environment the user is then presented with a login screen to the BusinessObjects platform and then has the ability to choose a universe to query, build a query, and then have the result set returned directly to SYSTAT without the need for cutting and pasting the data.

BO import involves translating an import request from SYSTAT to the Business Objects Web Service. Using the BO Web Service, the user can use the data base available in the server and can filter the available data according to his request.

The Windows based SYSTAT product will connect to the BusinessObjects platform through a web services interface provided by BusinessObjects. A SYSTAT user should be able to log in to the platform, define a query, execute it, and have the result set returned to SYSTAT.

The following use case illustrates how an end user, within SYSTAT, should be able to pull data from the BusinessObjects platform..
In the File menu, there is a menu item called Import Business Object to get BO Query Making Wizard. The Wizard will help to make queries in Database using Business Objects Web Service (Application Program Interface (API)s) and data can be imported to SYSTAT.

It connects to BO server and prompts for login into the server.

Upon successful login, click the Next button to get a list of databases where the user is asked to choose a Universe in the following screen. The universe layer understands how the tables in a database need to be joined and automatically creates the SQL based on the desired business data the user requests.
This is an eFashion retail Data Warehouse created on 14 Oct 1998. Based upon the universe selection, you will be able to create a query using the following QueryBuilder screen.
There are two parts: Result Objects and Query Filter. You can drag the items of Classes and Objects to the Results Objects area so that those will become variables in the SYSTAT data set.

You can drag the classes and objects required to filter according to your need to the Query Filter area of Query builder screen.
We have an option to filter the data by using Query Filter button and can pop-up the following dialog box for editing.

The filtered object appears in the Filter editor. You can select a list of operators from the list of operators. You have an option to give an exact value using constant=value. If the selected class and object has an available list of Values then you can select the operand type as Value(s) from list and then click on the button LOVs as shown below:
The list of values available for the filtered object can be seen and now the user can select from the available list of values.

Now you can observe the filtered object under the Query filter area of Query Builder screen.
It shows the number of objects in the query filter area.

If you want to save the query then he can go to the next screen given below:

User has a choice to go back to choose a universe and create a new query. If they want to see the data that is imported then they need to click the FINISH button. After the user finishes the Wizard and saves the query, data gets added to SYSTAT dataset as follows:
Now you can save this file in the SYSTAT format. User can go back to Import Business Object Menu and can get all the queries that were saved earlier and get the following screen. At this point of time you can add/modify the query.
Saving Data Files

To save a new data file, or to save an existing data file under a new name, from the menus choose:

File
  Save As…

or

File
  Save
  Data...

or

Right-click the Data editor tab for the active data file and select Save As from there.

Specify a directory and filename. SYSTAT supports long filenames with a three-letter extension, and the names can contain letters or numbers. Use underscores for spaces. Filenames cannot include periods, besides the period that separates the filename from the extension.
To save or export the data in a different format, such as a spreadsheet or database file, select the desired format from the Save as type drop-down list. When you save a file and do not specify an extension, SYSTAT automatically adds the default extension for that file type (for example, *.SYZ for SYSTAT data files). Data files are stored in a compressed format with the extension .SYZ along with variable and data information such as variable labels and value labels, and category information; because of this and also because the variable name length has increased, data file sizes are now larger than in earlier versions. To save uncompressed data files the extension is .SYZU and to save the data files of earlier version (8-11) the extension is .SYD.

From the data editor we can always save Frequency and Weight properties set to any variable. We have an option of saving Idvar and Category information about the variables. But we can not save the BY, ORDER and information about the Selected cases to a data file. If the column has been resized then that information of resized column can also be saved to a data file.

**Save Options**

When saving SYSTAT data files, you can click **Save Options** from the drop-down list of **Save** button, to specify a matrix type.

Available types include:

- **Rectangular.** Saves the raw data in a cases by variable matrix.
- **SSCP.** Saves the data as a sums-of-squares and cross-products matrix.
- **Covariance.** Saves the data as a covariance matrix.
- **Correlation.** Saves the data as a correlation matrix.
- **Dissimilarity.** Saves the data as a dissimilarity matrix.
- **Similarity.** Saves the data as a similarity matrix.
**File Comments**

You can store a comment in your data file. SYSTAT displays the comment when you use the file in order to document your data files - for example, include the source of the data, the date they were entered, the particulars of the variables, etc. The comment can be as many lines as you need. If your comment is too long to fit in one line, use commas to continue onto subsequent lines. Enclose each line in single or double quotation marks:

```
DSAVE filename / 'Edited by Joe on Jan. 10',
    'Data from Dr. Jones'
```

Also you can right-click the Data editor tab and select File Comments from it and then save the data file. You can also view this information by placing the cursor on the left topmost corner of the Data editor.

To view the file comments in the output, employ the USE command with the COMMENT option:

```
USE filename / COMMENT
```

**Export Formats**

SYSTAT can export files in the following formats:

**ASCII.** Data are written in rectangular format, like a spreadsheet, in plain text rather than binary.

**dBASE.** Because dBASE requires fixed-format numeric fields, EXPORT first checks to find the best format to use for the numbers. The fixed format also limits the size of the values that dBASE accepts. If a number is larger than the field size allowed, SYSTAT writes the largest value that fits in that field. Missing values are inserted as blanks. Dollar signs ($) are illegal characters in dBASE variable names and are replaced by underscores ( _ ).

**Excel and Lotus.** SYSTAT writes missing values as blanks, string values without trailing blanks, and variable names exactly as they appear in the SYSTAT file.

**Data Interchange Format (DIF).** Missing values are written as SYSTAT missing values. String values are written without trailing blanks.
**MINITAB.** SYSTAT exports MINITAB data files (*.MTW) for Version 13 worksheets only. When string variables are exported to MINITAB, the column numbers are suffixed with '-T'.

**STATISTICA.** SYSTAT exports STATISTICA data files with (*.STA) extension. Statistica does not have a string type, so character variables cannot be exported. When writing a Statistica file, SYSTAT will use a value of -9999 for missing.

**JMP.** Dollar signs ($) are removed from the string variables when exporting into JMP. Missing string values are replaced with a blank.

**S-PLUS.** SYSTAT exports S-PLUS data files through commands only. SYSTAT writes N.A. for missing values.

**STATA.** SYSTAT exports STATA data files with (*.DTA) extension.

**SAS.** SYSTAT exports SAS data files for the Windows platform (*.SD2) and SAS transport files (*.XPT). Transport files exported from SYSTAT must be read in SAS using PROC COPY (with the XPORT engine specified in the LIBNAME statement). Missing values are written as SAS missing values.

See *Language Reference* for the export command for each format.

**Writing SAS Variables**

When exporting data to a SAS format, variable names may be modified to satisfy SAS naming requirements. In version 9 of SAS, variable names have a maximum length of eight characters and can contain letters, numbers, and underscores. Furthermore, names must begin with a letter or an underscore. The following general rules are applied as needed to transform SYSTAT variable names to valid SAS names:

- SYSTAT removes any parentheses and $ symbols.
- All characters beyond the eighth are dropped.
- If the resulting names are not unique, SYSTAT replaces the rightmost characters with digits incrementing from 001 until all names are unique.

Although the preceding rules yield variable names that vary between SYSTAT and SAS data files, the order of the variables in both files is identical.

In addition, SYSTAT variable types need to be exported into valid SAS variable types. The correspondence between variable types in the two applications is as follows:

- Numeric SYSTAT variables appear in SAS using one of SAS’s numeric formats.
- String variables in SYSTAT are displayed in SAS as strings.
- Variables using date formats in SYSTAT use the MMDDYY format in SAS.
- Variables using time formats in SYSTAT use the TIME format in SAS.
- Variables using date/time formats in SYSTAT use the DATETIME format in SAS.

**Temporary Data Files**

Data files can either be permanent or temporary. A permanent file exists across multiple sessions, whereas a temporary file is available for the current session only. Within a session, the software handles temporary data files and permanent data files identically; you can create graphs and perform analyses using either type of file. At the end of the session, however, the temporary data files are deleted automatically.

Temporary data files serve many purposes. You can perform data exploration and transformation without overwriting your original data by saving the permanent file as a temporary file at the beginning of the session. Using the temporary file for subsequent analyses prevents accidental corruption of the (permanent) raw data. Furthermore, temporary files simplify file management. Using these files allows the software to eliminate intermediate results, preventing your folders from becoming a depository for non-required files. For example, you can compare nonparametric smoothers by appending temporary files containing the estimates for each smoother together in a single permanent file from which you can create a scatterplot overlaying the individual smoothers. Having the permanent file of all the estimates eliminates the need to maintain the individual smoother estimates in separate files. By making the files temporary, you permit the software to eliminate these files instead of having to manually delete them.

**Creating Temporary Data Files**

Temporary data files can only be created using the command language. To generate a temporary data file, use the DWORK command:

```
DWORK filename
```

DWORK can be used in place of any DSAVE command to create a temporary file. For example:

```
DSAVE FILE1
```
As an alternative, you can use DWORK:

DWORK FILE2

Similarly, in statistical modules, WORK and SAVE commands save the data files. For example, many statistical modules include a SAVE command with an option to specify the statistics saved in the file, such as:

SAVE FILE1/RESID

As an alternative, you can use WORK:

WORK FILE2/RESID

Both commands create a data file containing the residuals. However, file2 will be deleted when you exit the software.

The software saves temporary files in a working folder defined by the File Locations tab of the Options dialog box. To prevent overwriting of a permanent data file with a temporary file, we recommend that the working folder differ from the folders assigned for opening and saving data files.

**Accessing Temporary Data Files**

You can open temporary files using the Open dialog box or the USE command. When using the latter method, you can specify the full path and the name of the file:

USE C:\temp\myfile

or just the file name:

USE MYFILE

The software looks for the first data file it encounters in the folder for input data files. If the file cannot be found in that folder, the search progresses to the folder for temporary data files. If the file still cannot be found, the search concludes in the folder for output data files. Because files in different folders may share a common name, omitting the path may result in opening a file from the wrong folder. For example, you may have the permanent data file MYFILE in the Data folder and a temporary data file named MYFILE in the C:\temp folder. If the Data folder corresponds to the folder assigned to input data files, submitting:

USE MYFILE
opens the permanent file because it is the first file having that name found in the search.

To immediately access a specific file in the working folder, precede the filename with the reserved token &WORK.

```plaintext
USE &WORK\filename
```

During command processing, the software automatically replaces &WORK with the path corresponding to the folder for working files, eliminating the need to search through multiple folders. In the example above, assuming the folder for temporary files corresponds to C:\temp, submit:

```plaintext
USE &WORK\myfile
```

to open the temporary file. Of course, you can always enter the full path to the file instead of using the &WORK token.

**Merging and Appending Data Files**

There are two ways to concatenate files: horizontally (side by side, joining different variables for the same cases) i.e., to append variables and vertically (end to end, adding more cases for the same variables) i.e., to append cases.

Append Variables performs horizontal concatenation:
Append Cases performs vertical concatenation:

You can merge or append only two files at one time. If you have more than two files, merge them successively, two at a time, until they are all part of one file.

**Append Variables**

Append Variables joins two SYSTAT files horizontally (side by side). SYSTAT joins the cases from the two files in order, matching the first case from each file, the second case from each file, and so on.

To open the Append Variables dialog box, from the menus choose:

Data
Merge Files
Append Variables...
For each file, click the File browse button to specify each of the files to be merged. If one file has more observations than the other, SYSTAT assigns missing values to the variables from the shorter file for all of the unmatched observations. If the same variable name appears in both files, SYSTAT names the variables trailed by the name of the files with an underscore, for example var1_file1 and var1_file2. Also, if the filenames are same and are at different locations then along with the filename it is also trailed by 1 and 2 to differentiate the two variables, for example var1_file1_1 and var1_file1_2.

Optionally, you can select the variables from each file to be included in the merged data file. If no variables are selected, all variables from both files are included.

Save file. Specifies an output file for the merged data. If you do not select this option, SYSTAT saves the data in a temporary file.

You can specify one or more key (index) variables by clicking the Key Selection tab.
Merging by Key Variables

You can merge files using one or more key (index) variables. In a drug study, for example, you might place demographic data for the patients in one file and laboratory test results in another file. If both files contain the patients’ ID numbers, you could use the ID as the key variable to link each patient’s demographics with their test results.

Click a variable name to select the variable and click the Add button to use the variable as a key. SYSTAT matches the cases that have the same values for the key variable(s) and joins them in a case in the new file. If there are values for the key variable(s) in one file and not in the other, the merged file records missing values for the variable whose file did not have values.
In the following example, we start with files \( A \) and \( B \) and create file \( C \):

<table>
<thead>
<tr>
<th>Key</th>
<th>( X )</th>
<th>Key</th>
<th>( Y )</th>
<th>Key</th>
<th>( X )</th>
<th>( Y )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>1</td>
<td>100</td>
<td>1</td>
<td>10</td>
<td>100</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
<td>3</td>
<td>300</td>
<td>2</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td></td>
<td>3</td>
<td></td>
<td>300</td>
</tr>
</tbody>
</table>

**Generating Replicates of a Record**

One key variable can have many occurrences of a value that appears only once in the other file. For example, you may need to join information about a mother to the data records for each of her children. For this, SYSTAT replicates the values from the \( MOMS \) file. For example, using the \( FAMILY ID \) as the key,

```
MERGE  KIDS  MOMS  /  FAMILY
DSAVE  PAIRS
```

yields the file represented in the final three columns in the following table:

<table>
<thead>
<tr>
<th>File KIDS</th>
<th>File MOMS</th>
<th>File PAIRS</th>
</tr>
</thead>
<tbody>
<tr>
<td>FAMILY X</td>
<td>FAMILY Y</td>
<td>FAMILY X</td>
</tr>
<tr>
<td>1 10</td>
<td>1 100</td>
<td>1 10</td>
</tr>
<tr>
<td>1 1</td>
<td>2 140</td>
<td>1 1</td>
</tr>
<tr>
<td>1 12</td>
<td></td>
<td>1 12</td>
</tr>
<tr>
<td>2 6</td>
<td>2 6</td>
<td>2 6</td>
</tr>
<tr>
<td>2 9</td>
<td>2 9</td>
<td>2 9</td>
</tr>
</tbody>
</table>

**Append Cases**

Append Cases joins two SYSTAT files vertically (end to end). SYSTAT places cases from the second file you name below those from the first.

To open the Append Cases dialog box, from the menus choose:

Data
Merge Files
Append Cases...
Click the File browse buttons to select the files to be appended. One of the following three options must be specified:

- **Match.** Both files must contain the same variables in the same order. If not, SYSTAT displays an error message.
- **Union.** Includes all variables from both input files.
- **Intersect.** Includes only those variables that appear in both files.

For example, if *file1* has variables *a*, *b*, and *d* and *file2* has variables *a*, *b*, and *c*:

- **Match** displays an error message because the two files do not contain the same variables in the same order.
- **Union** writes a new file with variables *a*, *b*, *c*, and *d*. Values of *c* are missing value codes for *file1*, and values of *d* are missing value codes for *file2*.
- **Intersect** includes only variables *a* and *b* in the new file.

**Save file.** Specifies an output file for the appended data. If you do not select this option, SYSTAT saves the data in a temporary file.
Chapter 2

Using Commands

Opening files. You can open an existing SYSTAT data file with the USE command:

```
USE filename
```

For all other types of files, including spreadsheet, database, and ASCII text files, use the IMPORT command:

```
IMPORT filename / TYPE=filetype
```

where `filename` is the name of the input data file that was written by one of the specified applications, and `filetype` is the type of file you are importing (ASCII, SPSS, SAS, SASTRANSPORT, Lotus, Lotus2, Excel, SigmaPlot, dBase, DIF, S-PLUS, MINITAB, STATISTICA, STATA, JMP, StatView or ArcView).

For ODBC import, use the IMPORT command without a filename:

```
IMPORT / TYPE=ODBC CONNECT='connect string'
    TABLE='table name' VARIABLES='varlist'
    SQL='statement'
```

For ODBC import, use the IMPORT command with a filename:

```
IMPORT filename / TYPE=EXCEL, SHEET=1 VARIABLES='varlist'
    SQL='statement' ROW1 = VARNAMES
    or DATA DELIMITERS = {TAB, SPACE,
                         COMMA, SEMICOLON, char}
```

The options CONNECT, VARIABLES, and SQL identify the database and table to access, the variables to import, and optional SQL clauses to apply during import.

It is recommended that you specify a file to save after IMPORT:

```
IMPORT test / TYPE=ASCII
DSAVE NEWTEST
```

Saving files. You can save a SYSTAT data file with the DSAVE and DWORK commands:

```
DSAVE filename / TYPE=filetype
ESAVE filename / TYPE=filetype
```

and

```
DWORK filename / TYPE=filetype
EWORK filename / TYPE=filetype
```
Data Files

DSAVE creates a permanent data file. DWORK creates a temporary data file that is deleted at the end of the session. For *filetype*, specify RECTANGULAR (default), SSCP, COVARIANCE, CORRELATION, DISSIMILARITY, or SIMILARITY.

To save data in a different format, such as Excel or dBASE, use EXPORT:

```
EXPORT filename / TYPE=filetype
```

where *filename* is the name of the exported data file you are creating, and *filetype* is the type of file you are exporting (ASCII, SAS, SATRANSPORT, Lotus, Lotus2, Excel, dBase, DIF, SPSS, MTW (for MINITAB), STA (for STATISTICA), DTA (for STATA), JMP, or SVD (for StatView)). SYSTAT automatically assigns the appropriate extension to the filename depending on the specified file type.

**Concatenating files.** To append variables of files, the command syntax is:

```
MERGE filename1 (varlist1) filename2 (varlist2) / keyvarlist
```

where *varlist1* and *varlist2* are optional--use them to select a subset of the variables in any order you want. The list of key variables is also optional.

To append cases of files:

```
APPEND filename1 filename2 / MATCH or UNION or INTERSECTION
```

To save the appended files permanently, use DSAVE after APPEND.

**Spaces in filenames or paths.** Filenames or paths to files that include a space must be enclosed in quotation marks for all commands that involve manipulating files. For example, to open *myfile* from the \Program Files\Systat\ folder on the C: drive, specify:

```
USE "C:\Program Files\Systat\myfile"
```

**Examples**

**Example 1**

**Selecting a Subset of the Variables**

You can select a subset and reorder variables when you merge:

```
MERGE MOE(X Y Z) JOE(C A B)
MERGE MOE(Z) JOE
MERGE MOE JOE(A B)
```
The first example merges two files by appending variables, extracting variables $x$, $y$, and $z$ from $MOE$ and variables $c$, $a$, and $b$ from $JOE$. The second selects variable $z$ from $MOE$ and all the variables in $JOE$. The third selects all the variables from $MOE$ and $a$ and $b$ from $JOE$.

**Example 2**  
**Side-By-Side Merge**

This example demonstrates merging two files by appending variables. One file, $NAME$, contains the names of men who have been presidential candidates in the variable $NAMES$. The second file, $PARTY$, contains their party affiliations in the variable $PARTYS$:

<table>
<thead>
<tr>
<th>$NAME$ data file</th>
<th>$PARTY$ data file</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAMES</td>
<td>PARTY$</td>
</tr>
<tr>
<td>Eisenhower</td>
<td>Republican</td>
</tr>
<tr>
<td>Stevenson</td>
<td>Democrat</td>
</tr>
<tr>
<td>Kennedy</td>
<td>Democrat</td>
</tr>
<tr>
<td>Goldwater</td>
<td>Republican</td>
</tr>
<tr>
<td>Johnson</td>
<td>Democrat</td>
</tr>
<tr>
<td>Humphrey</td>
<td>Democrat</td>
</tr>
<tr>
<td>McGovern</td>
<td>Democrat</td>
</tr>
<tr>
<td>Nixon</td>
<td>Republican</td>
</tr>
<tr>
<td>Ford</td>
<td>Republican</td>
</tr>
<tr>
<td>Carter</td>
<td>Democrat</td>
</tr>
<tr>
<td>Reagan</td>
<td>Republican</td>
</tr>
<tr>
<td>Bush</td>
<td>Republican</td>
</tr>
<tr>
<td>Clinton</td>
<td>Democrat</td>
</tr>
</tbody>
</table>

A one-to-one correspondence exists between the cases in the two files. The first case from the $NAME$ file corresponds with the first case in the $PARTY$ file. To merge these files:

```
MERGE NAME PARTY
DSAVE CANDIDAT
LIST
```
Since no variables are specified, all variables from both files are included in the merged file.

<table>
<thead>
<tr>
<th>Case</th>
<th>NAMES$</th>
<th>PARTY$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Eisenhower</td>
<td>Republican</td>
</tr>
<tr>
<td>2</td>
<td>Stevenson</td>
<td>Democrat</td>
</tr>
<tr>
<td>3</td>
<td>Kennedy</td>
<td>Democrat</td>
</tr>
<tr>
<td>4</td>
<td>Goldwater</td>
<td>Republican</td>
</tr>
<tr>
<td>5</td>
<td>Johnson</td>
<td>Democrat</td>
</tr>
<tr>
<td>6</td>
<td>Humphrey</td>
<td>Democrat</td>
</tr>
<tr>
<td>7</td>
<td>McGovern</td>
<td>Democrat</td>
</tr>
<tr>
<td>8</td>
<td>Nixon</td>
<td>Republican</td>
</tr>
<tr>
<td>9</td>
<td>Ford</td>
<td>Republican</td>
</tr>
<tr>
<td>10</td>
<td>Carter</td>
<td>Democrat</td>
</tr>
<tr>
<td>11</td>
<td>Reagan</td>
<td>Republican</td>
</tr>
<tr>
<td>12</td>
<td>Bush</td>
<td>Republican</td>
</tr>
<tr>
<td>13</td>
<td>Clinton</td>
<td>Democrat</td>
</tr>
</tbody>
</table>

**Example 3**

**Merging with a Key Variable**

This example merges the files CANDIDAT and ELECTION by the variable NAMES$. CANDIDAT was created by merging NAME and PARTY files. The data in the ELECTION file are shown below.

<table>
<thead>
<tr>
<th>NAMES$</th>
<th>LOSERS</th>
<th>YEAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eisenhower</td>
<td>Stevenson</td>
<td>1952</td>
</tr>
<tr>
<td>Eisenhower</td>
<td>Stevenson</td>
<td>1956</td>
</tr>
<tr>
<td>Kennedy</td>
<td>Nixon</td>
<td>1960</td>
</tr>
<tr>
<td>Johnson</td>
<td>Goldwater</td>
<td>1964</td>
</tr>
<tr>
<td>Nixon</td>
<td>Humphrey</td>
<td>1968</td>
</tr>
<tr>
<td>Nixon</td>
<td>McGovern</td>
<td>1972</td>
</tr>
<tr>
<td>Carter</td>
<td>Ford</td>
<td>1976</td>
</tr>
<tr>
<td>Reagan</td>
<td>Carter</td>
<td>1980</td>
</tr>
<tr>
<td>Reagan</td>
<td>Mondale</td>
<td>1984</td>
</tr>
<tr>
<td>Bush</td>
<td>Dukakis</td>
<td>1988</td>
</tr>
<tr>
<td>Clinton</td>
<td>Bush</td>
<td>1992</td>
</tr>
</tbody>
</table>

To merge the two files, specify NAMES$ as the key variable.

```
MERGE ELECTION CANDIDAT/NAMES$
DSAVE MERGFILE
LIST
```
Chapter 2

The resulting file includes:

<table>
<thead>
<tr>
<th>Case</th>
<th>NAME$</th>
<th>PARTY$</th>
<th>LOSER$</th>
<th>YEAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Bush</td>
<td>Republican</td>
<td>Dukakis</td>
<td>1988.000000</td>
</tr>
<tr>
<td>2</td>
<td>Carter</td>
<td>Democrat</td>
<td>Ford</td>
<td>1976.000000</td>
</tr>
<tr>
<td>3</td>
<td>Clinton</td>
<td>Democrat</td>
<td>Bush</td>
<td>1992.000000</td>
</tr>
<tr>
<td>4</td>
<td>Eisenhower</td>
<td>Republican</td>
<td>Stevenson</td>
<td>1952.000000</td>
</tr>
<tr>
<td>5</td>
<td>Eisenhower</td>
<td>Republican</td>
<td>Stevenson</td>
<td>1956.000000</td>
</tr>
<tr>
<td>6</td>
<td>Ford</td>
<td>Republican</td>
<td></td>
<td>.</td>
</tr>
<tr>
<td>7</td>
<td>Goldwater</td>
<td>Republican</td>
<td></td>
<td>.</td>
</tr>
<tr>
<td>8</td>
<td>Humphrey</td>
<td>Democrat</td>
<td></td>
<td>.</td>
</tr>
<tr>
<td>9</td>
<td>Johnson</td>
<td>Democrat</td>
<td>Goldwater</td>
<td>1964.000000</td>
</tr>
<tr>
<td>10</td>
<td>Kennedy</td>
<td>Democrat</td>
<td>Nixon</td>
<td>1960.000000</td>
</tr>
<tr>
<td>11</td>
<td>McGovern</td>
<td>Democrat</td>
<td></td>
<td>.</td>
</tr>
<tr>
<td>12</td>
<td>Nixon</td>
<td>Republican</td>
<td>Humphrey</td>
<td>1968.000000</td>
</tr>
<tr>
<td>13</td>
<td>Nixon</td>
<td>Republican</td>
<td>McGovern</td>
<td>1972.000000</td>
</tr>
<tr>
<td>14</td>
<td>Reagan</td>
<td>Republican</td>
<td>Carter</td>
<td>1980.000000</td>
</tr>
<tr>
<td>15</td>
<td>Reagan</td>
<td>Republican</td>
<td>Mondale</td>
<td>1984.000000</td>
</tr>
<tr>
<td>16</td>
<td>Stevenson</td>
<td>Democrat</td>
<td></td>
<td>.</td>
</tr>
</tbody>
</table>

The missing values occur where entries in the candidate file have no matching value for NAME$ in the election data file (for example, Ford, Goldwater, Humphrey, etc., did not win). The election file has three names that have more than one entry: Eisenhower, Nixon, and Reagan. In these cases, SYSTAT replicates the corresponding values from the candidate file.

Note: If you select a subset of variables while merging, you need not include the key variable(s) in both subsets (of course, they must be in both files). The following, for example, is valid:

```
MERGE INDOOR(TIME,LOC,CO2) OUTDOOR(NOX)/TIME,LOC
```

Example 4
End-to-End Append

The following are two SYSTAT files, named MEN and WOMEN:

<table>
<thead>
<tr>
<th>MEN data file</th>
<th>WOMEN data file</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SEX$</strong></td>
<td><strong>AGE</strong></td>
</tr>
<tr>
<td>Male</td>
<td>18</td>
</tr>
<tr>
<td>Male</td>
<td>35</td>
</tr>
<tr>
<td>Male</td>
<td>24</td>
</tr>
<tr>
<td>Male</td>
<td>20</td>
</tr>
<tr>
<td>Female</td>
<td>23</td>
</tr>
<tr>
<td>Female</td>
<td>40</td>
</tr>
<tr>
<td>Female</td>
<td>40</td>
</tr>
<tr>
<td>Female</td>
<td>31</td>
</tr>
</tbody>
</table>
To append them end to end:

```
APPEND WOMEN MEN
DSAVE SEXES
LIST
```

SYSTAT places the cases from *WOMEN* before those from *MEN* because *WOMEN* are listed first in the APPEND command.

<table>
<thead>
<tr>
<th>Case</th>
<th>SEX$</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Female</td>
<td>23.000000</td>
</tr>
<tr>
<td>2</td>
<td>Female</td>
<td>40.000000</td>
</tr>
<tr>
<td>3</td>
<td>Female</td>
<td>40.000000</td>
</tr>
<tr>
<td>4</td>
<td>Female</td>
<td>31.000000</td>
</tr>
<tr>
<td>5</td>
<td>Male</td>
<td>18.000000</td>
</tr>
<tr>
<td>6</td>
<td>Male</td>
<td>35.000000</td>
</tr>
<tr>
<td>7</td>
<td>Male</td>
<td>24.000000</td>
</tr>
<tr>
<td>8</td>
<td>Male</td>
<td>20.000000</td>
</tr>
</tbody>
</table>

**References**


The Data editor is used for entering, editing, and saving data. Entering data is a straightforward process—simply type the data. SYSTAT assigns default variable names (which can be changed if desired) to each column.

Editing data includes changing variable names or attributes, adding and deleting cases or variables, moving variables or cases, and correcting data errors. In addition, you can locate specific variables, find cases that meet specific conditions, or identify each case in the Data editor using a variable’s values.

There are two tabs, namely Data tab and Variable tab. Apart from these we have an icon for File comments where you can click on this icon to view or edit the file comments.

**Data Editor**

Viewing, entering, and editing data occurs in the Data editor. You can also run transformations and view the results, select subsets of cases, and transpose, append, and merge data files as described in this and the following chapters.
Chapter 3

To display the Data Editor, choose Data Editor from the View menu or click the Data Editor tab in the Viewspace. Besides Data and Variable tabs next to file comments, we have information about number of cases (N) and number of variables (NVAR). Number of selected cases (NSEL) will appear when the cases are selected in the datafile.

Data File Structure

SYSTAT uses data organized in rows and columns. The rows are cases and the columns are variables. A case contains information for one unit of analysis, such as a person, an animal, a business, or a jet engine. Variables are the information collected for each case, such as age, body weight, profits, or fuel consumption.

For example, a portion of the data from the file OURWORLD is shown. Each row (case) has data for one of 15 countries, and the columns (variables) include the name of the country, population in 1990, gross domestic product per capita, years of life expectancy estimates for females, literacy, type of country, and the number of McDonald’s restaurants.
When data are arranged in rows and columns (as in these data), and then stored in a
file, we call the file a cases-by-variables or rectangular data file.

### Variable Editor

The Data editor allows you to edit data values directly in the grid. The Variable editor
allows you to edit the properties of variables. The entered name will be considered only
after checking the validity of the name and presence of another variable with the same
name.

- It has one row corresponding to each variable, and the row includes all the
  properties that are in the Variable Properties dialog. With it, you can set any of the
  properties for any variable with a single click of the mouse.
- The Variable editor also has a pane where you can view and set the processing
  conditions in effect for the current data set. For details on processing conditions,
  refer to the section “Processing Conditions in Effect” on page 49.

To view the Variable editor, from the menus, choose:

View
  Variable Editor...
Name. You can assign a name to every variable. Variable names are used to specify analyses and to label output. Variable names can contain up to 256 letters or numbers and must begin with a letter or an underscore. Names of character or string variables must end with a dollar sign ($), which counts as a character. SYSTAT defines all variables whose names do not end in $ as numeric. Use the underscore character ( _ ) to indicate a space within a name. The only other characters that can be used in a variable name are parentheses, which denote subscripted variables.

Variable names, unlike character values, are not case sensitive. If you type govern, SYSTAT automatically uses GOVERN.

Label. A variable label is a string or text associated with the variable. It is an alias for the variable name. It can be up to 256 characters in length, and in contrast to the variable name, can contain spaces and special characters, and can start with any character. This will not be displayed unless changed to something other than the name.

You can define a variable label using either the Variable editor, the Variable Properties dialog box, or the VARLAB command. If an analysis is performed using a given variable, its variable label is displayed instead of its name in the Output editor. You can control the display of variable labels in the output using the Edit: Options dialog or the
VDISPLAY command. To view the variable labels as a tool tip, pause the mouse pointer on it. Any defined variable labels will be saved in the data file.

**Value label.** Value labels are the labels given to the values of a variable. You can define value labels using either the Variable editor, the Value Labels dialog, or the LABEL command. To view value labels in the Data editor, right-click on a variable name and select View Value Labels. By default, value labels will appear in the output including in graphs. You can control the display of value labels in the output using the Edit: Options dialog or the LDISPLAY command. To view the value label as a tool tip, pause the mouse pointer on it. Any defined value labels will be saved in the data file.

**Type.** Choose the variable type. SYSTAT accepts numbers and characters as data.

- **Numeric.** Use for variables that contain numbers only. Numeric variables can contain up to 23 digits. Use a minus sign for negative numbers. You can also use exponential notation for very large or very small numbers. For example, $1.5E4$ and $1.5E-4$ appear as:

\[
\begin{align*}
1.5E4 &= 1.5 \times 10^4 = 15000 \\
1.5E-4 &= 1.5 \times 10^{-4} = 0.0015
\end{align*}
\]

- **String.** String or character variables contain text information. A string variable can contain up to 256 characters and can contain any common typewriter characters. Specify the number of characters the variable will hold. When storing values of string variables, SYSTAT differentiates between upper and lower case—"europe" is not the same as "EUROPE" or "Europe." If you include numbers within string values, SYSTAT treats them as text; you cannot perform numerical analyses on them. Some valid values are:

```
male        New York  $)#($&%^/#@
female      Chicago   !@#$%^&*)?($
```

**Display.** For numeric variables, you can choose among several display options:

- **Normal.** Displays variable values in standard decimal notation—for example, 123.45. Specify the number of decimal places to display.

- **Exponential notation.** Displays variable values in exponential notation. Specify the number of decimal places to display.

- **Date and time.** Select the desired date and time format from the list. The number of letters indicates the number of characters displayed. Two m’s (mm) represent months as two digits; for example, September would appear as 09. Three m’s (mmm) represent month abbreviations, so September would be “Sep”. To display the entire month name, use four m’s (mmmm). Use the same rules for days of the
week, using \( d \)’s instead of \( m \)’s. For instance, \( ddd \) displays “Tue”. Use ‘yy’ to display the last two digits of the year, ‘yyyy’ to display all the digits of the year, ‘hh’ to display time in hours, ‘mm’ in minutes, and ‘ss’ in seconds. The ‘hh’ display format uses 12 hour scale. If you want to display the 24 hour scale, use the upper case, i.e., ‘HH’. You can select any date and format from the dropdown list. You can also use any custom format by simply double-clicking and typing the desired format. For example, you can specify the following formats:

\[
\begin{align*}
&\text{dd:MMMM:yyyy-HH:mm:ss} \\
&\text{MMMM dd,yyyy}
\end{align*}
\]

Display options are specific to variables, not cells. For any given variable, all cells have the same format.

**Width.** Specify the maximum field width of the variable, i.e., the total number of characters to be displayed in the Data editor. For string variables, you can specify up to 256 characters. For numeric variables you can specify up to 23 characters.

**Decimal.** Specify the number of decimal places to be displayed in the Data editor. This option is applicable for numeric variables when the display option is either Normal or Exponential notation. You can specify up to 14 digits. The number of decimal places should be less than the corresponding field width. The default is the number specified in the Output tab of the Edit:Options dialog box.

**Category.** Use to declare a variable as categorical. SYSTAT treats the values of such variables, whether they are numeric or string, as discrete categories. Categorical variable information will be saved in the data file by default. You can prevent this by unchecking Save category variable information to data file in the Data tab of the Edit:Options dialog box.

**Format.** Specify the DateTime format to be displayed in the Data editor. This option is applicable for numeric variables when the display option is DateTime. You can specify various formats for the date and time from the list of options in the combo box.

**Comments.** A comment is a string or text associated with the variable. You can define a comment to each variable using either the Variable editor, the Variable Properties dialog box, or the DEFVAR command in the command space.

See “Data Edit Bar” on page 59 for more about navigation.
Pasting Variable Properties

The Variable editor not only allows you to edit the variable properties manually, but also allows you to edit them through the right-click options like copy and paste.

So, you can copy properties of one or more variables and paste them for the desired variables. You can copy properties like variable label, value labels, width, decimals, display format, date and time format, and variable comments. You can paste a property of one variable to the same property of another variable.

You can also change the properties of variables and create new ones through the shortcut menu of rows in the Variable editor. The functioning of these options is similar to that of the right-click menu options for variable names in the Data editor. For more information on these options, refer to the section “Right-Click Editing” on page 66.

Processing Conditions in Effect

The Variable editor in SYSTAT contains a processing conditions pane, which displays the processing conditions in effect for the current data set. The parameters in the processing conditions are case selection conditions, grouping variable specifications, weight, frequency, Id, category, and order variable specifications. You can change the parameter values in the processing condition by double clicking on each desired parameter, and making the necessary changes in the dialog that opens.
Variable Properties Dialog Box

To set the properties of individual variables, use the Variable Properties dialog box. To open the Variable Properties dialog box, from the menus choose:

Data
Variable Properties..
Notice that you can open the Variable Properties dialog box from the menus only when the Data editor tab is active.

**Variable name.** You can assign a name to every variable. Variable names are used to specify analyses and to label output. Variable names can contain up to 256 letters or numbers and must begin with a letter. Names of character or string variables must end with a dollar sign ($), which counts as a character. SYSTAT defines all variables whose names do not end in $ as numeric. Use the underscore character (_) to indicate a space within a name. The only other characters that can be used in a variable name are parantheses, which denote subscripted variables.

Variable names, unlike character values, are not case sensitive. If you type govern, SYSTAT automatically uses GOVERN.
Variable label. A variable label is a string or text associated with the variable. It is an alias for the variable name. It can be up to 256 characters in length, and in contrast to the variable name, can contain spaces and special characters, and can start with any character.

You can define a variable label using either the Variable editor, the Variable Properties dialog box, or the VARLAB command. If an analysis is performed using a given variable, its variable label is displayed instead of its name in the Output editor. You can control the display of variable labels in the output using the Edit : Options dialog or the VDISPLAY command. To view the variable labels as tool tips, pause the mouse pointer on it. Any defined variable labels will be saved in the data file.

Display options

- **Characters.** Specify the maximum field width of the variable, i.e., the total number of characters to be displayed in the Data editor. For string variables, you can specify up to 256 characters. For numeric variables you can specify up to 23 characters.

Save changes while navigating. You can navigate from one variable to the next while setting the properties. To navigate, click on the navigation buttons at the bottom of the dialog box. From a given variable, you can go to the next, previous, first or last variable in the data file with a single click of the mouse. By default, changes to properties of any variable are saved while navigating. If you do not want to save the changes, you can uncheck the Save changes while navigating option and press OK to save the changes.

To Specify Variable Properties

In the Variable editor

- From the menus choose:
  View
  Variable editor...

  or

- Click on the Variable tab of the active data file.

  or

- Double-click on a variable name in the Data editor.

  or
- Right-click on the Data tab of the active data file and select Data/Variable editor.
  
or
- CTRL + SPACEBAR to go to the Variable editor.

In the Variable Properties Dialog Box

- From the menus choose:
  
  Data
  Variable Properties...

  
or
- Right-click on a variable name in the Data editor and select Variable Properties.

File properties

To display the list of variable(s) in the file along with properties, choose from the menus:

Utilities
  File Information
    Dictionary...

To display only the variable names, select Names.

Subscripted Variables

An easy way to specify a subset of variables is to use subscripted variables. For example, if you have 10 questions, you could simply name them $Q(1)$, $Q(2)$, ..., $Q(10)$ or, for string variables, $Q$(1), $Q$(2), ..., $Q$(10). You can then refer to the range of variables using subscript notation, such as:

```
cSTATISTICS Q(1 .. 10)
```

to get descriptive statistics for the responses to all 10 questions. SYSTAT will perform analyses on all of the variables from $Q(1)$ to $Q(10)$, regardless of the order in which they appear in the file.

Subscripted names can be used for numeric or string variables. The total length of the variable name can be up to 256 characters, including the parentheses (and the dollar sign for character variables).
**Grouping Variables**

A grouping variable contains a value that identifies group membership for each case. The values of these variables are used to separate the cases into groups (or cells) in subsequent analyses and graphs. In the OURWORLD data file, GROUP$ is a string grouping variable, and GROUP is a numeric grouping variable that gives each variable the following identification: 1 for Europe, 2 for Islamic, and 3 for New World. A numeric grouping variable named SEX might contain the number 1 for males and 2 for females, while a string grouping variable such as SEX$ could have the values Male and Female.

**Variable Comments**

The Variable editor contains a column to record information about each variable. There is no limit to the amount of information you can record in a variable's comments. You can also record comments using the Variable Properties dialog box.

The comments of a variable can be viewed as a tool tip in the Data/Variable editor. To view, pause the mouse pointer on the variable name.

**Missing Data**

Some cases may be missing data for a particular variable—for example, a subject might not have a middle name, or a state might have failed to report its total sales. In the Data editor, missing numeric values are indicated by a period, and missing string values are represented by an empty cell.

Arithmetic that involves missing values propagates missing values. If you add, subtract, multiply, or divide when data are missing, the result is missing. If you sort your cases using a variable with missing values, the cases with values missing on the sort variable are listed first. If you specify conditions and a value is missing, SYSTAT sets the result to missing. For example, if you specify:

```
IF age > 21 THEN LET age$='Adult'
```

and AGE is missing, the value of AGES$ is set to missing. To perform an analysis on only those cases with no values missing, use SELECT COMPLETE() prior to the analysis.
Note: If you are entering data in an ASCII text file, enter a period (.) to flag the position where a numeric value is missing. Where character data are missing in an ASCII text file, enter a blank space surrounded by single or double quotation marks.

**Data Entry**

Entering data in the Data editor is a straightforward process. Simply type the data, and after typing each value, press Enter.

SYSTAT denotes empty columns with the name 1. Upon the entry of a value in a column, SYSTAT assigns the name VAR (n) to that variable, where n corresponds to the variable number. SYSTAT fills all empty columns to the left of this variable with missing values and assigns default variable names. If the entered value contains characters, SYSTAT appends a $ to the variable name.

Note that, SYSTAT determines the decimal and digit grouping symbols for numbers from the current settings in the Regional and Language Options dialog of the Windows Control Panel. You can enter numbers in the Data editor using the specified decimal and digit grouping symbols. They will be displayed with the specified digit grouping.
The active (current) cell is highlighted with a thin black border. You can enter data by case (one row at a time) or by variable (one column at a time). To specify whether the active cell moves across (for case entry) or down (for variable entry) when you press Enter, choose Options from the Edit menu, click the Data tab, and select the desired behavior.

Data values are not recorded until you press Enter or select another cell. To open new data, press CTRL + N or from the menus choose:

File
    New
        Data...

Note that there is no limit to the number of variables or cases your data file can contain. However, the amount of memory available on your system may impose a practical limit.

**Editing Data**

You can edit data to correct errors, change variable names, or add, delete, or move variables or cases. SYSTAT provides a number of tools to help you edit your data. You can locate specific variables, find cases that meet specific conditions, or specify an ID variable to identify each case in the Data editor.

*Tip:* Any changes you make to the data file are not permanent unless you save the data file.

**Data Editor Navigation**

You can use either the mouse or the keyboard to move around in the Data editor and to select cells, cases, or variables.

- Click the mouse in any cell to select the cell.
- Use the scroll bars on the bottom and right side of the editor to scroll through the data to see more cases and variables. Click the scroll arrows to move one case or variable at a time, or drag the icon to move to a new location in the data.
- Using the keyboard, you can move in all directions using the arrow keys. You can also move the active cell using the Enter and Tab keys alone or in combination with the Shift key.
Selecting Cells

You can select cells using either the mouse or the keyboard.

Using the Mouse

- Drag the mouse to choose a range of adjacent cells, variables, or cases.
- Click on the case number at the left side of a row or the variable name at the top of a column to select the entire row or column.
- Hold down the Shift key as you click on any cell, case number, or variable name. The resulting selection depends on the initial state of the Data editor and the location of the mouse click.
- Hold down the CTRL key as you click on any cell, case number, or variable name. This selects the data which are not continuous.
**Chapter 3**

- **Using the Keyboard**
  - Hold down the **SHIFT** key and use the arrow keys to extend the selection from the current cell to the target cell defined by that arrow key. For example, **SHIFT** and the left-arrow key selects the active cell and the cell immediately to the left.
  - You can also use **SHIFT** with some shortcut key combinations; for example, **CTRL+SHIFT+END** extends the selection to the bottom right corner.

- **Correcting Data Values**
  - You can easily replace data values in the Data editor. Simply select the cell, edit the value, and move to another cell using the mouse or keyboard.
    - Editing a cell often involves changing portions of an entry instead of replacing the entire value. You can edit in a cell.
  - **In-place editing.** Double-click the cell whose value you wish to edit. Double-clicking within the cell value places the insertion point at begin. This insertion point can be moved using the following keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Moves the insertion point:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left arrow</td>
<td>One character left.</td>
</tr>
<tr>
<td>Right arrow</td>
<td>One character right.</td>
</tr>
<tr>
<td>Home</td>
<td>Beginning of value.</td>
</tr>
<tr>
<td>End</td>
<td>End of value.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Initial state</th>
<th>Shift-Click Target</th>
<th>Resulting selection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Selected cell</td>
<td>Cell</td>
<td>Range of cells between row I and target cell row, and between column J and target cell column.</td>
</tr>
<tr>
<td>(active cell in row I and column J)</td>
<td>Case number</td>
<td>Target row.</td>
</tr>
<tr>
<td></td>
<td>Variable name</td>
<td>Target column.</td>
</tr>
<tr>
<td>Selected row (I)</td>
<td>Cell</td>
<td>Range of cells between row I and target cell row, and between first column and target cell column.</td>
</tr>
<tr>
<td></td>
<td>Case number</td>
<td>Rows from row I to the target row.</td>
</tr>
<tr>
<td></td>
<td>Variable name</td>
<td>Columns from column I to the target column.</td>
</tr>
<tr>
<td>Selected Column (J)</td>
<td>Cell</td>
<td>Range of cells between first row and target cell row, and between column I and target cell column.</td>
</tr>
<tr>
<td></td>
<td>Case number</td>
<td>Rows from row 1 to the target row.</td>
</tr>
<tr>
<td></td>
<td>Variable name</td>
<td>Columns from column J to the target column.</td>
</tr>
</tbody>
</table>
If you select a portion of the value by dragging the mouse over it, SYSTAT deletes the selected portion before inserting the typed values. Note that, in the display format, SYSTAT may not show the number completely but when we double-click on a cell, the number is completely shown in the cell. For example, suppose a cell has an entry of 123.3654329. If the display format for that variable contains three decimal places only, the number appears as 123.365 in the cell but when we double click SYSTAT shows 123.3654329, the complete value.

**Single-click replacement.** To replace an entire cell entry, single-click the cell and type the new entry. In contrast to in-place editing, the replacement value is not constrained by the display format. However, shifting the active cell to another location formats the replacement value in accordance with its column. For example, suppose a variable has a display format of eight characters with two decimal places. We want to replace the entry 111.11 with 222.2222, so we single-click the cell and type the new entry. Moving to another cell formats the entered value to 222.22. To edit the final two decimal places, we must use the in-place editing.

Note: No changes to a cell entry become permanent until another cell becomes active. Use the Escape key (Esc) before moving to another cell to cancel changes and revert a cell entry to its original value.

**Editing Dates and Times**

To change a date or time, click the cell and type the new date value. For example, suppose we want to change the date 06/10/2003 to 08/14/2003. Click the cell, and type the new value 08/14/2003.

**Data Edit Bar**

Data edit bar is a part of Data Editor. It is a composite tool bar that allows navigation and editing of the data in the active file. It has three parts.

- The first part has a drop down list box which lists the variables of the file currently seen. The user can use this to navigate the variables in the file.
- The second part gives the information about the cases. It has a spin button which can be used to navigate through the cases. Clicking the spin button navigates consecutive cases. One may also jump directly to the desired case by entering the case number and pressing Enter. This is disabled in the Variable editor.
The third part has a text box which displays the unformatted data values (as same as the data displayed when F2 is pressed in the grid) of the active cell. You can modify the values. This text box will be disabled in the variable editor as well as in the view mode of data editor.

![Image of text box]

**Changing Variable Names and Types**

You can change variable names and types using the Variable editor. To change a variable name, click on the existing variable name in the Variable Name column and type a new name. Double-click if you just want to edit the existing name.

You can also change variable names using the Variable Properties dialog box (or the DEFVAR command). To do this, right-click on a variable name and select Variable Properties.

**Changing variable names.** Type the new name. String variables must end in $.

**Changing variable types.** You can change a variable from numeric to string (and vice versa).

- **String to numeric.** If the variable contains non-numeric data, SYSTAT converts these entries to missing values. Cells with numeric data retain their numeric information with good accuracy. Cells with a combination of numeric and string data are converted to missing values.

- **Numeric to string.** SYSTAT uses the display format for the character entries.

**Adding Variable(s) or Case(s)**

You can add new variables and/or cases to a file in the same way you entered the original data.

- To add a new variable using the Variable editor, scroll past the existing variable names until you see empty rows. Click on the first empty variable name cell, and type a name for the new variable. If you are using the Data editor, scroll past the existing columns to any empty column, not necessarily the first, but the empty column after the file variables named with some numbers. Right-click on the name, select Variable Properties, type a name for the new variable or use the default name.
Entering and Editing Data

in the dialog box, and press OK. SYSTAT automatically creates variables corresponding to the preceding empty columns as well.

- To add new case(s), move the cursor to the first empty (all blank) row and type the new data.

- To insert case(s) between existing cases, select a row and right click; then select: Insert case(s)...

or, from the menus choose:

Edit
Data
Insert Case(s)...

Choose or type the number of cases you want to insert.

- Select Before or After, to insert new case(s) before or after the selected row respectively.

- Click OK.

The inserted cases contain missing values.

- To insert variable(s) between existing variables, select a column and right click; then select:
  Insert variable(s)...

or, from the menus choose:

Edit
Data
Insert Variable(s)...
Choose or type the number of variables you want to insert.

Select Before or After, to insert new variable(s) before or after the selected variable respectively.

Click OK.

Inserted variable(s) receive default name(s) and are of numeric type. All cells are assigned a missing value.

You can insert cut or copied variables from the clipboard, between existing variables. To insert, right click on a variable name and select Insert Variable(s) from Clipboard. For details, refer to the section “Cutting/Copying and Pasting Data” on page 63.

To add variable(s) or case(s) from another file, use Append Variables or Append Cases from Merge files in the Data menu.

**Dropping Variable(s)**

To delete one or more variable(s), select the variables in the Data editor and choose:

- Edit
  - Cut...

or, from the menus choose:

- Edit
  - Data
  - Delete Variable(s)...

The remaining variables are then shifted to the left.

The difference between Cut and Delete Variable(s) is that the former deletes the selected data and also copies the data to the clipboard, so that it can be pasted to a new location, whereas the latter simply deletes the selected data.
**Deleting Case(s)**

Using the mouse, you can delete case(s) just like you delete variable(s). Select the case(s) in the Data editor and choose:

- Edit
  - Cut...

*or*, right-click and select Cut.

*or*, from the menus choose:

- Edit
  - Data
  - Delete Case(s)...

SYSTAT removes the specified cases, moving subsequent cases up to fill their places. Be aware that SYSTAT renumbers the cases after they have been deleted. It is often helpful to create an ID variable that numbers the case(s), to make it easier to track them through various file manipulations.

If you want to delete case(s) based on conditions, use the If Then Let dialog box.

To open the dialog box, from the menus choose:

- Data
  - Transform
  - If..Then Let...

**Cutting/Copying and Pasting Data**

You can rearrange variable(s) or case(s) by using Cut and Paste Variable(s) from the Edit menu, or the CUT and PASTE commands. For example, to move one or more adjacent variables to a different position in the data file:

- Choose the variable(s) you want to move using the mouse.
- Choose Cut from the Edit menu (or by right-clicking) to remove the variable(s) from their current location and copy them to the clipboard.
- Select the column where you want to paste the variable(s) and select Insert Variable(s) from Clipboard from Data in the Edit menu. The variable(s) are pasted to the left of the selected column.

You can overwrite one or more existing columns by selecting them and pasting. The original (overwritten) variable names are retained.
In the case of variables, you can also use CUT and PASTE commands. You can also use Copy to copy variable(s) to the clipboard without deleting them from their original location. However, if you try to paste the same variables to another location in the same data file, SYSTAT gives default names to the pasted variable(s), since two variables in the same data file cannot have the same name.

You can use Copy and Insert Variable(s) from Clipboard to copy just about any type of data, and even to copy data from one application to another, within the limits of common sense. In the Data editor, that means that pasted data must be of the appropriate type and “shape”. You cannot paste a string value into a numeric variable or copy a range of cells and paste them into a single cell.

You can also cut, copy and paste data using the shortcut menu. For more information on right-click options, refer to the section “Right-Click Editing” on page 66.

**Copy as Text**

SYSTAT provides a feature for copying data, exactly as it appears in the Data editor, to the clipboard. To copy, from the menus choose:

```
Edit
  Data
    Copy as Text...
```

For example, if value labels are defined for a variable and are displayed in the Data editor, then Copy as Text copies the value labels to the clipboard instead of the original values. If a numeric variable is formatted with 3 decimal places, then Copy as Text copies the data with just 3 decimal places.

**Paste with Options**

Paste with Options is a dialog that will allow you to specify the row and column separators used in a block of ASCII data. Thus, you can now arrange a block of data before pasting it in the Data editor. To open the dialog, right-click on a Data editor cell and select Paste with Options. Or, from the menus, choose:

```
Edit
  Data
    Paste with Options...
```
You can choose any of the commonly used separators, namely, the tab, space, comma, semi-colon and the enter character. You can also specify other separators.

**Improved paste**

Data dragged from any application can be pasted in the Data editor. The paste with options dialog box will be opened to allow split data as necessary.

**Insert Variable(s) from Clipboard**

You can insert cut or copied variable(s) from the clipboard in the Data editor directly, without first having to create new columns. You can insert single or multiple variables
within the same data file or in another data file. To insert variables, right-click on a variable name and select Insert Variable(s) from Clipboard. Or, from the menus choose:

Edit
  Data
  Insert Variable(s) from Clipboard...

The inserted variables will have the same variable properties as the original variables. However variables inserted after a copy operation will receive default names, since no two variables in the Data editor can have the same name.

Right-Click Editing

When you right-click on any variable in the active data file on the Data editor, you will be presented with several editing options. The behavior of these options depends on the location of the right-click.

<table>
<thead>
<tr>
<th>Editing Option</th>
<th>Row Heading</th>
<th>Column Heading</th>
<th>Data Cells</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable Properties</td>
<td>Menu item does not appear.</td>
<td>Gives the user access to the Variable Properties dialog box. If the variable has already been defined, it brings up the dialog box with the settings for that variable. If multiple variables are selected, the dialog reflects properties of the current variable in the selection.</td>
<td>Menu item does not appear.</td>
</tr>
<tr>
<td>Variable Statistics</td>
<td>Menu item does not appear.</td>
<td>Displays the basic statistics and histogram of the current variable. In case of categorical variable, bar graph is shown.</td>
<td>Menu item does not appear.</td>
</tr>
<tr>
<td>Cut</td>
<td>Cuts the selected case(s) to the clipboard and shifts the succeeding case(s) up by the number of cases cut.</td>
<td>Cuts the selected variable(s) to the clipboard and shifts the succeeding variable(s) to the left by the number of variables cut.</td>
<td>Cuts the content of the selected cells to the clipboard and defines the selected cells as missing values.</td>
</tr>
<tr>
<td>Copy</td>
<td>Copies the selected case(s) to the clipboard.</td>
<td>Copies the selected variable(s) to the clipboard.</td>
<td>Copies the content of the selected cells to the clipboard.</td>
</tr>
<tr>
<td>Menu Item</td>
<td>Description</td>
<td></td>
<td></td>
</tr>
<tr>
<td>------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Copy as text</td>
<td>Copies the content of the selected cells exactly as it appears in the Data editor. If value labels of a variable are displayed instead of data values, this option copies the value labels of that variable.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Paste</td>
<td>Pastes the content of the clipboard in or starting from the current window.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Paste with Options</td>
<td>Menu item does not appear.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Paste Data</td>
<td>Menu item does not appear.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Paste Properties</td>
<td>Menu item does not appear.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Insert Variable(s)</td>
<td>Menu item does not appear.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Insert Variable(s) from Clipboard</td>
<td>Insert variables with data from the clipboard to the left of the current variable. This item is disabled when there is no data in the clipboard.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Delete</td>
<td>Sets the selected case(s) to missing values.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Insert Case(s)</td>
<td>Opens the Insert Case(s) dialog box where you can specify the number of cases to insert. You can insert up to 10 cases at a time. All inserted values are set to missing values.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Delete Case(s)</td>
<td>Deletes the selected case(s) from the Data editor and shifts all succeeding case(s) up by the number of cases deleted.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Chapter 3

<table>
<thead>
<tr>
<th>Insert Variable(s)</th>
<th>Menu item does not appear.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Opens the Insert Variable(s) dialog box where you can specify the number of variables to insert. You can insert upto 10 variables at a time. All inserted values are set to missing values.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Delete Variable(s)</th>
<th>Menu item does not appear.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Deletes the selected variable(s) from the Data editor and shifts all succeeding variable(s) by the number of variable(s) deleted.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>View Value Labels</th>
<th>Menu item does not appear.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Displays either the already defined value labels or the data values of the current variable. This option is disabled when no value labels are defined for a variable.</td>
</tr>
</tbody>
</table>

Right-click on the Data editor tab in the Viewspace

<table>
<thead>
<tr>
<th>Editing Option</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copy All</td>
<td>Copies all the contents of the data file to the clipboard.</td>
</tr>
<tr>
<td>Set as Active Data File</td>
<td>Sets the current data file as Active, if it is already not. Otherwise, this option is disabled.</td>
</tr>
<tr>
<td>Data/Variable Editor</td>
<td>Takes the control to the Variable editor, if it is in the Data editor and vice versa.</td>
</tr>
<tr>
<td>File Comment</td>
<td>Allows you to give comments and edit it for the current data file. To view the comments given to the current data file, pause the mouse pointer at the top left corner of the Data/Variable editor.</td>
</tr>
<tr>
<td>Show Toolbar</td>
<td>Shows /Hides the Toolbar for editing data.</td>
</tr>
<tr>
<td>New</td>
<td>Opens a new data file.</td>
</tr>
<tr>
<td>Save As</td>
<td>Saves the current data file if it is not active, otherwise this option is disabled.</td>
</tr>
<tr>
<td>Options</td>
<td>Opens the Global Options dialog box.</td>
</tr>
<tr>
<td>Close</td>
<td>Closes the current data file. If the current data file is active a prompt for saving the data file appears before closing.</td>
</tr>
</tbody>
</table>
Find Values in Column

To search for numbers, strings, or dates in a column, from the menus choose:

Edit
Data
Find in Column...

You can search for a string or number as a part or whole of each cell entry in any given column of the data file. You can specify the direction of the search — Up to search from the current location to the first case, — Down to search from the current location to the last case in the column. By default, the search is performed on the current variable; you can change the variable to be searched. To search only for complete entries, check Match whole word. To search for entries with the matching case, check Match case.

Replace Values in Column

To search and replace for numbers, strings, or dates in a column, from the menus choose:

Edit
Data
Replace in Column...
You can search for the specified string or number as a part or whole of each cell entry in a variable, and replace them with the desired string or number. By default, the search is performed on the current variable; you can change the variable to be searched. To search only for complete entries, check Match whole word. To search for entries with the matching case check Match case. Press Replace to replace the search string, and Find Next if you do not want to replace a particular entry. To replace all the entries, press Replace All. You can replace values of a numeric variable with numbers only.

**Find Variable**

To search for the variables in the Data editor, from the menus choose:

Edit
  Data
    Find Variable...
Select a variable from the Available variable(s) list that you want to find.

**Go To**

To go to a particular case of a variable in the Data editor, from the menus choose:

*Edit*
  *Data*
    *Go To...*

The current variable and case number is shown by default. Select a desired variable and specify the case number. The chosen cell will then be highlighted in the Data editor.
Chapter 3

**Undo/Redo**

The Undo/Redo facility allows you to retrieve any data that may have been modified accidentally. To undo an action in the Data editor, from the menus, choose

**Edit**
- **Undo**…

To undo using the keyboard, click in the Data editor and press CTRL + Z.

To redo an action in the Data editor, from the menus, choose

**Edit**
- **Redo**…

To redo using the keyboard, click in the Data editor and press CTRL + Y.

You can undo up to a maximum of 32 recent changes in the data you are currently editing. Any changes you make in the Data editor can be undone through this facility. For example, manual editing of data, Find and Replace, commands like LET, RANK, SORT, etc.. In the case of variable properties, undo/redo will work only for the changes made to the variable name and type. Also note that undo/redo will not work for any modifications in the data processing conditions, namely, Category, By groups, Select cases, Frequency, Weight, Id variable, and Order of categories.

If Undo/Redo introduces a new variable in the data file, that variable will have exactly the same properties it had when it was removed from the file. For example, suppose you insert a variable, set it to be categorical and then undo the operation. Now, suppose you redo the operation. The variable gets inserted as a categorical variable.

If the variable removed was a frequency variable, Undo will bring it back as a frequency variable provided no frequency setting change has been made in the interim. The same holds for weight and ID variables as well.

**Case ID**

You can specify an ID variable to be displayed by choosing **ID Variable** from the Data menu or by using the **IDVAR** command. The value of the specified variable (instead of the case number) is displayed.
To return to displaying the case number, select Turn off.

**Other Data Structures**

In addition to the usual rectangular, case-by-variable organization, SYSTAT accommodates two other data structures:

- Tabulated form.
- Matrix input.

**Data in Tabulated Form**

Choose By Frequency from Case Weighting in the Data menu or use the FREQ command to identify that the data are counts. That is, case(s) with the same values are entered as a single case with a count. If a variable is declared as a frequency variable, an icon indicating the frequency is displayed on the top of the variable in the Data editor. Note that frequency works for rectangular data only.
For example, Morrison’s data from a breast cancer (CANCER) study of 764 women are shown below. Instead of 764 cases, the data file contains 72 records for cells defined by the factors: 1) survival, 2) age group, 3) diagnostic center, and 4) tumor status.

<table>
<thead>
<tr>
<th>Survives</th>
<th>Age</th>
<th>Centers</th>
<th>Tumors</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alive</td>
<td>50</td>
<td>Boston</td>
<td>MaxBengn</td>
<td>0</td>
</tr>
<tr>
<td>Alive</td>
<td>50</td>
<td>Boston</td>
<td>MaxMalig</td>
<td>4</td>
</tr>
<tr>
<td>Alive</td>
<td>50</td>
<td>Boston</td>
<td>MinBengn</td>
<td>24</td>
</tr>
<tr>
<td>Alive</td>
<td>50</td>
<td>Boston</td>
<td>MinMalig</td>
<td>11</td>
</tr>
<tr>
<td>Dead</td>
<td>50</td>
<td>Boston</td>
<td>MaxBengn</td>
<td>0</td>
</tr>
<tr>
<td>Dead</td>
<td>50</td>
<td>Boston</td>
<td>MaxMalig</td>
<td>6</td>
</tr>
<tr>
<td>Dead</td>
<td>50</td>
<td>Boston</td>
<td>MinBengn</td>
<td>7</td>
</tr>
<tr>
<td>Dead</td>
<td>50</td>
<td>Boston</td>
<td>MinMalig</td>
<td>6</td>
</tr>
<tr>
<td>Alive</td>
<td>50</td>
<td>Glamorgn</td>
<td>MaxBengn</td>
<td>1</td>
</tr>
<tr>
<td>Alive</td>
<td>50</td>
<td>Glamorgn</td>
<td>MaxMalig</td>
<td>8</td>
</tr>
<tr>
<td>Dead</td>
<td>60</td>
<td>Boston</td>
<td>MinBengn</td>
<td>20</td>
</tr>
<tr>
<td>Dead</td>
<td>60</td>
<td>Glamorgn</td>
<td>MinMalig</td>
<td>8</td>
</tr>
<tr>
<td>Alive</td>
<td>60</td>
<td>Glamorgn</td>
<td>MaxBengn</td>
<td>4</td>
</tr>
<tr>
<td>Alive</td>
<td>60</td>
<td>Glamorgn</td>
<td>MaxMalig</td>
<td>10</td>
</tr>
<tr>
<td>Alive</td>
<td>60</td>
<td>Glamorgn</td>
<td>MinBengn</td>
<td>39</td>
</tr>
<tr>
<td>Alive</td>
<td>60</td>
<td>Glamorgn</td>
<td>MinMalig</td>
<td>27</td>
</tr>
<tr>
<td>Dead</td>
<td>60</td>
<td>Glamorgn</td>
<td>MaxBengn</td>
<td>0</td>
</tr>
<tr>
<td>Dead</td>
<td>60</td>
<td>Glamorgn</td>
<td>MaxMalig</td>
<td>3</td>
</tr>
<tr>
<td>Dead</td>
<td>60</td>
<td>Glamorgn</td>
<td>MaxBengn</td>
<td>12</td>
</tr>
</tbody>
</table>
**Entering and Editing Data**

<table>
<thead>
<tr>
<th>NUMBER</th>
<th>Alive/Dead</th>
<th>Age</th>
<th>Region</th>
<th>Condition</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Dead</td>
<td>60</td>
<td>Glamorgn</td>
<td>MinMalig</td>
<td>14</td>
</tr>
</tbody>
</table>

|        | Alive      | 70  | Tokyo   | MaxBengn  | 1     |
|        | Alive      | 70  | Tokyo   | MaxMalig  | 5     |
|        | Alive      | 70  | Tokyo   | MinBengn  | 6     |
|        | Alive      | 70  | Tokyo   | MinMalig  | 1     |
|        | Dead       | 70  | Tokyo   | MaxBengn  | 0     |
|        | Dead       | 70  | Tokyo   | MaxMalig  | 1     |
|        | Dead       | 70  | Tokyo   | MaxBengn  | 3     |
|        | Dead       | 70  | Tokyo   | MinMalig  | 2     |

*NUMBER* is the count of women in each cell. Choose By Frequency from Case Weighting in the Data menu to identify this variable as the count variable.
Matrix Input

You can enter a triangular matrix such as a correlation matrix. As usual, variable names go in the top row. Next, type the correlations. Since correlation matrices are symmetric, type only the lower diagonal portion.

Choose Save As from the File menu to save the data, and from the Save drop-down, click on Save Options to specify the type of matrix. The available types include rectangular (the usual file of case(s) and variable(s)), SSCP, covariance, correlation, dissimilarity, and similarity.

Global Data Options

To customize the behavior of the Data editor, from the menus choose:

Edit
Options...

Then click the Data tab on the Edit:Options dialog box.
Default font. You can specify the font type and size to use for displaying data values and variable properties in the Data editor.

Default variable numeric format. These settings control the default display of numeric data in the Data editor. Field width is the total number of digits in the data value. Decimal places are the number of decimal places to display in the data value. This affects only newly created variables in the current and also the subsequent data files.

Default date and time format. All date and time variables without a specified format appear in this selected format. The number of characters indicates the number of digits or letters displayed.
**Data editor cursor.** Controls the navigation behavior of the Enter key in the Data editor. The Enter key can either move to the next cell on the right for case-by-case entry or down to the next row for variable-by-variable entry.

**Set century range for 2-digit years.** When dealing with two-digit years, SYSTAT’s Data editor and some date and time function need a reference point for assigning a century. By default, this reference point is 1930 which means all two-digit years are assumed to correspond to the 100-year range from 1930 to 2029. SYSTAT treats two digits years that fall below the final two digits of the Begin year as members of the next century. For example, 28 is treated as 2028 and 31 is treated as 1931. You can select 20th century or 21st century if you want the 100-year range to begin at 1900 or 2000 respectively.

**Apply Category variable information from data file (require reopen of file).** By default, if you define a variable to be categorical and save the data file subsequently, the category information is saved to the file. This implies that you need not declare the variable to be categorical every time you open the data file. However, you may want to uncheck this option if you need to process the variable as non-categorical sometimes, or if you need to process different sets of categorical variables each time.

**Save ID variable information to data file.** An ID variable is used to identify cases in the output, instead of simply displaying case numbers. By default, like categorical information, ID variable information is also saved to the data file. You can uncheck this option if you do not want ID variable values to identify cases all the time.

**Trim leading and trailing spaces for string variable data.** SYSTAT allows you to enter string data values with leading and trailing spaces. Some of SYSTAT's character functions, namely LFT$, CNT$, RGT$, LPD$ and RPD$, also insert leading and/or trailing spaces in string values. Check this option if you want such spaces to be trimmed out for processing by the LABEL, ORDER and RECODE commands.

**Close active data file when another is opened (requires restart of application).** When this option is checked, the currently active data file closes down after prompting the user to save changes, if any. By default this option is unchecked and the currently active data file switches to view mode instead of closing down.
Using Commands

To define variable properties for numeric variables, use:

```plaintext
DEFVAR varlist / TYPE = NUMBER or DATE or EXPONENTIAL,
          DISPLAY = {m,n} or 'dateformat', COMMENT='text'
```

```plaintext
VALLAB varlist / 'label'
```

```plaintext
LABEL varlist / n1='text1', n2='text2',...
               or
               n1,n2,...='text1',
               n3,n4,...='text2',
               or
               n1..n2 = 'text1',
               n3..n4 = 'text2',...
```

To define variable properties for string variables, use:

```plaintext
DEFVAR varlist$ / TYPE=STRING DISPLAY = n COMMENT = 'text'
```

```plaintext
VALLAB varlist$ / 'label'
```

```plaintext
LABEL varlist$ / 'oldtext1' = newtext1, 'oldtext2' = newtext2, ...
```

You may use VALLAB instead of LABEL.

To set a datafile named filename as an active file under active tab, use:

```plaintext
ACTIVE filename
```

To control the display of variable labels, use:

```plaintext
VDISPLAY LABELS or NAME or BOTH
```

To control the display of value labels, use:

```plaintext
LDISPLAY LABELS or DATA or BOTH
```

To specify category variable(s), use:

```plaintext
CATEGORY varlist
```

To reset the previously declared category variables in varlist, use:

```plaintext
CATEGORY varlist/OFF
```
Some more options of CATEGORY are available. Refer the Language Reference for details.

To specify an ID variable:

```plaintext
IDVAR var
```

Typing IDVAR without an argument returns the display to case numbers.

The following editing commands are also available:

```plaintext
REPEAT n

INSERT nloc, m / ROWS or COLUMNS

CUT var1 or varlist

PASTE nloc

DELETE n1, n2, n3, ...

DELETE COLUMNS=n1, n2, n3, ...
```

To close any opened file, use:

```plaintext
CLOSE filename
```
Chapter 4

Data Transformations

Leland Wilkinson and Laszlo Engelmann
(revised by Mangalmurti Badgujar, Anand Ramchandran, and S. Anoopama)

Often, the data as recorded are not sufficient for a complete analysis. For example, you may need to reexpress values to improve symmetry, derive new variables such as a total score or a ratio of two variables, select a subset of the cases, etc. To execute these tasks, you specify statements in this form:

```
LET variable = expression
IF condition THEN LET assignment
SELECT condition1 AND condition2 OR ...
```

SYSTAT provides a variety of functions that you can use in these statements. You can:

**Derive new variables.** If you have two quiz scores and a final exam score for each student, you can use the arithmetic operators + and * to compute their total score:

```
LET total = quiz1 + quiz2 + 2*final
```

**Reexpress data.** Transformed values may meet assumptions required for statistical analyses better than the original data. For example, often the distribution of the logarithm of body weight is more symmetric than that of untransformed weights. To reexpress weight in log base 10 units, use the \( \text{L10} \) function:

```
LET lg_wt = L10(weight)
```

**Assigning a value of data variable to temporary variable.** Assigns a particular value of a data variable to a temporary variable using the \text{DATA} function. This temporary variable will not appear in the Data editor.

```
Temp avg~ = DATA(average,1)
```
This avg contains the value of average.

**Compute summary variables.** For each case, SYSTAT’s multivariable functions compute results across variables. For example, use the AVG function to compute the average of the scores S1 to S6 for each subject:

\[
\text{LET average } = \text{AVG}(s1, s2, s3, s4, s5, s6)
\]

**Assign new values conditionally.** For example, using the variable TEMP (winter temperature), derive a new variable TEMPS containing a character descriptor:

\[
\text{IF temp } < 32 \text{ THEN LET temps } = \text{'Cold'}
\]

**Execute global edits.** Use relational operators (<, >, and =) and logical operators (AND and OR) in conditional transformations. For example, the statement

\[
\text{IF (age } > 20 \text{ AND income } < 7000 \text{ AND job$ } = \text{'No'},
\text{ THEN LET relief$ } = \text{'Yes'}
\]

stores the word Yes in the variable named RELIEFS for every subject over 20 years old who reports an income of less than $7,000 and is unemployed.

**Select cases.** Use relational operators to restrict your analysis to subjects satisfying certain conditions. For example,

\[
\text{SELECT age } \geq 21
\]

omits all subjects under 21.

**Rank, center or standardize the values of a variable.** You can rank the values of a variable from smallest to largest, center the values around their mean, or standardize the values of a variable with their $z$ scores or score on a 0,1 scale.

**Manipulate character variables.** Use SYSTAT’s character functions to manipulate character values, including changing case, extracting characters or text strings, converting characters to numbers, etc.

**Format and transform dates and times.** For example, you could write the fifth of February, 2007, as Feb 5, 2007, 02/05/07, or 05–02–07; or, you could compute the time between events in, say, days or even seconds.
Generate random numbers. SYSTAT has functions to generate values from a variety of univariate discrete and continuous distributions. In addition, density, cumulative and inverse cumulative functions are available for univariate continuous and discrete distributions. For more details, refer to the section “Functions Relating to Probability Distributions” on page 109.

For more complex data transformations, you can use IF…THEN… ELSE statements, FOR…NEXT or WHILE…ENDWHILE loops, ARRAY, and DIM (Dimension) statements.

Let Dialog Box

LET assigns values to a variable according to a specified expression. You can use any mathematically valid combination of variables, numbers, functions, and operators, including complex expressions joined by the logical operators AND, OR, or NOT. Valid expressions include:

```
LET CHANGE = WEEK2 - WEEK1
LET LOGIT = 1 / (1 + EXP(A+B*X))
LET trendy = income > 60000 AND car$=='BMW'
```

To open the Let dialog box, from the menus choose:

Data
    Transform
        Let...
To specify the target variable, select a variable from the Available variable(s) list and press the Variable button. If you want to create a new variable, simply type it into the box. Select the function type, functions and variable(s) to be used in the expression from the list and use the Add and Expression buttons to add them to the expression. Your expression will appear in the Expression area as you create it.

You can transform existing variables or create new ones. If the target variable (the variable following LET) already exists in the file, its values are replaced by the value of the expression for each case. If the variable does not exist, then a new variable with that name is added in the first column at the right side of the data matrix. For example, LET B = A creates a new variable named B if B does not already exist. If it does exist, its values are replaced by the values of A.

**Rules For Expressions**

- If the expression contains any character values, they must be enclosed in single or double quotation marks. Character values are case sensitive.
- Arguments for functions must be inside parentheses, for example, LOG(\textit{WEIGHT}) or SQR(\textit{INCOME}). If there are two or more arguments, they must be separated by commas.
**If...Then Let Dialog Box**

IF...THEN LET allows you to transform variables conditionally. For all cases where the condition is true, SYSTAT executes the action. You can use any mathematically valid combination of variables, numbers, functions, and operators. Examples of valid statements are:

```
IF AGE > 80 THEN LET AGE$ = 'ELDERLY'
IF X = 99 THEN LET X = .
IF (SEX$ = 'MALE') AND (AGE > 30) THEN LET GROUP = 1
```

To open the If...Then Let dialog box, from the menus choose:

Data
   Transform
   If...Then Let...

You can transform existing variables or create new ones using the If Then Let dialog. If the target variable (the variable following LET) already exists in the file, its values are replaced by the transformation value for each case. If the target variable does not exist,
then a new variable with that name is added in the first column at the right side of the data matrix.

**Mode of input.** Select one of the following two options:

- **Select.** Select the variables and functions to build the transformation expression. To add an element to the transformation expression, position the cursor in the desired location before selecting the variable or function to be added. Choose the appropriate condition from the Condition box. To add a variable to the expression, click on it and click the Expression button. Choose the appropriate Function type and function from the Functions box and click the Add button. If you want to create a new variable, simply type it into the desired Expression box.

- **Type.** Type the variables and functions to get the desired transformation expression. To add a variable from the list, use the Add button. Choose the appropriate Function type and function from the Functions box and click the Add button.

**Rules For Expressions**

- If the expression contains any character values, those values must be enclosed in single or double quotation marks. Character values are case sensitive.
- Arguments for functions must be inside parentheses, for example, LOG(weight) or SQR(income). If there are two or more arguments, they must be separated by commas.

**Built-In Functions**

Built-in variables allow you to index aspects of files:

- `CASE()`: Case (observation) number
- `COMPLETE()`: Column of 1’s (for complete cases) and 0’s (for cases containing missing values)

For example, if your cases are ordered by time, create a new variable named `TIME` containing the case sequence numbers:

```plaintext
LET time = CASE()
```
Shortcuts for Transformations

There are several shortcuts you can use to minimize typing for transformation statements.

**Multiple transformations: The @ sign.** To transform several variables, you can specify a LET statement for each:

```
LET gdp_cap = L10(gdp_cap)
LET mil = L10(mil)
LET gnp_86 = L10(gnp_86)
```

Or, you can use the @ sign to specify the same transformations in one statement:

```
LET (gdp_cap, mil, gnp_86) = L10(@)
```

The syntax for multiple transformations is:

```
LET (var1, var2, var3, ...) = expression with @ sign
```

The variable names must be separated by commas or space and enclosed within parentheses (). The @ sign is a placeholder for the variable names. The function can be any valid SYSTAT function or expression.

This @ shortcut notation can also be used with relational operators:

```
LET (blue, depress, cry, sad, no_eat, getgoing) = @ <> 0
```

This dichotomizes the values of six variables into 0 and 1.0. When the value of a variable is not 0, the statement on the right is true, so the result is 1.0. If the value is 0, the statement is false, so the result is 0.0.

**Specifying contiguous variables.** As a shortcut when selecting adjacent variables in a file, type the first and last variable names and use a double period (..) to include all variables in between:

```
LET total = SUM(score1, score2, score3, score4)
LET total = SUM(score1 .. score4)
```

This shortcut can also be used with the @ sign shortcut:

```
LET (blue .. getgoing) = @ <> 0
```
When using subscripts (such as $Q(1)$, $Q(2)$, etc.), variables that are not contiguous can be specified using the double-period shortcut. For example,

```
STANDARDIZE Q(1) .. Q(10)
```

standardizes the variables $Q(1)$ to $Q(10)$, regardless of their order in the data file.

**Operators and Functions**

SYSTAT’s functions and operators can be used:

- In any LET, IF...THEN LET, or SELECT expression.
- In some FIND expressions (depending on data type).
- In the calculator.

Operators are categorized as arithmetic, relational, and logical. Functions are categorized as mathematical, multivariable, multicase, group and interval, date and time, character, and distribution.

**Arithmetic Operators**

You can use the following arithmetic operators:

- `+` Addition
- `-` Subtraction or negative sign
- `/` Division
- `*` Multiplication
- `**` or `^` Exponentiation

Some examples include:

- `quiz1 + quiz2`
- `y - 2`
- `year*year`
- `birth_rt / death_rt`
- `x^2`

These could be used, for example, as follows:

```
LET total = quiz1 + quiz2
IF (city$ = 'LA') THEN LET code=y-2
```
SYSTAT sets the resulting values of inadmissible operations (for example, division by 0) to missing.

**Relational Operators**

Relational operators are used in relational expressions to compare either two numeric or two character expressions.

- `= or ==` Equal to
- `<` Less than
- `<=` Less than or equal to
- `>` Greater than
- `>=` Greater than or equal to
- `<>` Not equal to

Some examples include:

```
score1 < score2
state$ = 'NY'
age > 21
x <> .
age <= 65
income >= 3000
```

These could be used, for example, as follows:

```
IF (state$ == 'NY') THEN LET region = 1
```

**Logical Operators**

The logical operators `AND` and `OR` connect two relational expressions, and `NOT` negates a logical expression. These operators yield results that are either true or false.

This expression selects people of work-force age (those older than 17 and younger than or equal to 65):

```
age > 17 AND age <= 65
```

Note that even if you are testing for different conditions on the same variable, you must specify a complete relational expression for each condition.

Correct: `age > 17 AND age <= 65`

Incorrect: `age > 17 AND <= 65`
Here, we select people who are not of work-force age (those younger than or equal to age 17 or older than age 65):

\[
age \leq 17 \text{ OR } age > 65
\]

The logical operator NOT reverses the value of an entire expression. To select people who are not of work-force age, we could also specify:

\[
\text{NOT} \ (age > 17 \text{ AND } age \leq 65)
\]

If the result of an expression is true, it is assigned a value of 1; if it is false, it is assigned a value of 0. NOT sets any nonzero value (true) to 0, and any 0 value (false) to 1 (true).

SYSTAT follows the standard for programming languages and returns 0 for false and 1 for true.

**Logical AND**

The following example demonstrates how a result is determined:

\[
\text{LET } \text{TRENDY} = \text{income} > 40000 \text{ AND car$='BMW'$}
\]

For any case where the subject’s income is over $40,000 AND the subject’s car is a BMW, the result of the expression is true and the value of TRENDY is 1. If the subject has a lower income OR a different car, the result is false and TRENDY is assigned 0. If the subject fails to report income or car model, the value of TRENDY is set to missing.

In summary, when AND is used:

<table>
<thead>
<tr>
<th>Condition 1</th>
<th>Condition 2</th>
<th>Result</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>AND</td>
<td>True</td>
<td>yields True</td>
</tr>
<tr>
<td>True</td>
<td>AND</td>
<td>False</td>
<td>yields False</td>
</tr>
<tr>
<td>False</td>
<td>AND</td>
<td>True</td>
<td>yields False</td>
</tr>
<tr>
<td>False</td>
<td>AND</td>
<td>False</td>
<td>yields False</td>
</tr>
<tr>
<td>True</td>
<td>AND</td>
<td>Missing</td>
<td>yields Missing</td>
</tr>
<tr>
<td>Missing</td>
<td>AND</td>
<td>True</td>
<td>yields Missing</td>
</tr>
<tr>
<td>False</td>
<td>AND</td>
<td>Missing</td>
<td>yields False</td>
</tr>
<tr>
<td>Missing</td>
<td>AND</td>
<td>False</td>
<td>yields False</td>
</tr>
<tr>
<td>Missing</td>
<td>AND</td>
<td>Missing</td>
<td>yields Missing</td>
</tr>
</tbody>
</table>
**Logical OR**

If the logical operator OR is used instead of AND, for example:

```sql
LET TRENDY = INCOME > 40000 OR CAR$=='BMW'
```

only one of the conditions has to be true for the result to be true (1).

<table>
<thead>
<tr>
<th>Condition 1</th>
<th>Condition 2</th>
<th>Result</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>OR</td>
<td>True</td>
<td>1</td>
</tr>
<tr>
<td>True</td>
<td>OR</td>
<td>False</td>
<td>1</td>
</tr>
<tr>
<td>False</td>
<td>OR</td>
<td>True</td>
<td>1</td>
</tr>
<tr>
<td>False</td>
<td>OR</td>
<td>False</td>
<td>0</td>
</tr>
<tr>
<td>True</td>
<td>OR</td>
<td>Missing</td>
<td>1</td>
</tr>
<tr>
<td>Missing</td>
<td>OR</td>
<td>True</td>
<td>1</td>
</tr>
<tr>
<td>False</td>
<td>OR</td>
<td>Missing</td>
<td>.</td>
</tr>
<tr>
<td>Missing</td>
<td>OR</td>
<td>False</td>
<td>.</td>
</tr>
<tr>
<td>Missing</td>
<td>OR</td>
<td>Missing</td>
<td>.</td>
</tr>
</tbody>
</table>

**Order of Expression Evaluation**

Expressions are evaluated from left to right according to the precedence of operators. That is, operators with higher precedence are evaluated before those with lower precedence. Order of precedence from highest to lowest runs as follows:

1. Expressions enclosed in parentheses `( )`
2. Exponentiation `^` or `**`
3. Negation `–`
4. Multiplication and division `*`, `/`
5. Addition and subtraction `+`, `–`
6. Comparison `=, <=, <, >, <, >`
7. Logical negation `NOT`
8. Logical comparison `AND`
9. Logical comparison `OR`

Because of this order of precedence, the expressions \((A + B \times C)\) and \((A + (B \times C))\) are the same, but they differ from \(((A + B) \times C)\).

Multiplication has a higher precedence than addition and is therefore performed first. If \(A = 1\), \(B = 2\), and \(C = 3\), the above expressions evaluate as follows:
The only exception to the “left to right” rule is with exponentiation. The exponentiation $a^b^c$ is evaluated “right to left”; that is, $a^b^c$ means $a^{(b^c)}$, or $a^{b^c}$.
## Mathematical Functions

These functions modify the variable, number or expression that you place inside the parentheses. Some examples include:

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQR (a)</td>
<td>Square root of $a$ [$a \geq 0$].</td>
</tr>
<tr>
<td>LOG (a)</td>
<td>Natural logarithm of $a$ [$a &gt; 0$].</td>
</tr>
<tr>
<td>L10 (a)</td>
<td>Logarithm base 10 [$a &gt; 0$].</td>
</tr>
<tr>
<td>EXP (a)</td>
<td>Exponential function $e^a$.</td>
</tr>
<tr>
<td>LAG (var, n)</td>
<td>Lag variable by shifting values down $n$ rows; if $n$ is omitted, default = 1.0.</td>
</tr>
<tr>
<td>INT(a)</td>
<td>Integer part of $a$.</td>
</tr>
<tr>
<td>LGM (a)</td>
<td>Log gamma: $\text{LGM}(n) = \log(\Gamma(n))$, $=\log((n-1)!) \text{ when } n \text{ is a positive integer}$</td>
</tr>
<tr>
<td>SGN (a)</td>
<td>$-1$ if $a &lt; 0$, $0$ if $a = 0$, and $1$ if $a &gt; 0$.</td>
</tr>
<tr>
<td>ABS (a)</td>
<td>Absolute value $</td>
</tr>
<tr>
<td>SIN (a)</td>
<td>Sine of $a$ (in radians).</td>
</tr>
<tr>
<td>COS (a)</td>
<td>Cosine of $a$ (in radians).</td>
</tr>
<tr>
<td>TAN (a)</td>
<td>Tangent of $a$ (in radians).</td>
</tr>
<tr>
<td>ASN (a)</td>
<td>Arcsine of $a$ (which yields radian results).</td>
</tr>
<tr>
<td>ACS (a)</td>
<td>Arccosine of $a$ (which yields radian results).</td>
</tr>
<tr>
<td>TNH (a)</td>
<td>Hyperbolic tangent of $a$.</td>
</tr>
<tr>
<td>ATN (a)</td>
<td>Arctangent of $a$ (which yields radian results).</td>
</tr>
<tr>
<td>ATH (a)</td>
<td>Arc hyperbolic tangent of $a$ (Fisher’s $Z$, which yields radian results).</td>
</tr>
<tr>
<td>AT2 (a,b)</td>
<td>Arctangent with sine ($a$) and cosine ($b$) argument.</td>
</tr>
<tr>
<td>MOD (a,b)</td>
<td>The remainder of $a / b$.</td>
</tr>
<tr>
<td>CASE()</td>
<td>Case number of that case.</td>
</tr>
<tr>
<td>CEIL(a)</td>
<td>Nearest integer more than or equal to the number $a$.</td>
</tr>
<tr>
<td>FLOOR(a)</td>
<td>Nearest integer less than or equal to the number $a$.</td>
</tr>
<tr>
<td>ROUND(a)</td>
<td>Nearest integer. The second argument will determine the significant digit upto which to round off. So, ROUND(arg1, 0) will be the same as ROUND(arg1) which will round off to the nearest integer. ROUND(arg1, 3) will round off to the third decimal place, and so on.</td>
</tr>
<tr>
<td>ACSH(a)</td>
<td>Arc hyperbolic cosine of $a$.</td>
</tr>
<tr>
<td>ASNH(a)</td>
<td>Arc hyperbolic sine of $a$.</td>
</tr>
<tr>
<td>COSH(a)</td>
<td>Hyperbolic cosine of $a$.</td>
</tr>
<tr>
<td>SINH(a)</td>
<td>Hyperbolic sine of $a$.</td>
</tr>
<tr>
<td>COLUMN (varname)</td>
<td>Returns the column number of the variable varname.</td>
</tr>
</tbody>
</table>
These could be used, for example, as follows:

\[
\text{LET } \lg \text{weight} = \text{LOG(}\text{weight}) \\
\text{LET preveven} = \text{EVEN(}\text{age}, -1)
\]

Arguments of trigonometric functions must be in radians, not degrees. \text{LOG} is the natural logarithm and \text{L10} is the base 10 logarithm. You can obtain logs in other bases by dividing the natural log by the log of the base; for example, for \(\log_2 A\), use \(\text{LOG}(A)/\text{LOG}(2)\). SYSTAT sets the resulting values of inadmissible operations (for example, square roots of negative numbers) to missing.

**Multivariable Functions**

The functions described below operate on values of variables within a case. With these functions, you can:

- Compute summary descriptive statistics across values of selected variables.
- Search selected variables for specific numeric or character values.
- Compute the slope of the line of best fit through points with equally spaced \(x\) values or values that you specify.
- Compute the area under a curve created by connecting equally spaced points or points with spacing that you specify.
Arguments are not restricted to variable names—they can be explicit values or other functions—and they can contain arithmetic operators, for example, \((\text{INDEX}–1)\).

Arguments of a function must be separated by commas. For example, when using \(\text{AVG}\) to compute the mean, specify: \(\text{AVG}(10,313,-29,236,19)\).

Use a double period (..) as a shortcut notation to shorten a list of contiguous variables—that is, for \(Q1, Q2, \ldots, Q20\), use \(Q1 .. Q20\).

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIS((y1,y2,\ldots))</td>
<td>Number of missing values.</td>
</tr>
<tr>
<td>NUM((y1,y2,\ldots))</td>
<td>Number of values that are not missing (number of usable values).</td>
</tr>
<tr>
<td>AVG((y1,y2,\ldots))</td>
<td>Mean of nonmissing values.</td>
</tr>
<tr>
<td>STD((y1,y2,\ldots))</td>
<td>Standard deviation of nonmissing values.</td>
</tr>
<tr>
<td>MIN((y1,y2,\ldots))</td>
<td>Smallest among nonmissing values (minimum value).</td>
</tr>
<tr>
<td>MAX((y1,y2,\ldots))</td>
<td>Largest among nonmissing values (maximum value).</td>
</tr>
<tr>
<td>SUM((y1,y2,\ldots))</td>
<td>Sum of nonmissing values.</td>
</tr>
<tr>
<td>SLE((y1,y2,\ldots))</td>
<td>Coefficient (b) of the regression line (y = a + bx), where (x)’s are assumed to be equally spaced at a distance of 1 unit.</td>
</tr>
<tr>
<td>SLU((x1,y1,x2,y2,\ldots))</td>
<td>Coefficient (b) of the regression line (y = a + bx).</td>
</tr>
<tr>
<td>ARE((y1,y2,\ldots))</td>
<td>Area under the values of ((x_i, y_i)), where (x)’s are assumed to be equally spaced at a distance of 1 unit.</td>
</tr>
<tr>
<td>ARU((x1,y1,x2,y2,\ldots))</td>
<td>Area under (y) by the trapezoidal rule.</td>
</tr>
<tr>
<td>COD((\text{number},\text{var1},\text{var2},\ldots))</td>
<td>The index (using integers 1, 2, 3, \ldots) when (\text{number}) matches a value in (\text{var1}) through (\text{varp}) respectively; 0 otherwise.</td>
</tr>
<tr>
<td>COD('\text{charval}',\text{var1$},\text{var2$},\ldots))</td>
<td>The index (using integers 1, 2, 3, \ldots) of (\text{var1$}, \text{var2$}, \ldots) when (\text{charval}) matches the value of the respective variable; 0 otherwise.</td>
</tr>
<tr>
<td>INC((\text{number},\text{var1},\text{var2},\ldots))</td>
<td>A 1 (true) if (\text{number}) matches a value in (\text{var1}, \text{var2}, \ldots), or (\text{varn}); 0 (false) otherwise.</td>
</tr>
<tr>
<td>INC('\text{charval}',\text{var1$},\text{var2$},\ldots))</td>
<td>A 1 (true) when (\text{charval}) matches a value in (\text{var1$}, \text{var2$}, \ldots), or (\text{varn$}); 0 (false) otherwise.</td>
</tr>
<tr>
<td>COMPLETE()</td>
<td>Returns 1 if value of variable(s) in the argument are all nonmissing, else zero; the built-in constant COMPLETE to be treated as a function COMPLETE().</td>
</tr>
</tbody>
</table>
Multicase Functions

The functions described below operate on cases of a variable. With these functions you can compute a value for specified cases.

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMIN(\text{variable, initial case, end case, step})</td>
<td>Smallest among non-missing values (minimum value).</td>
</tr>
<tr>
<td>CMAX(\text{variable, initial case, end case, step})</td>
<td>Largest among non-missing values (maximum value).</td>
</tr>
<tr>
<td>CSUM(\text{variable, initial case, end case, step})</td>
<td>Sum of non-missing values.</td>
</tr>
<tr>
<td>CPROD(\text{variable, initial case, end case, step})</td>
<td>Product of non-missing values.</td>
</tr>
<tr>
<td>CRANGE(\text{variable, initial case, end case, step})</td>
<td>Range of non-missing values (maximum value - minimum value).</td>
</tr>
<tr>
<td>CMEAN(\text{variable, initial case, end case, step})</td>
<td>Mean of non-missing values.</td>
</tr>
<tr>
<td>CVAR(\text{variable, initial case, end case, step})</td>
<td>Variance of non-missing values.</td>
</tr>
<tr>
<td>CSTD(\text{variable, initial case, end case, step})</td>
<td>Standard deviation of non-missing values.</td>
</tr>
</tbody>
</table>

- Initial case and step are '1' by default.
- Default end case is the last case of the file in use.
- Argument may not be just a variable name - it can be a mathematical function, containing the variable as an argument, and it can contain arithmetic operators.

Here is an example of a data file:

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
<td>11</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>12</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>13</td>
</tr>
<tr>
<td>4</td>
<td>9</td>
<td>14</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>15</td>
</tr>
</tbody>
</table>

\[ \text{EXP}\left(\text{CSUM}\left(Y \times (X+Z)\right)\right) \]

multiplies the sum of X and Z, with Y for each case and adds up the evaluated values. The exponential of the resultant is 4.308817E+286.
Group and Interval Functions

The COD function is used to recode values of a grouping variable. The INC function compares each value of a numeric or character variable with a list of values you specify. If a match is found, SYSTAT returns a 1; if not, SYSTAT returns a 0. The CUT function lets you define intervals on a quantitative (continuous) variable or divide the values of a character variable alphabetically.

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>COD(var,num1,num2,...)</td>
<td>Values of var are replaced: num1 becomes 1, num2 becomes 2, etc.</td>
</tr>
<tr>
<td>COD(var$,'text1','text2',...)</td>
<td>Values of var$ are replaced: text1 becomes 1, text2 becomes 2, etc.</td>
</tr>
<tr>
<td>INC(var,num1,num2,...)</td>
<td>A 1 (true) if the value in var matches one of the numbers num1, num2, ...; 0 (false) otherwise.</td>
</tr>
<tr>
<td>INC(var$,'text1','text2',...)</td>
<td>A 1 (true) if the value in var$ matches a value in text1, text2, ...; 0 (false) otherwise.</td>
</tr>
<tr>
<td>CUT(var,num1,num2,...)</td>
<td>Use to define intervals along a continuous variable. Values in var less than or equal to num1 get value 1. Values greater than num1 and less than or equal to num2 get value 2, etc.</td>
</tr>
<tr>
<td>CUT(var$,'text1','text2',...)</td>
<td>Use to group variables alphabetically. Values in var$ less than text1 (alphabetically), including text1, get value 1. Values between text1 and text2, including text2, get value 2, etc.</td>
</tr>
<tr>
<td>LAB$(var)</td>
<td>yields a variable that contains values generated by the LABEL command.</td>
</tr>
<tr>
<td>NCAT(var or var$, 0 or 1)</td>
<td>yields number of categories of var$. The result includes or does not include missing value as a separate category according as ‘1’ or ‘0’ is chosen for second argument.</td>
</tr>
<tr>
<td>NCAT(a)</td>
<td>Returns number of categories in the specified variable a.</td>
</tr>
<tr>
<td>BOF()</td>
<td>Beginning of file; returns 1 if beginning-of-file, else it returns 0.</td>
</tr>
<tr>
<td>EOF()</td>
<td>End of file; returns 1 if end-of-file, else it returns 0.</td>
</tr>
<tr>
<td>BOG()</td>
<td>When BY group is active, returns 1 if beginning-of-BY group, else it returns 0.</td>
</tr>
<tr>
<td>EOG()</td>
<td>When BY group is active, returns 1 if end-of-BY group, else it returns 0.</td>
</tr>
</tbody>
</table>
Chapter 4

**Date and Time Functions**

SYSTAT has functions that format and transform dates and times. SYSTAT can:

- Give the current date or time in the format you specify.
- Transform date or time character values to numeric values.
- Transform numeric date and time values to character values with a specified format.
- Return the day of the century (DOC) that corresponds to a given year, month, and day. Use DOC to compute time between events.
- Return the day of the week corresponding to a given day of the century.
- Use the following functions to change the format of dates and times, to convert numbers to dates and times, and vice versa.

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOW$( 'format' )</td>
<td>The current time and/or date in format you specify.</td>
</tr>
<tr>
<td>VAL(var$, 'format', field)</td>
<td>Extract numbers from dates or times with character values in var$. Format is the format of the dates or times in var$. Field is the number of the format element you want to extract (for example, for format mm/dd/yy, 1 corresponds to month, 2 to day, and 3 to year).</td>
</tr>
<tr>
<td>STR$(var,'format')</td>
<td>Write numeric dates or time values stored in var as characters using format.</td>
</tr>
<tr>
<td>DOC(var$, 'format')</td>
<td>Returns day of century from a character variable containing a date in the specified format.</td>
</tr>
<tr>
<td>DOC(yvar,mvar,dvar)</td>
<td>Returns day of century from year, month, and day variables (specify the arguments in this order).</td>
</tr>
<tr>
<td>DOW$(docvar)</td>
<td>Returns day of week (Monday, Tuesday, …) from numeric day of century docvar.</td>
</tr>
<tr>
<td>DAT(n,'format')</td>
<td>Returns day or time from a numeric day of century n in specified format of either Y (year), M (month), D (day), h (hour), m (minute), or s (second)—specify only one.</td>
</tr>
<tr>
<td>FDAYM</td>
<td>First day of the month in which given date occurs; FDAYM (date, format, 1).</td>
</tr>
<tr>
<td>LDAYM</td>
<td>Last day of the month in which given date occurs; LDAYM (date, format, 1).</td>
</tr>
<tr>
<td>FDAYW</td>
<td>First day of the week in which given date occurs; FDAYW (date, format, 2).</td>
</tr>
<tr>
<td>LDAYW</td>
<td>Last day of the week in which given date occurs; LDAYW (date, format, 2).</td>
</tr>
<tr>
<td>MON</td>
<td>Number of months between two dates; MON (date1, date2, format).</td>
</tr>
</tbody>
</table>
In these date and time functions, SYSTAT uses the following built-in symbols:

- **D**: Day
- **s**: Second
- **.**: Decimal point
- **M**: Month
- **m**: Minute
- **#**: A number before or after a decimal
- **Y**: Year
- **h**: Hour

Note that hours, minutes, and seconds are assumed to be coded as fractions of days. Also, the number of *m*’s indicates how the month is reported:

- **MM**: 1 through 12
- **MMM**: Jan, Feb, Mar, Apr, May, …, Dec
- **MMMMMMM**: January, February, March, April, May, …, December

### Current Time and Date: **NOW$**

The **NOW$** function gives the current date or time. You can use **NOW$** in the calculator or as part of a transformation statement. For example, if on December 25, 2006, you type

\[
\text{CALC NOW$(\text{'}\text{mm/dd/yy}\text{')} )}
\]

SYSTAT informs you that it is:

12/25/06

If you type

\[
\text{CALC NOW$(\text{'}\text{mmm dd, yyyy}\text{')} )}
\]

the result is:

Dec 25, 2006

The following are the same **NOW$** functions used in **LET** statements:

\[
\begin{align*}
\text{LET TIME$} &= \text{NOW$(\text{'}\text{hh:mm:ss}\text{')} )} \\
\text{LET DATE1$} &= \text{NOW$(\text{'}\text{mm/dd/yy}\text{')} )} \\
\text{LET DATE2$} &= \text{NOW$(\text{'}\text{mmm dd, yyyy}\text{')} )}
\end{align*}
\]
Converting Times and Dates to Numbers: VAL

The VAL function converts date or time character values to numeric values. You must specify the format of the character variable and indicate which field you want to save as a numeric variable. For example,

\[
\text{LET month} = \text{VAL(DATE$,'mm/dd/yy',1)}
\]

- In this command, \textit{DATES$} is the name of the character variable holding the dates.
- \textit{mm/dd/yy} tells SYSTAT that the first two characters of each \textit{DATES$} value represent the month. This is followed by a slash, two characters representing the day of the month, another slash, and two characters representing the year. For example, a value of \textit{DATES$} could be 03/24/04.
- The last parameter tells SYSTAT which field to read into the new variable. The 1 in this example represents the month field. So, for the date 03/24/04, SYSTAT reads the value 3 and stores it in the numeric variable \textit{MONTH}.

Converting Times or Dates to Characters: STR$

The STR$ function transforms a numeric date or time to a character value using the format you specify:

\[
\text{LET newvar$} = \text{STR$(oldvar,'format$')}
\]

SYSTAT interprets the values as days.

The LET statement,

\[
\text{LET date$} = \text{STR$(dayofcnt,'mm/dd/yyyy$')}
\]

transforms the following day-of-the-century values in \textit{DAYOFCNT} to the character dates in \textit{DATES$}:

<table>
<thead>
<tr>
<th>DAYOFCNT</th>
<th>DATES$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>01/01/1900</td>
</tr>
<tr>
<td>366</td>
<td>01/01/1901</td>
</tr>
<tr>
<td>15316</td>
<td>12/07/1941</td>
</tr>
<tr>
<td>30000</td>
<td>02/19/1982</td>
</tr>
<tr>
<td>38000</td>
<td>01/15/2004</td>
</tr>
</tbody>
</table>
**Day of the Century: DOC**

The DOC function gives the day of the century corresponding to a particular year, month, and day. SYSTAT begins counting at December 31, 1899; DOC returns a value of 0 for this date. Dates before this zero point appear as negative values. Dates after December 31, 1899 yield the number of days between the zero point and the date in question. Thus, DOC returns a value of 36525 for January 1, 2000, rather than returning a value of 1. The syntax of the DOC function is

```
DOC(year, month, day)
DOC(yvar, mvar, dvar) or DOC(var$, 'format')
```

where `year` is either a year value or a variable containing year values, `month` is a month value or a variable containing month values, and `day` is a day of the month value or a variable containing the day of the month. A month value must be the number of the month (for example, 12) rather than the name of the month (December). A year value can be a complete year (for example, 1964) or just the last two digits (64). Use the Data tab of the Options dialog in the Edit menu (or the WINDOW command) to specify the century to use for two digit years.

**Get the Day of the Week: DOW$**

The DOW$ function gives the day of the week corresponding to a given day of the century. The syntax of the DOW$ function is

```
DOW$(dayofcnt)
```

where `dayofcnt` is either a day-of-the-century value or a variable containing the day-of-the-century value.

You can embed the DOC function in the DOW$ function to find the day of the week corresponding to a particular date. For example, to find out on which day of the week October 24, 1945, fell, type:

```
CALC DOW$(DOC(1945,10,24))
```

SYSTAT responds:

Wednesday
Character Functions

SYSTAT has the following functions for manipulating character values:

- **UPR$**: Change to upper case.
- **LOW$**: Change to lower case.
- **CAP$**: Initial upper case.
- **CNT$**: Center.
- **RGT$**: Right-justify.
- **LFT$**: Left-justify.
- **SQZ$**: Remove embedded blanks.
- **SQZ$**: Remove the specified text.
- **VAL**: Convert character numbers to numeric values.
- **STR$**: Convert numbers to character values.
- **LAB$**: Extract values specified by LABEL.
- **LEN**: Returns the length of the specified string.
- **CHR$**: Returns the character for an ASCII code.
- **ISNUM**: Returns 1 if the string variable contains a number else returns 0.
- **IND**: Find the position of a character.
- **SND$**: Soundex encoding.
- **CAT$**: Join character strings.
- **MID$**: Extract characters.
- **SUB$**: Replace text1 with text2.
- **PUT$**: Replace characters.
- **RPD$**: Right-pad with character string.
- **LPD$**: Left-pad with character string.
- **ASC$**: ASCII characters.
- **ICH**: ASCII codes.

These functions can:

- Convert the values of a character variable to all upper case or all lower case, or capitalize only the first letter and make the rest of the letters lower case.
- Center, or left- or right-justify character values.
- Delete blanks or specified text from character values.
- Find the position of a particular character in each value of a character variable.
- Convert numbers to character values or character variables containing numbers to numeric variables.
- Extract text strings from or insert text strings into character values.
- Use Soundex coding to create four-digit, alphanumeric codes based on the sound of the word rather than its spelling.
- Concatenate the values of two character variables into a single variable.
- Find the ASCII code corresponding to a character or find the character corresponding to the ASCII code.
- Finds the length of a specified string.
- Finds the character corresponding to the ASCII code.
- Checks for numbers in the string variable.

In this version of SYSTAT, results of operations on character variables are limited to 256 characters, or leading characters relevant to the function.

**Changing Case: UPR$, LOW$, and CAP$**

The UPR$, LOW$, and CAP$ functions allow you to change the case of character values.

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPR$(var)$</td>
<td>Change the values of var to all capitals.</td>
</tr>
<tr>
<td>LOW$(var)$</td>
<td>Change the values of var to all lower case.</td>
</tr>
<tr>
<td>CAP$(var)$</td>
<td>Capitalize the first letter of each var value. Make the remaining letters of each value lower case.</td>
</tr>
</tbody>
</table>

**Justifying Character Values: CNT$, RGT$, and LFT$**

The CNT$, RGT$, and LFT$ functions center and justify character values.

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNT$(var,n)$</td>
<td>Center values of variable var, where n is width of character field. If you do not specify n, SYSTAT assumes a width of 72.</td>
</tr>
<tr>
<td>CNT$(text')$</td>
<td>Center text in a 72-character field.</td>
</tr>
<tr>
<td>RGT$(var,n)$</td>
<td>Right-justify values of variable var, where n is width of character field. If you do not specify n, SYSTAT assumes a width of 72.</td>
</tr>
<tr>
<td>RGT$(text')$</td>
<td>Right-justify text in a 72-character field.</td>
</tr>
<tr>
<td>LFT$(var)$</td>
<td>Left-justify values of var.</td>
</tr>
<tr>
<td>LFT$(text')$</td>
<td>Left-justify text.</td>
</tr>
</tbody>
</table>

SYSTAT assumes that each character field is 72 characters wide, which centers text on the screen or page. Internally, transformations use 72 characters.
Deleting Characters: SQZ$

Use SQZ$ to delete blanks, commas, apostrophes, or other characters from character values.

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQZ$(var$)</td>
<td>Remove all embedded blanks.</td>
</tr>
<tr>
<td>SQZ$(var$,'text')</td>
<td>Remove text from the values of var$.</td>
</tr>
</tbody>
</table>

Locating a Character in a String: IND

The IND function locates a particular character in each value of a character variable.

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>IND(var$,'char')</td>
<td>Find the position of the first occurrence of char in each value of var$.</td>
</tr>
</tbody>
</table>

For example, the following LET statement:

```
LET position = IND(team$,'e')
```

finds the character position of the letter $e$ in each team name. The POSITION values that SYSTAT returns for a given value of TEAM$ are:

<table>
<thead>
<tr>
<th>TEAM$</th>
<th>POSITION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mets</td>
<td>2</td>
</tr>
<tr>
<td>Yankees</td>
<td>5</td>
</tr>
<tr>
<td>Blue Jays</td>
<td>4</td>
</tr>
<tr>
<td>Cubs</td>
<td>0</td>
</tr>
</tbody>
</table>

Notice that SYSTAT returns the position of the first $e$ in Yankees.

Converting Numbers to Characters and Vice Versa: VAL and STR$

Although SYSTAT allows you to type numbers as values of character variables, the numbers are considered characters and cannot be used in calculations. You can use the VAL function to convert the values into numbers. Conversely, you can use the STR$ function to convert numbers to characters.
Suppose you entered years as numbers but want to use the values to label points in a plot. First, convert the numbers to characters and store them in the variable `YEAR$`:

\[
\text{LET } \text{year$} = \text{STR$}(\text{year})
\]

Then specify `YEAR$` as the label. For example:

\[
\text{PLOT } \text{yvar}$*\text{xvar} / \text{LABEL}=$\text{year}$
\]

Now suppose, conversely, that you enter the year as characters and then want to do calculations. Use the VAL function to convert the character values to numbers:

\[
\text{LET } \text{year} = \text{VAL}($\text{year}$)
\]

**Justifying Numeric Values: ISNUM**

The ISNUM function checks for numeric values.

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISNUM(var$)</td>
<td>Returns 1 if argument is a number, else zero.</td>
</tr>
</tbody>
</table>

The following example can be used to illustrate the ISNUM function when the command:

\[
\text{LET num}=\text{ISNUM} (\text{var$}) \text{ is given.}
\]

<table>
<thead>
<tr>
<th>VARS</th>
<th>NUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
</tr>
</tbody>
</table>
**Extracting and Inserting Characters**

SYSTAT has several functions for manipulating character strings.

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>MID$(var$,p,j)</td>
<td>Extracts a string of $j$ characters from each value of $var$, beginning with the $p^{th}$ character.</td>
</tr>
<tr>
<td>SUB$(var$,'text1','text2')</td>
<td>Replaces $text1$ with $text2$.</td>
</tr>
<tr>
<td>PUT$(var$,'text',p,j)</td>
<td>Beginning at the $p^{th}$ character of $var$, replace the next $j$ characters with the first $j$ characters of $text$.</td>
</tr>
<tr>
<td>RPD$(var$,'char')</td>
<td>Right-pad the values of $var$ with $char$.</td>
</tr>
<tr>
<td>LPD$(var$,'char')</td>
<td>Left-pad the values of $var$ with $char$.</td>
</tr>
</tbody>
</table>

**Concatenating Character Strings: CAT$**

Use the CAT$ function to join the values of two variables into a single variable.

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAT$(var1$,var2$)</td>
<td>Join contents of $var1$ with $var2$, eliminating trailing blanks.</td>
</tr>
</tbody>
</table>

In this version of SYSTAT, the result is limited to 256 characters.

**Character codes: CHR$**

The CHR$ function returns the character for an ASCII code.

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHR$(int)$</td>
<td>Character corresponding to the integer int.</td>
</tr>
<tr>
<td>CHR$(var)$</td>
<td>Character corresponding to the value in $var$.</td>
</tr>
</tbody>
</table>

The following table shows some ASCII codes and their characters:

<table>
<thead>
<tr>
<th>ASCII code</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>65</td>
<td>A</td>
</tr>
<tr>
<td>97</td>
<td>a</td>
</tr>
<tr>
<td>51</td>
<td>3</td>
</tr>
<tr>
<td>63</td>
<td>?</td>
</tr>
<tr>
<td>35</td>
<td>#</td>
</tr>
<tr>
<td>125</td>
<td>}</td>
</tr>
</tbody>
</table>
**ASCII Codes: ICH**

The ICH function gives the ASCII code for a character.

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICH('char')</td>
<td>ASCII code corresponding to the character char.</td>
</tr>
<tr>
<td>ICH(var$)</td>
<td>ASCII code corresponding to the value in var$.</td>
</tr>
<tr>
<td>ASC$(int)</td>
<td>ASCII character corresponding to the integer int.</td>
</tr>
<tr>
<td>ASC$(var)</td>
<td>ASCII character corresponding to the value in var.</td>
</tr>
</tbody>
</table>

The following table shows some characters and their ASCII codes:

<table>
<thead>
<tr>
<th>Character</th>
<th>ASCII code</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>65</td>
</tr>
<tr>
<td>a</td>
<td>97</td>
</tr>
<tr>
<td>3</td>
<td>51</td>
</tr>
<tr>
<td>?</td>
<td>63</td>
</tr>
<tr>
<td>#</td>
<td>35</td>
</tr>
<tr>
<td>}</td>
<td>125</td>
</tr>
</tbody>
</table>

So, for example, ICH('3') = 51 and ASC$(35) = #.

To obtain a table of ASCII codes, specify:

```plaintext
NEW
REPEAT 1
FOR I = 1 to 256
  LET A$ = ASC$(I)
  PRINT I, A$
NEXT
```

**Using Soundex to Code Names: SND$**

The Soundex encoding function (SND$) allows you to create an alphanumeric code of a word based on how the word sounds rather than its spelling. The SND$ function can be used to create a new variable to select similarly spelled duplicate records.

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>SND$(var$)</td>
<td>An alphanumeric Soundex code of var$.</td>
</tr>
</tbody>
</table>
The SND$ function uses a formula to replace letters with numbers in an $Xnnn$ format, where $X$ is the letter with which the word begins and $nnn$ is up to three numbers indicating the “sound” of the rest of the word.

Soundex ignores vowels ($A$, $E$, $I$, $O$, $U$) and the consonants $H$, $W$, and $Y$. It then codes for the remaining letters:

- B, F, P, V: 1
- C, G, J, K, Q, S, X, Z: 2
- D, T: 3
- L: 4
- M, N: 5
- R: 6

If the same code occurs consecutively, the second occurrence is skipped. Some names with their Soundex codes are:

- Berry: B6
- Wilson: W425
- Anderson: A536
- Johnson: J525
- Smith: S53
- Carlson: C642

For a quick way to see the Soundex code of a name, use SYSTAT’s calculator:

```calc
CALC SND$('Smith')
```

SYSTAT returns:

```calc
S53
```

Using LEN to find the length of a string

The length (LEN) is used to find the number of characters in a string. The numbers of spaces between the characters are considered to find the length of a string.

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>LEN(name$)</td>
<td>Length of name$.</td>
</tr>
</tbody>
</table>
### Functions Relating to Probability Distributions

An important aspect of statistical analysis involves the calculation of values of the **probability mass function** (pmf), **probability density function** (pdf), **cumulative distribution function** (cdf), and **inverse of cdf** (cut-off points) relating to various standard distributions and drawing random samples from them. SYSTAT computes these functions called by $CF, $DF, $IF and $RN respectively with '$' denoting a symbol(s) for the distribution (e.g., Z for normal, T for Student's t, DE for double-exponential etc.) for 43 discrete and continuous distributions.

The result of the density function is the height at $x$ of the ordinate under the density curve of the specified distribution. Cumulative distribution functions compute the probability that a random value from the specified distribution falls below or is equal to the given value. **Inverse distribution functions** take a specified alpha (a probability value between 0 and 1) and return the critical value $x$, below which lies that proportion of the specified distribution. **Random variate functions** generate pseudo-random variates from the specified distribution.

<table>
<thead>
<tr>
<th>Distribution</th>
<th>Cumulative</th>
<th>Density</th>
<th>Inverse</th>
<th>Random data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uniform(0,1)</td>
<td>UCF (x,low,hi)</td>
<td>UDF (x,low,hi)</td>
<td>UIF (a,low,hi)</td>
<td>URN (low,hi)</td>
</tr>
<tr>
<td>Normal (0,1)</td>
<td>ZCF (z,loc,sc)</td>
<td>ZDF (z,loc,sc)</td>
<td>ZIF (a,loc,sc)</td>
<td>ZRN (loc,sc)</td>
</tr>
<tr>
<td>t</td>
<td>TCF (t,df)</td>
<td>TDF (t,df)</td>
<td>TIF (a,df)</td>
<td>TRN (df)</td>
</tr>
<tr>
<td>F</td>
<td>FCF (F,df1,df2)</td>
<td>FDF (F,df1,df2)</td>
<td>FIF (a,df1,df2)</td>
<td>FRN (df1,df2)</td>
</tr>
<tr>
<td>Chi-Square</td>
<td>XCF ($\chi^2$,df)</td>
<td>XDF ($\chi^2$,df)</td>
<td>XIF (a,df)</td>
<td>XRN (df)</td>
</tr>
<tr>
<td>Gamma</td>
<td>GCF (x,shp,sc)</td>
<td>GDF (x,shp,sc)</td>
<td>GIF (a,shp,sc)</td>
<td>GRN (shp,sc)</td>
</tr>
<tr>
<td>Beta</td>
<td>BCF (x,shp1,shp2)</td>
<td>BDF (x,shp1,shp2)</td>
<td>BIF (a,shp1,shp2)</td>
<td>BRN (shp1,shp2)</td>
</tr>
<tr>
<td>Exponential (0,1)</td>
<td>ECF (x,loc,sc)</td>
<td>EDF (x,loc,sc)</td>
<td>EIF (a,loc,sc)</td>
<td>ERN (loc,sc)</td>
</tr>
<tr>
<td>Logistic (0,1)</td>
<td>LCF (x,loc,sc)</td>
<td>LDF (x,loc,sc)</td>
<td>LIF (a,loc,sc)</td>
<td>LRN (loc,sc)</td>
</tr>
<tr>
<td>Studentized</td>
<td>SCF (x,k,df)</td>
<td>SDF (x,k,df)</td>
<td>SIF (a,k,df)</td>
<td>SRN (k,df)</td>
</tr>
<tr>
<td>Weibull</td>
<td>WCF (x,sc,shp)</td>
<td>WDF (x,sc,shp)</td>
<td>WIF (a,sc,shp)</td>
<td>WRN (sc,shp)</td>
</tr>
<tr>
<td>Cauchy</td>
<td>CCF (x,loc,sc)</td>
<td>CDF (x,loc,sc)</td>
<td>CIF (a,loc,sc)</td>
<td>CRN (loc,sc)</td>
</tr>
<tr>
<td>Double exponential</td>
<td>DECF (x,loc,sc)</td>
<td>DEDF (x,loc,sc)</td>
<td>DEIF (a,loc,sc)</td>
<td>DERN (loc,sc)</td>
</tr>
<tr>
<td>Gompertz</td>
<td>GOCF (x,b,c)</td>
<td>GODF (x,b,c)</td>
<td>GOIF (a,b,c)</td>
<td>GORN (b,c)</td>
</tr>
<tr>
<td>Gumbel</td>
<td>GUCF (x,loc,sc)</td>
<td>GUDF (x,loc,sc)</td>
<td>GUF (a,loc,sc)</td>
<td>GURN (loc,sc)</td>
</tr>
<tr>
<td>Inverse Gaussian(Wald)</td>
<td>IGCF (x,loc,sc)</td>
<td>IGDF (x,loc,sc)</td>
<td>IGIF (a,loc,sc)</td>
<td>IGRN (x,loc,sc)</td>
</tr>
<tr>
<td>Logit normal</td>
<td>ENCF (x,loc,sc)</td>
<td>ENDF (x,low,hi)</td>
<td>ENIF (a,loc,sc)</td>
<td>ENRN (loc,sc)</td>
</tr>
<tr>
<td>Lognormal</td>
<td>LNCF (x,loc,sc)</td>
<td>LNDF (x,loc,sc)</td>
<td>LNIF (a,loc,sc)</td>
<td>LNRN (loc,sc)</td>
</tr>
<tr>
<td>Pareto</td>
<td>PACF (x,thr,shp)</td>
<td>PADF (x,thr,shp)</td>
<td>PAIF (a,thr,shp)</td>
<td>PARN (thr,shp)</td>
</tr>
<tr>
<td>Rayleigh</td>
<td>RCF (x,sc)</td>
<td>RDF (x,sc)</td>
<td>RIF (a,sc)</td>
<td>RRN (sc)</td>
</tr>
<tr>
<td>Triangular</td>
<td>TRCF (x,a,b,c)</td>
<td>TRDF (x,a,b,c)</td>
<td>TRIF (a,a,b,c)</td>
<td>TRRN (a,b,c)</td>
</tr>
<tr>
<td>Loglogistic</td>
<td>LOCF (x,logsc,shp)</td>
<td>LODF (x,logsc,shp)</td>
<td>LOIF (a,logsc,shp)</td>
<td>LORN (logsc,shp)</td>
</tr>
</tbody>
</table>
Here, \( \text{low} \) is the smallest value and \( \text{hi} \) the largest value; \( \text{loc} \) is the location parameter and \( \text{sc} \), the scale parameter; \( \text{shp} \) is the shape parameter and \( \text{thr} \), the threshold parameter; and finally, \( \text{df} \) is the degrees of freedom. If \( \text{low} \), \( \text{hi} \), \( \text{loc} \), or \( \text{sc} \) are omitted, the default values, which are displayed in the Distribution column, are assumed.

### Expressions for the Functions

**Cumulative distribution function** (CDF) for discrete distributions is calculated by

\[
P(X \leq x) = \sum_{y \leq x} PDF(y)
\]

**Inverse cumulative distribution function** for discrete distributions is obtained by the smallest \( x \) such that

<table>
<thead>
<tr>
<th>Distribution</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Erlang</td>
<td>( \text{ERCF} (x, \text{shp}, \text{sc}) )</td>
</tr>
<tr>
<td>Non-central Chi-square</td>
<td>( \text{NXCF} (x, \text{df}, \text{df}) )</td>
</tr>
<tr>
<td>Non-central F</td>
<td>( \text{NFCF} (x, \text{df}, \text{df}, \text{df}) )</td>
</tr>
<tr>
<td>Non-central t</td>
<td>( \text{NTCF} (x, \text{df}, \text{df}) )</td>
</tr>
<tr>
<td>Smallest extreme value</td>
<td>( \text{SECF} (x, \text{loc}, \text{sc}) )</td>
</tr>
<tr>
<td>Studentized maximum modulus</td>
<td>( \text{SMC} (x, \text{df}, \text{df}) )</td>
</tr>
<tr>
<td>Binomial</td>
<td>( \text{BECF} (x, \text{p}) )</td>
</tr>
<tr>
<td>Poisson</td>
<td>( \text{PCF} (x, \text{p}) )</td>
</tr>
<tr>
<td>Discrete uniform</td>
<td>( \text{DUCF} (x, \text{N}) )</td>
</tr>
<tr>
<td>Geometric</td>
<td>( \text{GECF} (x, \text{p}) )</td>
</tr>
<tr>
<td>Hypergeometric</td>
<td>( \text{HCF} (x, \text{N}, \text{m}, \text{n}) )</td>
</tr>
<tr>
<td>Negative binomial</td>
<td>( \text{NBCF} (x, \text{k}, \text{p}) )</td>
</tr>
<tr>
<td>Benford's Law</td>
<td>( \text{BLCF} (x, \text{B}) )</td>
</tr>
<tr>
<td>Logarithmic series</td>
<td>( \text{LSCF} (x, \theta) )</td>
</tr>
<tr>
<td>Zipf</td>
<td>( \text{ZICF} (x, \text{shp}) )</td>
</tr>
<tr>
<td>Beta general</td>
<td>( \text{BGCF} (x, \text{shp}1, \text{shp}2, \text{min}, \text{max}) )</td>
</tr>
<tr>
<td>Pert</td>
<td>( \text{PECF} (x, \text{min}, \text{max}, \text{most}) )</td>
</tr>
<tr>
<td>Pearson Type VI</td>
<td>( \text{P6CF} (x, \text{shp}1, \text{shp}2, \text{beta}) )</td>
</tr>
<tr>
<td>Pearson Type V</td>
<td>( \text{P5CF} (x, \text{shp}, \text{sc}) )</td>
</tr>
<tr>
<td>General discrete</td>
<td>( \text{GDCF} (x, \text{x} \text{or numvar}) )</td>
</tr>
<tr>
<td>Empirical</td>
<td>( \text{EMCF} (x, \text{x} \text{or numvar}) )</td>
</tr>
</tbody>
</table>

Here, \( \leq \) is the smallest value and \( \geq \) the largest value; \( \sum \) is the sum.
\[ \sum_{y \leq x} PDF(y) \geq p \]

In expressions below, the following will hold:
- \( G \) denotes Gamma function.
- \( p \in (0, 1) \) for calculating inverse cumulative distribution function.
- Pdf is zero when it is not specified.

**Univariate Discrete Distributions**

- **Benford’s law**

  PDF:
  \[
  f(x) = \begin{cases} 
  \frac{\ln(1 + 1/x)}{\ln(B)} & x \in \{1, 2, 3, ..., B-1\} \\
  0 & \text{otherwise}
  \end{cases}
  \]

  Parameter(s):
  - \( B \): Base ---\{3,4, ..., 16\}

  CDF:
  \[
  F(x) = \begin{cases} 
  0 & x < 1 \\
  \frac{\ln(1 + \lfloor x \rfloor)}{\ln(B)} & 1 \leq x < B-1 \\
  1 & x \geq B-1
  \end{cases}
  \]

  Inverse CDF:
  \[
  F^{-1}(p) = \lfloor \exp\{p \ln(B)\} \rfloor, \text{ where } [\ast] \text{ is integer part }\]
■ Binomial distribution

PDF:

\[ f(x) = \frac{n!}{x!(n-x)!} p^x (1-p)^{n-x} \quad x \in \{0, 1, 2, ..., n\} \]

Parameter(s):
- \( n \): number of trials --- \{1, 2, 3, ...\}
- \( p \): probability of success --- (0, 1)

■ Discrete uniform distribution

PDF:

\[ f(x) = \frac{1}{N} \quad x \in \{1, 2, 3, ..., N\} \]

Parameter(s):
- \( N \): number of points --- \{1, 2, 3, ...\}

■ General discrete distribution

PDF:

\[ f(x) = p_i \quad \text{for } x = x_i \]

\[ f(x) = 0 \quad \text{for } x \notin \{x\} \]

■ Geometric distribution

PDF:

\[ f(x) = p(1-p)^x \quad x \in \{0, 1, 2, ...\} \]

Parameter(s):
- \( p \): probability of success --- (0, 1)
# Hypergeometric distribution

PDF:

\[
f(x) = \frac{m! (N-m)!}{x!(m-x)!(n-x)!(N-m-n+x)!} \frac{N!}{n!(N-n)!} \\
\text{max}(0,n-N+m) \leq x \leq \min(n,m)
\]

Parameter(s):
- \(N\): population size --- \(\{1,2,3,\ldots\}\)
- \(m\): number of units having some specific property --- \(\{1,2,3,\ldots\}\)
- \(n\): sample size --- \(\{1,2,3,\ldots\}\)

\(N \geq \max(m,n)\)

# Logarithmic series distributions

PDF:

\[
f(x) = \frac{a \theta^x}{x}, \quad x = 1,2,3,\ldots \text{, where } a = 1 \ln \left( \frac{1}{1-\theta} \right)
\]

Parameter(s):
- \(\theta\) --- \((0,1)\)

# Negative binomial distribution

PDF:

\[
f(x) = \binom{k+x}{x} p^k (1-p)^x \quad x \in \{0,1,2,\ldots\}
\]

Parameter(s):
- \(k\) --- \((0,\infty)\)
- \(p\): probability --- \((0,1)\)
■ Poisson distribution

PDF:

\[ f(x) = \frac{\lambda^x e^{-\lambda}}{x!} \quad x \in \{0, 1, 2, \ldots\} \]

Parameter(s):
\( \lambda \): mean \(-(0, \infty)\)

■ Zipf distribution

PDF:

\[ f(x) = \left[ \sum_{x=1}^{\infty} \frac{1}{x^{a+1}} \right]^{-1} x^{-(a+1)} \quad x \in \{1, 2, 3, \ldots\} \]

Parameter(s):
\( a \): shape \(-(0, \infty)\)

Univariate continuous distributions

For some continuous distributions like Normal, Gamma, t, F, etc., the CDF and Inverse CDF do not have closed form expressions. In such cases, the CDF and Inverse CDF are computed using numerical methods.

■ Beta distribution

PDF:

\[ f(x) = \frac{\Gamma(\alpha + \beta)}{(\Gamma \alpha)(\Gamma \beta)} x^{\alpha-1} (1 - x)^{\beta-1} \quad 0 < x < 1 \]

Parameter(s):
\( \alpha \): shape 1 \(-(0, \infty)\)
\( \beta \): shape 2 \(-(0, \infty)\)
**Beta general distribution**

PDF:

\[
f(x) = \frac{(x - \text{min})^{\alpha_1 - 1} (\text{max} - x)^{\alpha_2 - 1}}{B(\alpha_1, \alpha_2)(\text{max} - \text{min})^{\alpha_1 + \alpha_2 - 1}}
\]

where \(B\) is the *Beta Function* and \(B_x\) is the *Incomplete Beta Function*

- **Cauchy distribution**

PDF:

\[
f(x) = \frac{1}{\pi \beta} \left( \frac{1}{x^2 + \beta^2} \right) \quad -\infty < x < +\infty
\]

Parameter(s):
- \(\alpha\): location \((-\infty, \infty)\)
- \(\beta\): scale \((0, \infty)\)

CDF:

\[
F(x) = \frac{1}{2} + \frac{1}{\pi} \tan^{-1}\left(\frac{x - \alpha}{\beta}\right)
\]

Inverse CDF:

\[
F^{-1}(p) = \alpha + \beta \tan(\pi \cdot p - 0.5)
\]

- **Chi-square distribution**

PDF:

\[
f(x) = \frac{1}{\Gamma(n/2)2^{n/2}} x^{(n/2) - 1} \exp\left(-\frac{x}{2}\right) \quad x > 0
\]

Parameter(s):
- \(n\): degrees of freedom \(\{1, 2, 3, \ldots\}\)
Chapter 4

- **Double exponential (Laplace) distribution**

PDF:

\[ f(x) = (2\phi)^{-1} \exp\left\{ -\frac{|x-\theta|}{\phi} \right\} \quad -\infty < x < +\infty \]

Parameter(s):

- \( \theta \): location \(-\infty, \infty\)
- \( \phi \): scale \((0, \infty)\)

CDF:

\[
F(x) = \begin{cases} 
0.5 \exp\left\{ -\frac{\theta-x}{\phi} \right\} & x \leq \theta \\
1 - 0.5 \exp\left\{ -\frac{x-\theta}{\phi} \right\} & x > \theta 
\end{cases}
\]

Inverse CDF:

\[
F^{-1}(p) = \begin{cases} 
\theta + \phi \ln(2p) & p \leq 0.5 \\
\theta - \phi \ln(2(1-p)) & p > 0.5 
\end{cases}
\]

- **Erlang distribution**

PDF:

\[ f(x) = \frac{1}{\Gamma \alpha \beta^\alpha} x^{\alpha-1} \exp\left\{ -\frac{x}{\beta} \right\} \quad x > 0 \]

Parameter(s):

- \( \alpha \): shape \{1, 2, 3, ...\}
- \( \beta \): scale \((0, \infty)\)
- **Exponential distribution**

  PDF:

  \[ f(x) = \lambda^{-1} \exp\left\{-(\frac{x-\theta}{\lambda})\right\} \quad x \geq \theta \]

  Parameter(s):
  
  \( \theta \): location \( (\infty, \infty) \)
  
  \( \lambda \): scale \( (0, \infty) \)

  CDF:

  \[ F(x) = \begin{cases} 
  0 & x \leq \theta \\
  1 - \exp\left\{-(\frac{x-\theta}{\lambda})\right\} & x > \theta 
  \end{cases} \]

  Inverse CDF:

  \[ F^{-1}(p) = \theta - \lambda \ln(1 - p) \]

- **F distribution**

  PDF:

  \[ f(x) = \frac{\{\Gamma\left(\frac{n_1 + n_2}{2}\right)\} \left(\frac{n_1}{2}\right)^{n_1/2} \left(\frac{n_2}{2}\right)^{(n_1+n_2)/2} \left(1 + \frac{n_1}{n_2} x\right)^{-\frac{n_1}{2} - 1}}{\Gamma\left(\frac{n_1}{2}\right) \Gamma\left(\frac{n_2}{2}\right)} \quad x > 0 \]

  Parameter(s):
  
  \( n_1 \): numerator degrees of freedom \( \{1,2,3, \ldots\} \)
  
  \( n_2 \): denominator degrees of freedom \( \{1,2,3, \ldots\} \)
- **Gamma distribution**

PDF:

\[
f(x) = \frac{1}{\beta^\alpha \Gamma(\alpha)} x^{\alpha-1} \exp\{-(\frac{x}{\beta})\} \quad x > 0
\]

Parameter(s):
- \(\alpha\): shape \(-(0, \infty)\)
- \(\beta\): scale \(-(0, \infty)\)

- **Generalized lambda distribution**

Inverse CDF:

\[
F^{-1}(p) = \lambda_1 + \frac{p^{\lambda_3} - (1-p)^{\lambda_4}}{\lambda_2}, \quad 0 \leq p \leq 1
\]

Parameter(s):
- \(\lambda_1\): location \(-(-\infty, \infty)\)
- \(\lambda_2, \lambda_3, \) and \(\lambda_4\) are such that

\[
\frac{\lambda_2}{\lambda_3 y^{\lambda_3-1} + \lambda_4 (1-y)^{\lambda_4-1}} \geq 0 \text{ for all } y \in [0, 1] \text{ and } \lambda_2 \neq 0
\]

- **Gompertz distribution**

PDF:

\[
f(x) = b c^x \exp\{-(\frac{b}{\ln c} (c^x - 1))\} \quad x \geq 0
\]

Parameter(s):
- \(b\): \(-(0, \infty)\)
- \(c\): \(-(1, \infty)\)

CDF:

\[
F(x) = \begin{cases} 
0 & x \leq 0 \\
1 - \exp\{-(\frac{b}{\ln c} (c^x - 1))\} & x > 0
\end{cases}
\]
Data Transformations

Inverse CDF:

\[ F^{-1}(p) = \frac{\ln(1 - \frac{\ln c \ln(1 - p)}{b})}{\ln c} \]

- **Gumbel distribution**

PDF:

\[ f(x) = \frac{1}{\theta} \exp\left\{-\left(\frac{x-\alpha}{\theta}\right)\right\} \exp\left\{-\exp\left\{-\left(\frac{x-\alpha}{\theta}\right)\right\}\right\} \quad -\infty < x < +\infty \]

Parameter(s):

- \( \alpha \): location \(-\infty, \infty\)
- \( \theta \): scale \((0, \infty)\)

CDF:

\[ F(x) = \exp\left\{-\exp\left\{-\left(\frac{x-\alpha}{\theta}\right)\right\}\right\} \]

Inverse CDF:

\[ F^{-1}(p) = \alpha - \theta \ln(-\ln p) \]

- **Inverse Gaussian (Wald) distribution**

PDF:

\[ f(x) = \left(\frac{\lambda}{2x^3\pi}\right)^{\frac{1}{2}} \exp\left\{-\frac{\lambda}{2\mu^2x}(x-\mu)^2\right\} \quad x > 0 \]

Parameter(s):

- \( \mu \): location \((0, \infty)\)
- \( \lambda \): scale \((0, \infty)\)
CDF:

\[
F(x) = \begin{cases} 
0 & \text{if } x \leq 0 \\
\Phi\left(\frac{\sqrt{2\lambda} \left(\frac{x - \mu}{\sigma} - 1\right)}{\lambda} + \exp\left\{\frac{2\lambda}{\mu}\right\} \Phi\left(-\sqrt{2\lambda} \left(\frac{x + 1}{\mu}\right)\right) & \text{if } x > 0
\end{cases}
\]

where \(\Phi(x)\) denotes the CDF of normal(0,1) distribution.

- **Logistic distribution**

PDF:

\[
f(x) = \beta^{-1} \exp\left\{-\frac{\left(x - \frac{\alpha}{\beta}\right)}{\beta}\right\} [1 + \exp\left\{-\frac{\left(x - \frac{\alpha}{\beta}\right)}{\beta}\right\}]^{-2} \quad -\infty < x < +\infty
\]

Parameter(s):

\(\alpha\) : location \(-\infty, \infty\)

\(\beta\) : scale \(0, \infty\)

CDF:

\[
F(x) = [1 + \exp\left\{-\frac{\left(x - \frac{\alpha}{\beta}\right)}{\beta}\right\}]^{-1}
\]

Inverse CDF:

\[
F^{-1}(p) = \alpha - \beta \ln\left(\frac{1}{p} - 1\right)
\]

- **Logit normal distribution**

PDF:

\[
f(x) = \frac{1}{\sqrt{2\pi} \sigma x (1-x)} \exp\left\{-\frac{1}{2} \left(\frac{\ln\left(\frac{x}{1-x}\right) - \mu}{\sigma}\right)^2\right\} \quad 0 < x < 1
\]

Parameter(s):

\(\mu\) : location \(-\infty, \infty\)

\(\sigma\) : scale \(0, \infty\)
- **Loglogistic distribution**

PDF:

\[
f(x) = \frac{1}{\beta x} \exp\left\{-\left(\frac{\ln x - \alpha}{\beta}\right)\right\} \frac{1}{(1 + \exp\left\{-\left(\frac{\ln x - \alpha}{\beta}\right)\right\})^2} \quad x > 0
\]

Parameter(s):

\(\alpha\) : log of scale --- \((-\infty, \infty)\)

\(\beta\) : shape --- \((0, \infty)\)

- **Lognormal distribution**

PDF:

\[
f(x) = \frac{1}{\sqrt{2\pi} \sigma x} \exp\left\{-\frac{1}{2} \left(\frac{\ln x - \mu}{\sigma}\right)^2\right\} \quad x > 0
\]

Parameter(s):

\(\mu\) : location --- \((-\infty, \infty)\)

\(\sigma\) : scale --- \((0, \infty)\)

- **Normal distribution**

PDF:

\[
f(x) = \frac{1}{\sqrt{2\pi} \sigma} \exp\left\{-\frac{1}{2} \left(\frac{x - \mu}{\sigma}\right)^2\right\} \quad -\infty < x < +\infty
\]

Parameter(s):

\(\mu\) : location --- \((-\infty, \infty)\)

\(\sigma\) : scale --- \((0, \infty)\)
Chapter 4

- **Non-central chi-square distribution**

  PDF:
  \[
  f(x) = \frac{\exp\left\{-\frac{1}{2}(x+\delta)\right\}}{\frac{n}{2^\frac{n}{2}}} \sum_{j=0}^{\infty} \left( \frac{n_{x^2+j-1}}{\Gamma\left(\frac{n}{2} + j\right)2^{2j}j!} \right) x > 0
  \]

  Parameter(s):
  
  \( n \): degree of freedom---\((0, \infty)\)
  
  \( \delta \): non-centrality parameter---\((0, \infty)\)

- **Non-central F distribution**

  PDF:
  \[
  f(x) = \sum_{j=0}^{\infty} e^{-\delta} \delta^j \frac{\Gamma\left(\frac{n_1+n_2}{2} + j\right)}{j! \Gamma(n_1/2+j)\Gamma(n_2/2)} \left(\frac{n_1}{n_2}\right)^{\frac{n_1}{2}+j} \frac{x^{\frac{n_1}{2}+j-1}}{(1 + \frac{n_1}{n_2}x)^{\frac{n_1+n_2}{2}+j}} x > 0
  \]

  Parameter(s):
  
  \( n_1 \): numerator degree of freedom---\((0, \infty)\)
  
  \( n_2 \): denominator degree of freedom---\((0, \infty)\)
  
  \( \delta \): non-centrality parameter---\((0, \infty)\)

- **Non-central t distribution**

  PDF:
  \[
  f(x) = \frac{1}{\sqrt{n\pi}} \sum_{j=0}^{\infty} e^{-\delta^2/2} (\delta x)^j \frac{2^j}{j!} \frac{\Gamma\left(\frac{n+j+1}{2}\right)}{\left(\frac{n}{2}\right) \Gamma(n/2)} \frac{1}{\left(1 + \frac{x^2}{n}\right)^{(n+j+1)/2}} - \infty < x < \infty
  \]

  Parameter(s):
  
  \( n \): degree of freedom---\((0, \infty)\)
  
  \( \delta \): non-centrality parameter---\((-\infty, \infty)\)
- Pareto distribution

PDF:
\[
f(x) = \frac{\beta \alpha^\beta}{x^{\beta+1}} \quad x \geq \alpha
\]

Parameter(s):
- \( \alpha \): threshold \(-(0, \infty)\)
- \( \beta \): shape \(-(0, \infty)\)

CDF:
\[
F(x) = \begin{cases} 
0 & x \leq \alpha \\
1 - \left(\frac{\alpha}{x}\right)^\beta & x > \alpha 
\end{cases}
\]

Inverse CDF:
\[
F^{-1}(p) = \frac{\alpha}{(1 - p)^{1/\beta}}
\]

- Pert Distribution

PDF:
\[
f(x) = \frac{1}{B(\alpha_1, \alpha_2)} \frac{(X - a)^{\alpha_1 - 1} (b - x)^{\alpha_2 - 1}}{(b - a)^{\alpha_1 + \alpha_2 - 1}}
\]

Parameter(s):
- M (most likely value) - mode parameter \((a \leq m \leq b)\)
- a,b - boundary parameters \((a < b)\)
  - Range is
  - \(a \leq x \leq b\)
The Pert distribution is a special case of the Beta distribution specified by the following parameters:
\[ \alpha_1 = \frac{(4m + b - 5a)}{b - a} \]
\[ \alpha_2 = \frac{(5b - a - 4m)}{b - a} \]

**Pearson type V distribution**

PDF:
\[ f(x) = \frac{1}{\beta \Gamma(\alpha)} \cdot \frac{e^{-\beta/x}}{(x/\beta)^{\alpha + 1}} \]

**Pearson Type VI distribution**

\[ f(x) = \frac{1}{\beta B(\alpha_1, \alpha_2)} \times \left(\frac{x/\beta}{1 + x/\beta}\right)^{\alpha_1 - 1} \]

- **Rayleigh distribution**

PDF:
\[ f(x) = \frac{1}{\sigma^2} x \exp\{-\left(\frac{x^2}{2\sigma^2}\right)\} \quad x \geq 0 \]

Parameter(s):
- \( \sigma \): scale \( (0, \infty) \)

CDF:
\[ F(x) = \begin{cases} 
0 & x \leq 0 \\
1 - \exp\{-\left(\frac{x^2}{2\sigma^2}\right)\} & x > 0 
\end{cases} \]

Inverse CDF:
\[ F^{-1}(p) = \sigma \left(-2 \ln(1 - p)\right)^{\frac{1}{2}} \]
Data Transformations

- **Smallest extreme value distribution**

PDF:

\[ f(x) = \frac{1}{\theta} \exp\left\{ \left( \frac{x-\alpha}{\theta} \right) - \exp\left\{ \left( \frac{x-\alpha}{\theta} \right) \right\} \right\} \quad -\infty < x < +\infty \]

Parameter(s):

\( \alpha : \text{location} \quad (-\infty, \infty) \)
\( \theta : \text{scale} \quad (0, \infty) \)

CDF:

\[ F(x) = 1 - \exp\left\{ -\exp\left\{ \frac{x-\alpha}{\theta} \right\} \right\} \quad +\infty < x < +\infty \]

Inverse CDF:

\[ F^{-1}(p) = \alpha + \theta \ln(-\ln(1 - p)) \]

- **Studentized maximum modulus distribution**

PDF:

\[ F(x) = \int_0^\infty [2\Phi(xy) - 1]^k g(y, \nu) \, dy \]

where

\[ \Phi(t) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^t \exp\{ -z^2 / 2 \} \, dz \]

and \( g(y, \nu) \) is the PDF of \( Y \), where \( \nu Y^2 \) is chi-squared distributed with \( \nu \) degrees of freedom.

Parameter(s):

\( k : \text{sample size} \quad \{2, 3, \ldots\} \)
\( \nu: \text{degrees of freedom} \quad \{1, 2, \ldots\} \)
■ Studentized range distribution

CDF:

\[
F(s) = C \int_0^\infty x^{V-1} e^{-(Vx^2)/2} \left\{ k \int_{-\infty}^{\infty} \theta(y) \left[ \Phi(y) - \Phi(y - sx) \right]^{k-1} dy \right\} dx \quad s > 0
\]

where

\[
C = \frac{V^{V/2}}{[\Gamma(V/2)][2^{V/2-1}]},
\]

\[
\theta(y) = \frac{1}{\sqrt{2\pi}} \exp\left\{ -y^2 / 2 \right\}
\]

and

\[
\Phi(y) = \int_{-\infty}^{y} \theta(t) dt
\]

Parameter(s):

- \( V \): degrees of freedom --- \{1,2,3, ...\}
- \( k \): number of samples --- \{2,3,4, ...\}

■ t distribution

PDF:

\[
f(x) = \frac{\Gamma\left(\frac{n+1}{2}\right)}{\Gamma(n/2)\sqrt{\pi} \ n} \left[ 1 + \frac{x^2}{n} \right]^{-\frac{n+1}{2}} \quad -\infty < x < +\infty
\]

Parameter(s):

- \( n \): degrees of freedom --- \{1,2,3, ...\}
- **Triangular distribution**

PDF:

\[
f(x) = \begin{cases} 
\frac{2(x-a)}{(b-a)(c-a)} & a \leq x < c \\
\frac{2(b-x)}{(b-a)(b-c)} & c \leq x \leq b 
\end{cases}
\]

Parameter(s):
- \(a\): minimum or low \((-\infty, \infty)\)
- \(b\): maximum or high \((-\infty, \infty)\)
- \(c\): mode \((-\infty, \infty)\)
- \(a < c < b\)

CDF:

\[
F(x) = \begin{cases} 
0 & x \leq a \\
\frac{(x-a)^2}{(b-a)(c-a)} & a < x < c \\
1 - \frac{(b-x)^2}{(b-a)(b-c)} & c \leq x < b \\
1 & x \geq b 
\end{cases}
\]

Inverse CDF:

\[
F^{-1}(p) = \begin{cases} 
\frac{a + (b-a)}{b-a} \sqrt{\frac{c-a}{p}} & p \leq \frac{c-a}{b-a} \\
\frac{a + (b-a)}{b-a} \left(1 - \sqrt{\frac{1 - \frac{c-a}{b-a}(1-p)}{1-p}}\right) & p > \frac{c-a}{b-a}
\end{cases}
\]

- **Uniform distribution**

PDF:

\[
f(x) = \frac{1}{b-a} \quad a \leq x \leq b
\]
Parameter(s):

\( a \): minimum or low \((-\infty, \infty)\)

\( b \): maximum or high \((-\infty, \infty)\)

\( a < b \)

CDF:

\[
F(x) = \begin{cases} 
0 & x \leq a \\
\frac{x-a}{b-a} & a < x < b \\
1 & x \geq b 
\end{cases}
\]

Inverse CDF:

\( F^{-1}(p) = a + p(b-a) \)

- **Weibull distribution**

PDF:

\[
f(x) = \frac{\alpha}{\beta^\alpha} \left(\frac{x}{\beta}\right)^{\alpha-1} \exp\left\{-\left(\frac{x}{\beta}\right)^\alpha\right\} \quad x > 0
\]

Parameter(s):

\( \beta \): scale \((0, \infty)\)

\( \alpha \): shape \((0, \infty)\)

CDF:

\[
F(x) = \begin{cases} 
0 & x \leq 0 \\
1 - \exp\left\{-\left(\frac{x}{\beta}\right)^\alpha\right\} & x > 0 
\end{cases}
\]

Inverse CDF:

\[
F^{-1}(p) = \beta \left(\ln\left(\frac{1}{1-p}\right)\right)^{\frac{1}{\alpha}}
\]

The algorithms for all functions are chosen to favor numerical accuracy over speed. Some functions may take more time to evaluate for some choices of parameters.
For more information about distributions, refer Evans et al. (2000), Johnson et al. (2005), Johnson et al. (1994), Johnson et al. (1995), and Lund and Lund (1983).

Using Distribution Functions

- Use the probability density functions (pdf) to obtain the height at $x$ of the ordinate under the density curve of the specified distribution. For example:

To compute the geometric density for a variable say $x$ with $p=0.75$:

```
LET geometric = GEDF(x, 0.75)
```

To compute Pareto density for a variable say $y$ with $\alpha=20.57$ and $\beta=1.75$:

```
LET pareto = PADF(y, 20.57, 1.75)
```

To compute Gumbel density for a variable say $z$ with $\text{loc}=-8.75$ and $\text{sc}=0.5$:

```
LET gumbel = GUDF(z, -8.75, 0.5)
```

- Use the cumulative distribution functions ($\text{cdf}$) to obtain probabilities associated with observed sample statistics. For many applications, to obtain significance levels, you must find the upper tail probabilities. For example:

```
LET p = 1 - FCF(4.14, 3, 7) !! from an $F(3,7)$ distribution
LET p = 1 - FCF(4.14, 3, 7) !! from a $t(8)$ distribution
LET p = 1 - ZCF(2.95, 4, 0.25) !! from a $N(4,0.25)$ distribution
```

- Use the inverse distributions ($\text{if}$) to determine the critical values and to construct confidence intervals. They are also handy for power calculations.

- You can use transformations for the random variate functions to produce random deviates from the desired distributions. To produce a random normal deviate with the required mean and standard deviation, use an expression like:

```
LET normal = mean + stddev*ZRN()
```

or

```
LET norm = ZRN(10, 2)
```

To produce a uniform deviate between $a$ and $b$, use an expression like:

```
LET unifrmab = a + (b-a)*URN()
```
or

LET unifrmab = URN(a,b)

To get an extreme value variate with location ALPHA and scale BETA, use:

LET extreme = alpha - beta*LOG(ERN())

or

LET extreme = GURN(alpha ,beta)

**Random Number Generators**

SYSTAT offers two algorithms for random number generation - the Wichmann-Hill (1982) algorithm from earlier versions and the more modern Mersenne-Twister (MT). Many features such as bootstrapping, random sampling from standard distributions and Monte Carlo computations depend crucially on the efficacy of the random number generator used. SYSTAT gives users the option between these two generators, which has to be exercised as a global option.

The Wichmann-Hill algorithm generates pseudorandom numbers by a triple modulo method. Each uniform variate is constructed from three multiplicative congruential generators with prime modulus. The initial seeds for each generator are 13579, 12345 and 131. You can modify the first seed of this generator by using RSEED:

```
RSEED seed
```

where seed is a positive integer.

Mersenne-Twister (MT) is a pseudorandom number generator developed by Makoto Matsumoto and Takuji Nishimura (1998). It is believed to have a far longer period and a far higher order of equidistribution than any of the other implemented generators. (It has been proved that the period is $2^{19937}-1$; and 623-dimensional equidistribution property is assured.) Here too you can mention your own seed by using RSEED:

```
RSEED seed
```

where *seed* is any positive integer from 1 to $2^{32}-1 = 4294967295$.

Mersenne-Twister is the default option. We recommend the MT option, especially if the number of uniform random numbers to be generated for your Monte Carlo exercise is large, say more than 10,000.
If you would like to reproduce results involving random number generation from earlier SYSTAT versions, use old command files or make sure that your random number generation option (under Edit=> Options=> General=>Random Number Generation=>) is Wichmann-Hill (and, of course, that your seed is the same as before.)

For the Wichmann-Hill generator, RSEED can be any positive integer from 1 to 30000. You can change the random number generator using RNDGEN:

```
RNDGEN WH
```

### Rank Dialog Box

The rank is the case number a value has if the data are sorted by that variable. For example, the smallest value has rank 1, and the largest value in a sample of 15 cases has rank 15. Ties are averaged. This example shows how ties are handled:

<table>
<thead>
<tr>
<th>Value</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3.5</td>
</tr>
<tr>
<td>3</td>
<td>3.5</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>11</td>
<td>6</td>
</tr>
</tbody>
</table>

To open the Rank dialog box, from the menus choose:

Data
Rank...
RANK replaces the values of one or more variables with their rank values. Only the selected variables are affected; all other variables are copied unchanged to the new file.

**Save file.** You can save the output in a data file. Otherwise, the transformed data are stored in a temporary file.

If you want to have both the ranks and the original values in one file, use LET to make a copy of the variable before ranking.

**Center Dialog Box**

You can center variables around their means, using Center from the Data menu (or CENTER command).

To open the Center dialog box, from the menus choose:

Data
   Center...

![Data: Center dialog box](image)

Center subtracts the mean of the specified variable from each value of that variable.

**Save file.** Saves the centered data in a data file. If you do not select this option, SYSTAT saves the centered data in a temporary file. If you want both centered values and values of the original variable, use LET to make a copy of the variable before centering.
**Standardize Dialog Box**

You can standardize one or more variables with Standardize from the Data menu (or the STANDARDIZE command).

To open the Standardize dialog box, from the menus choose:

Data
Standardize...

Standardize replaces the values of each specified variable with its sample standard score ($z$ score) or range-standardized scores. The following options are available:

**SD.** For each variable, SYSTAT subtracts the variable’s sample mean from each value and then divides the difference by the sample standard deviation. The standardized values have a mean of 0 and a standard deviation of 1. Standard scores are the default standardization measures.

**Range.** Subtracts the smallest data value of each variable from each value and divides by its range. The new scale starts at 0 and ends at 1.0.

**Save file.** Saves the standardized data in a data file. If you do not select this option, SYSTAT saves the standardized data in a temporary file. If you want both standardized scores and values of the original variable, use LET to make a copy of the variable before standardizing.
Trimming Data

Data sets often contain a few extreme values or outliers which may influence the computed estimates or statistics of interest. In such cases, you may opt for trimming a specified proportion of extreme observations, from one side or both sides.

To open the Trim dialog box, from the menus choose:

Data
Trim …

Trim removes (or excludes) the specified proportion (default is 0.10) of extreme observations from the data file for the selected variable(s). A two-sided 0.10 trim will trim the 0.10 proportion of the smallest and 0.10 proportion of the largest observations, not the 0.10 proportion of overall observations. You can employ one of the following two methods for trimming:

- **Separate.** Removes the specified proportion of extreme observations, separately for each of the selected variables. Here trimming of observations for one variable is independent of trimming of other variable(s). In this method, the trimmed observations are deleted and replaced by "." marks. Separate is the default trimming method.
- **Listwise.** Excludes the specified proportion of extreme observations separately for each of the selected variables at the first step, then completely excludes all those
cases which have observations for at least one of the selected variable(s) excluded. In this method, the trimmed observations are not deleted -- the cases that remain after trimming are shaded in yellow colour. For this method, you can invoke the Inverse Case Selection Toolbar (Data => Inverse Case Selection) which allows you to trim only the cases which are currently untrimmed.

Note that both the methods provide the same trimming if only one variable is trimmed.

The following options are available:

- **Trimming proportion.** Specifies the proportion of trimming. The value of the trimming proportion for two-sided trimming should lie in the range (0, 0.5). For lower or upper trimming, it should lie in the range (0,1). The default is 0.10.

- **Trimming region.** Selects whether two-sided (default) extreme observations have to be trimmed, or only the lower or upper extreme observations.

- **Save file.** Saves the trimmed data in a data file. If you do not select this option, SYSTAT saves the trimmed data in a temporary file. If you select the trimming method as "Separate", then this option saves all the cases as in the original data file with "." marks for the trimmed observations in the specified variable(s). If you select the "Listwise" method then this option saves only the shaded cases in the data editor.

- **Turn off.** Turns off case trimming so that all cases are used in subsequent analysis. You can also turn off case trimming by closing SYSTAT and opening a new data file.

---

**Reshaping Data**

SYSTAT enables you to reshape your data file in order to make it suitable for performing the desired analysis.

**Transpose**

Transpose changes rows to columns and viceversa, transposing cases and variables.

To open the Transpose dialog box, from the menus choose:

- Data
- Reshape
- Transpose…
Chapter 4

You can select variables to include in the transposed data file. If no variables are selected, all numeric variables are included.

You can transpose case labels into variable labels if the case labels are in a variable named $LABELS$ in the untransposed file. The values of $LABELS$ are used to label the variables in the transposed file. If there is no $LABELS$ variable, the names for the columns in the new transposed file are $COL(n)$ where $n$ is the case number in the original (untransposed) file.

In the transposed data file, SYSTAT creates a variable ($LABELS$) that preserves variable names from the untransposed file. For example:

<table>
<thead>
<tr>
<th>$X$</th>
<th>$Y$</th>
<th>$Z$</th>
<th>$COL(1)$</th>
<th>$COL(2)$</th>
<th>$COL(3)$</th>
<th>$COL(4)$</th>
<th>$LABELS$</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>15</td>
<td>4</td>
<td>10</td>
<td>6</td>
<td>4</td>
<td>8</td>
<td>$X$</td>
</tr>
<tr>
<td>6</td>
<td>11</td>
<td>9</td>
<td>15</td>
<td>11</td>
<td>13</td>
<td>9</td>
<td>$Y$</td>
</tr>
<tr>
<td>4</td>
<td>13</td>
<td>1</td>
<td>4</td>
<td>9</td>
<td>1</td>
<td>16</td>
<td>$Z$</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

String variables (other than $LABELS$) are not transposed and are dropped from the resulting data file.

**Save file.** Specifies an output file for the transposed data. If you do not select this option, SYSTAT saves the data in a temporary file.
Wrap/Unwrap

Wrap/Unwrap enables you to reshape your data in order to change a multivariate repeated measures layout to a split-plot data layout (or the WRAP command) or to change a split-plot data layout to a multivariate measures layout (or the UNWRAP command).

To open the Wrap/Unwrap dialog box, from the menus choose:
Data
  Reshape
    Wrap/Unwrap...

Wrap. Unpacks data from each case over multiple records according to the number of variables selected.

Unwrap. Packs each block of $n$ cases into a single record.

The left side of the following table illustrates three variables and two cases wrapped using all three variables. The right side of the table illustrates one variable with six cases unwrapped using blocks of size 3.
Stack

Stack arranges the observations from two or more numeric variables into a single column.

To open the Stack dialog box, from the menus choose:

Data
  Reshape
    Stack...

You can select the variables that are to be stacked. After stacking, two columns are created: GROUPS, which indicates the variable name, and VARIABLE, which shows the corresponding observations.

If the data file has other variables, they will be displayed after the VARIABLE column.
For example,

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>A</th>
<th>GROUPS VARIABLE</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>9</td>
<td>15</td>
<td>X</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>-2</td>
<td>12</td>
<td>X</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>7</td>
<td>76</td>
<td>X</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Y</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Y</td>
<td>-2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Y</td>
<td>7</td>
</tr>
</tbody>
</table>

becomes

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>A</th>
<th>GROUPS VARIABLE</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>-2</td>
<td>12</td>
<td>X</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>7</td>
<td>76</td>
<td>X</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>15</td>
<td>Y</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>-2</td>
<td>12</td>
<td>Y</td>
<td>-2</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>76</td>
<td>Y</td>
<td>7</td>
</tr>
</tbody>
</table>

**Weight Dialog Box**

Weight gives cases different weights for statistical analysis. Use Weight from the Data menu (or the WEIGHT command) to assign a weighting variable.

To open the Weight dialog box, from the menus choose:

Data
Case Weighting
By Weight...

You must select a numeric value that represents the degree of importance (or weight) the case should have when performing analyses. Typically, the weight is proportional to 1 divided by the variance, when such information is known. If single cases in your data file represent multiple observations, use frequency (from the menus, Data => Case Weighting => By Frequency). Once you apply a weight variable, it remains in effect until you select another weight variable or turn weighting off.
The Calculator

SYSTAT’s calculator is available with every statistical analysis or graphical display. Use Calculator from the Utilities menu (or the CALCULATE command) to calculate.

- Balance your checkbook.
- Get the current date and time.
- Take the antilog of an output result (for example, \(\exp(1.826)\)).
- Compute your own test statistic for comparing a smaller model (S) nested within a larger model (L)—use the residual sum of squares from the two models:

\[
F = \frac{(SS_S - SS_L) / (df_S - df_L)}{SS_S / df_L}
\]

- Find the probability associated with the resulting \(F\) statistic.
- Try out a transformation statement to check if your formula is correct—for example, to compute percentage change:

\[
\frac{after - before}{before} \times 100
\]

The calculator does not know the values of any variables in your current file—it uses only the numbers you enter into it. All the functions (except File related functions like DATA(), CASE() and COMPLETE()) and operators in SYSTAT are available to the calculator. CALC command allows library functions returning numeric values like LOG(), SQR(), LEN, etc. in its expression. The function arguments are restricted to constants or temporary variables. You cannot pass data variables as arguments for these functions. Functions like DATA(), CASE() and COMPLETE() cannot be used.
For example, to calculate today’s date and time, type

```
CALC NOW$ ('mmm dd, hhhh:mm')
```

SYSTAT gives you today’s date and time in the requested format.

To exponentiate the value 1.826 (that is, to compute $e^{1.826}$), type

```
CALC EXP(1.826)
```

yielding the result:

```
6.209
```

You can specify transformation commands in any of SYSTAT’s statistical or graphical procedures. Just type your transformation statements before the `HOT` command for that procedure. For example, to transform `GNP` per capita to log base 10 units for an `ANOVA` procedure

```
ANOVA
USE OURWORLD
LET gdp_cap = L10(gdp_cap)
DEPEND gdp_cap
CATEGORY gdp$
ESTIMATE
```

The results of transformations are not automatically saved in a file. To save results using commands, type

```
DSAVE filename
```

after `ESTIMATE`. 
To replace variable values with ranks, type
   
   RANK varlist

To center variables, type
   
   CENTER varlist

To standardize variables, type
   
   STANDARDIZE varlist / SD or RANGE

To stack variables, type
   
   STACK varlist

To trim data, type
   
   TRIM varlist
   / TRPROP =p (0<p<1) for lower or upper and (0<p<0.5) for two-sided trimming.
   TREGION = TWOSIDED or UPPER or LOWER
   METHOD = SEPARATE or LISTWISE

To turn off case trimming, type
   
   TRIM / OFF

For RANK, CENTER, STANDARDIZE, STACK, and TRIM commands, if no variable is specified, all numeric variables in the data file are used. If you want to have both the modified and the original values in one file, use LET to make a copy of the variable before issuing these commands:
   
   LET age2 = age

To weight cases, type
   
   WEIGHT var

To transpose cases and variables, type
   
   TRANSPOSE varlist

Varlist is optional and may include only LABELS and numeric variables. Notice that transposing a symmetric matrix (for example, correlations) is unnecessary, since the transpose of a symmetric matrix is the original matrix.
To wrap variables, type

WRAP varlist

To unwrap variables, into a block of $n$ cases on a single record, type

UNWRAP $n$

To specify random seed, type

RSEED $n$

To select random number generator, type

RNDGEN MT or WH (Default is MT)

**Examples**

**Example 1**

**Lagging Variables**

The LAG function shifts values down $n$ rows, replacing the first $n$ values with a missing value:

LAG ($var$, $n$)

If $n$ is omitted, the default is 1. Examples include:

LET $y$=LAG($x$)
LET $z$=LOG(LAG($x$))

The first case of a lagged variable is set to missing.

**Cumulative Sum**

The cumulative sum of a variable can be computed using the LAG function.
The input is:

```
NEW
REPEAT 10
LET A = CASE()
IF (CASE()=1) THEN LET CUSUM=A
IF (CASE() > 1) THEN LET CUSUM=LAG(CUSUM)+A
LIST CUSUM
```

The output is:

<table>
<thead>
<tr>
<th>Case</th>
<th>CUSUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.000</td>
</tr>
<tr>
<td>2</td>
<td>3.000</td>
</tr>
<tr>
<td>3</td>
<td>6.000</td>
</tr>
<tr>
<td>4</td>
<td>10.000</td>
</tr>
<tr>
<td>5</td>
<td>15.000</td>
</tr>
<tr>
<td>6</td>
<td>21.000</td>
</tr>
<tr>
<td>7</td>
<td>28.000</td>
</tr>
<tr>
<td>8</td>
<td>36.000</td>
</tr>
<tr>
<td>9</td>
<td>45.000</td>
</tr>
<tr>
<td>10</td>
<td>55.000</td>
</tr>
</tbody>
</table>

**Example 2**

**Finding the Number of Missing Values**

Use the MIS function to report the number of missing values. For example, if you have survey results, use MIS to find the number of questions each respondent did not answer. Here, we use the TESTSCOR data file.

<table>
<thead>
<tr>
<th>SUBJECTS</th>
<th>Q1</th>
<th>Q2</th>
<th>Q3</th>
<th>Q4</th>
<th>Q5</th>
<th>Q6</th>
<th>Q7</th>
<th>Q8</th>
<th>Q9</th>
<th>Q10</th>
<th>NUM_MISS</th>
</tr>
</thead>
<tbody>
<tr>
<td>N. Smith</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td>.</td>
<td>3</td>
<td>.</td>
<td>5</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>G. Henry</td>
<td>.</td>
<td>2</td>
<td>.</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>.</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>C. Bauer</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>8</td>
<td>9</td>
<td>1</td>
<td>5</td>
<td>.</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>R. McMahon</td>
<td>2</td>
<td>5</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>.</td>
<td>.</td>
<td>5</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>S. Collins</td>
<td>1</td>
<td>2</td>
<td>.</td>
<td>5</td>
<td>2</td>
<td>.</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>L. Ryan</td>
<td>5</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>.</td>
<td>.</td>
<td>3</td>
<td>.</td>
<td>5</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>E. Clark</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>.</td>
<td>4</td>
<td>5</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>T. Price</td>
<td>5</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>.</td>
<td>2</td>
<td>.</td>
<td>.</td>
<td>1</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

We create a new variable, `NUM_MISS`, containing the number of questions each respondent did not answer:

```
LET num_miss = MIS(q1 .. q10)
```
Instead of typing sequential variable names (Q1, Q2, Q3, Q4, etc.), use a double period (..) to include all variables in between (Q1 .. Q10). If you save the data, NUM_MISS is the last variable.

**Example 3**

**Summary Statistics**

Using four scores (S1 through S4) as data for each subject, we use the NUM, SUM, MIN, and STD functions to compute:

- **NUM_SCRS** (how many scores are present)
- **TOTAL** (the sum of their scores)
- **SMALLEST** (their minimum score)
- **SD** (the standard deviation of the scores)

The input is:

\[
\begin{align*}
\text{LET num_scrs} &= \text{NUM} (s_1, s_2, s_3, s_4) \\
\text{LET total} &= \text{SUM} (s_1, s_2, s_3, s_4) \\
\text{LET smallest} &= \text{MIN} (s_1, s_2, s_3, s_4) \\
\text{LET sd} &= \text{STD} (s_1, s_2, s_3, s_4)
\end{align*}
\]

The following are the original data with results for three subjects:

<table>
<thead>
<tr>
<th>Names</th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>NUM_SCR</th>
<th>TOTAL</th>
<th>SMALLEST</th>
<th>SD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td>7</td>
<td>6</td>
<td>.</td>
<td>3</td>
<td>3</td>
<td>16</td>
<td>3</td>
<td>2.082</td>
</tr>
<tr>
<td>Jones</td>
<td>8</td>
<td>4</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>23</td>
<td>4</td>
<td>1.708</td>
</tr>
<tr>
<td>Wilson</td>
<td>2</td>
<td>5</td>
<td>3</td>
<td>7</td>
<td>4</td>
<td>17</td>
<td>2</td>
<td>2.217</td>
</tr>
</tbody>
</table>

**Example 4**

**Slope of the Line of Best Fit**

For each case, the SLE and SLU functions compute the slope of the line of best fit across two or more responses. Consider these four responses across four time points:

<table>
<thead>
<tr>
<th>T1</th>
<th>R1</th>
<th>T2</th>
<th>R2</th>
<th>T3</th>
<th>R3</th>
<th>T4</th>
<th>R4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case 1</td>
<td>2</td>
<td>10</td>
<td>4</td>
<td>20</td>
<td>8</td>
<td>40</td>
<td>16</td>
</tr>
<tr>
<td>Case 2</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>20</td>
<td>8</td>
<td>15</td>
<td>16</td>
</tr>
</tbody>
</table>
The SLE function produces the slope if you assume that the times are equally spaced. To incorporate the actual spacing (2, 4, 8, 16), use the SLU function. The resulting slopes can then be used as data in statistical procedures or graphical displays—for example, to test if the average slope for a control group differs from that for the treatment group. The following, for example, are the responses for case 1 plotted at equally spaced times and at the actual times:

To request the slope for equally spaced time points, \( a \), or the slope using the actual times, \( b \), specify:

\[
\begin{align*}
\text{LET } & \text{slope}_a = \text{SLE}(r1, r2, r3, r4) \\
\text{LET } & \text{slope}_b = \text{SLU}(t1, r1, t2, r2, t3, r3, t4, r4)
\end{align*}
\]

You can get the same results for \( \text{SLOPE}_B \) by inserting the times directly:

\[
\begin{align*}
\text{LET } & \text{slope}_b = \text{SLU}(2, r1, 4, r2, 8, r3, 16, r4)
\end{align*}
\]

The results are:

\[
\begin{array}{ccc}
\text{\textbf{SLOPE}_A} & \text{\textbf{SLOPE}_B} \\
\hline
\text{Case 1} & 23.000 & 5.000 \\
\text{Case 2} & -0.200 & -0.440
\end{array}
\]
Example 5  
Computing the Area under a Curve

For each case, the ARE and ARU functions compute the area under the polygon formed by connecting values of variables you specify. Consider the data for Case 2:

<table>
<thead>
<tr>
<th></th>
<th>( T_1 )</th>
<th>( R_1 )</th>
<th>( T_2 )</th>
<th>( R_2 )</th>
<th>( T_3 )</th>
<th>( R_3 )</th>
<th>( T_4 )</th>
<th>( R_4 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case 1</td>
<td>2</td>
<td>10</td>
<td>4</td>
<td>20</td>
<td>8</td>
<td>40</td>
<td>16</td>
<td>80</td>
</tr>
<tr>
<td>Case 2</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>20</td>
<td>8</td>
<td>15</td>
<td>16</td>
<td>2</td>
</tr>
</tbody>
</table>

Let’s use the ARE and ARU functions:

\[
\text{LET } \text{area}_1 = \text{ARE} (r_1, r_2, r_3, r_4) \\
\text{LET } \text{area}_2 = \text{ARU} (t_1, r_1, t_2, r_2, t_3, r_3, t_4, r_4) \\
\text{or} \\
\text{LET } \text{area}_2 = \text{ARU} (2, r_1, 4, r_2, 8, r_3, 16, r_4)
\]

The results are:

<table>
<thead>
<tr>
<th></th>
<th>\text{AREA}_1</th>
<th>\text{AREA}_2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case 1</td>
<td>105.000</td>
<td>630.000</td>
</tr>
<tr>
<td>Case 2</td>
<td>36.500</td>
<td>159.000</td>
</tr>
</tbody>
</table>
Chapter 4

Example 6
Matching Specific Values

The INC and COD functions can be used with character or numeric variables to find specified values. Within each case, the INC and COD functions search the values of selected variables for a value that matches the value you specify. When a match is found:

- INC returns a 1 (true).
- COD returns the index or order of the variable with the match (1 if the value is found in the first variable listed; 2, if it is found in the second, etc.).

If no match is found, both INC and COD return 0.

As an example, suppose respondents in a survey were asked about their favorite bath soaps:

- Is Ivory soap among the first, second, or third choices?
- What is the preference order of Ivory?

Character data. The data and results for three cases are:

<table>
<thead>
<tr>
<th>NAMES</th>
<th>CHOICE_1$</th>
<th>CHOICE_2$</th>
<th>CHOICE_3$</th>
<th>CHOICE_4$</th>
<th>PREFRNCES</th>
<th>ORDER</th>
<th>PREF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td>Ivory</td>
<td>Dial</td>
<td>Camay</td>
<td>Safeguard</td>
<td>yes</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Jones</td>
<td>Palmolive</td>
<td>Irish Spring</td>
<td>Camay</td>
<td>Ivory</td>
<td>no</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Wilson</td>
<td>Zest</td>
<td>Ivory</td>
<td>Camay</td>
<td>Palmolive</td>
<td>yes</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

To create the new variables PREFRNCES and ORDER, we use the INC and COD functions:

```plaintext
LET PREFRNCES$ = 'no'
IF INC('Ivory', CHOICE_1$, CHOICE_2$, CHOICE_3$),
   THEN LET PREFRNCES$ = 'yes'
LET ORDER = COD('Ivory', CHOICE_1$, CHOICE_2$, CHOICE_3$)
```

A more direct use of the INC function is:

```plaintext
LET PREF = INC('Ivory', CHOICE_1$, CHOICE_2$, CHOICE_3$)
```

A PREF value of 1 indicates that Ivory is among the first three choices, and a value of 0 indicates that Ivory is not one of the first three choices.
**Numeric data.** For the same scenario, the data might be numeric brand codes:

<table>
<thead>
<tr>
<th></th>
<th>Camay</th>
<th>Dove</th>
<th>Ivory</th>
<th>Safeguard</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>5</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

The data and results are:

<table>
<thead>
<tr>
<th>NAME</th>
<th>CHOICE_1</th>
<th>CHOICE_2</th>
<th>CHOICE_3</th>
<th>CHOICE_4</th>
<th>PREFRNCE$</th>
<th>ORDER</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>7</td>
<td>yes</td>
<td>1</td>
</tr>
<tr>
<td>Jones</td>
<td>6</td>
<td>4</td>
<td>1</td>
<td>5</td>
<td>no</td>
<td>0</td>
</tr>
<tr>
<td>Wilson</td>
<td>8</td>
<td>5</td>
<td>1</td>
<td>6</td>
<td>yes</td>
<td>2</td>
</tr>
</tbody>
</table>

The commands to obtain `PREFRNCE$` and `ORDER` are:

```plaintext
LET PREFRNCE$ = 'no'
IF INC(5,CHOICE_1,CHOICE_2,CHOICE_3),
   THEN LET PREFRNCE$ = 'yes'
LET ORDER = COD(5,CHOICE_1,CHOICE_2,CHOICE_3)
```

Remember to use `DSAVE filename` if you want to save the new variables.

**Example 7**

*Changing the Case of Character Values*

The following examples use the `NATIONS` data file with 16 cases:

**COUNTRY$**

<table>
<thead>
<tr>
<th>Austria</th>
<th>New Guinea</th>
</tr>
</thead>
<tbody>
<tr>
<td>Burkina Faso</td>
<td>New Zealand</td>
</tr>
<tr>
<td>Costa Rica</td>
<td>Norway</td>
</tr>
<tr>
<td>France</td>
<td>Portugal</td>
</tr>
<tr>
<td>Ireland</td>
<td>Sierra Leone</td>
</tr>
<tr>
<td>Italy</td>
<td>UK</td>
</tr>
<tr>
<td>Ivory Coast</td>
<td>USA</td>
</tr>
<tr>
<td>Netherlands</td>
<td>Yugoslavia</td>
</tr>
</tbody>
</table>

Let’s use the `UPR$`, `LOW$`, and `CAP$` functions to display the country names in four ways:

- As entered in the file.
- In capital letters.
- In lowercase letters.
- With initial capital letters.

To list these variables, we use the picture format, where the < specifies that the values of the character variable are left-justified.

The input is:

```plaintext
USE NATIONS
LET COUNTRY2$ = UPR$(COUNTRY$)
LET COUNTRY3$ = LOW$(COUNTRY$)
LET COUNTRY4$ = CAP$(COUNTRY3$)
LIST COUNTRY$ .. COUNTRY4$ / FORMAT =,
'<<<<<<<<<<<     <<<<<<<<<<<<     <<<<<<<<<<<<
              <<<<<<<<<<<<
Case  COUNTRY$  COUNTRY2$  COUNTRY3$  COUNTRY4$
-----+--------------------------------------------------------
 1   Austria        AUSTRIA        austria        Austria
 2   Burkina Faso   BURKINA FASO   burkina faso   Burkina faso
 3   Costa Rica     COSTA RICA     costa rica     Costa rica
 4   France         FRANCE         france         France
 5   Ireland        IRELAND        ireland        Ireland
 6   Italy          ITALY          italy          Italy
 7   Ivory Coast    IVORY COAST    ivory coast    Ivory coast
 8   Netherlands    NETHERLANDS    netherlands    Netherlands
 9   New Guinea     NEW GUINEA     new guinea     New guinea
10   New Zealand    NEW ZEALAND    new zealand    New zealand
11   Norway         NORWAY         norway         Norway
12   Portugal       PORTUGAL       portugal       Portugal
13   Sierra Leone   SIERRA LEONE   sierra leone   Sierra leone
14   UK             UK             uk             Uk
15   USA            USA            usa            Usa
16   Yugoslavia     YUGOSLAVIA     yugoslavia     Yugoslavia
```

The output is:

```plaintext
<table>
<thead>
<tr>
<th>Case</th>
<th>COUNTRY$</th>
<th>COUNTRY2$</th>
<th>COUNTRY3$</th>
<th>COUNTRY4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Austria</td>
<td>AUSTRIA</td>
<td>austria</td>
<td>Austria</td>
</tr>
<tr>
<td>2</td>
<td>Burkina Faso</td>
<td>BURKINA FASO</td>
<td>burkina faso</td>
<td>Burkina faso</td>
</tr>
<tr>
<td>3</td>
<td>Costa Rica</td>
<td>COSTA RICA</td>
<td>costa rica</td>
<td>Costa rica</td>
</tr>
<tr>
<td>4</td>
<td>France</td>
<td>FRANCE</td>
<td>france</td>
<td>France</td>
</tr>
<tr>
<td>5</td>
<td>Ireland</td>
<td>IRELAND</td>
<td>ireland</td>
<td>Ireland</td>
</tr>
<tr>
<td>6</td>
<td>Italy</td>
<td>ITALY</td>
<td>italy</td>
<td>Italy</td>
</tr>
<tr>
<td>7</td>
<td>Ivory Coast</td>
<td>IVORY COAST</td>
<td>ivory coast</td>
<td>Ivory coast</td>
</tr>
<tr>
<td>8</td>
<td>Netherlands</td>
<td>NETHERLANDS</td>
<td>netherlands</td>
<td>Netherlands</td>
</tr>
<tr>
<td>9</td>
<td>New Guinea</td>
<td>NEW GUINEA</td>
<td>new guinea</td>
<td>New guinea</td>
</tr>
<tr>
<td>10</td>
<td>New Zealand</td>
<td>NEW ZEALAND</td>
<td>new zealand</td>
<td>New zealand</td>
</tr>
<tr>
<td>11</td>
<td>Norway</td>
<td>NORWAY</td>
<td>norway</td>
<td>Norway</td>
</tr>
<tr>
<td>12</td>
<td>Portugal</td>
<td>PORTUGAL</td>
<td>portugal</td>
<td>Portugal</td>
</tr>
<tr>
<td>13</td>
<td>Sierra Leone</td>
<td>SIERRA LEONE</td>
<td>sierra leone</td>
<td>Sierra leone</td>
</tr>
<tr>
<td>14</td>
<td>UK</td>
<td>UK</td>
<td>uk</td>
<td>Uk</td>
</tr>
<tr>
<td>15</td>
<td>USA</td>
<td>USA</td>
<td>usa</td>
<td>Usa</td>
</tr>
<tr>
<td>16</td>
<td>Yugoslavia</td>
<td>YUGOSLAVIA</td>
<td>yugoslavia</td>
<td>Yugoslavia</td>
</tr>
</tbody>
</table>
```

Example 8
Justifying Character Values

The following commands

```plaintext
USE NATIONS
LET country2$ = CNT$(country$,12)
LET country3$ = RGT$(country$,12)
LET country4$ = SQZ$(country3$)
LIST country$ .. country4$ / FORMAT =,
'$$$$$$$$$$$$$$ $$$$$$$$$$$$$ $$$$$$$$$$$$$ $$$$$$$$$$$$$$'
```
Data Transformations

produce three new variables with values centered, right justified, and without spaces. To list these variables, we use the picture format, where $\$" specifies that the values of the character variables are printed as they are stored. Note that we also used the SQZ$ function to remove the blanks from some of the names.

The output is:

<table>
<thead>
<tr>
<th>Case</th>
<th>COUNTRY$</th>
<th>COUNTRY2$</th>
<th>COUNTRY3$</th>
<th>COUNTRY4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Austria</td>
<td>Austria</td>
<td>Austria</td>
<td>Austria</td>
</tr>
<tr>
<td>2</td>
<td>Burkina Faso</td>
<td>Burkina Faso</td>
<td>Burkina Faso</td>
<td>BurkinaFaso</td>
</tr>
<tr>
<td>3</td>
<td>Costa Rica</td>
<td>Costa Rica</td>
<td>Costa Rica</td>
<td>CostaRica</td>
</tr>
<tr>
<td>4</td>
<td>France</td>
<td>France</td>
<td>France</td>
<td>France</td>
</tr>
<tr>
<td>5</td>
<td>Ireland</td>
<td>Ireland</td>
<td>Ireland</td>
<td>Ireland</td>
</tr>
<tr>
<td>6</td>
<td>Italy</td>
<td>Italy</td>
<td>Italy</td>
<td>Italy</td>
</tr>
<tr>
<td>7</td>
<td>Ivory Coast</td>
<td>Ivory Coast</td>
<td>Ivory Coast</td>
<td>IvoryCoast</td>
</tr>
<tr>
<td>8</td>
<td>Netherlands</td>
<td>Netherlands</td>
<td>Netherlands</td>
<td>Netherlands</td>
</tr>
<tr>
<td>9</td>
<td>New Guinea</td>
<td>New Guinea</td>
<td>New Guinea</td>
<td>NewGuinea</td>
</tr>
<tr>
<td>10</td>
<td>New Zealand</td>
<td>New Zealand</td>
<td>New Zealand</td>
<td>NewZealand</td>
</tr>
<tr>
<td>11</td>
<td>Norway</td>
<td>Norway</td>
<td>Norway</td>
<td>Norway</td>
</tr>
<tr>
<td>12</td>
<td>Portugal</td>
<td>Portugal</td>
<td>Portugal</td>
<td>Portugal</td>
</tr>
<tr>
<td>13</td>
<td>Sierra Leone</td>
<td>Sierra Leone</td>
<td>Sierra Leone</td>
<td>SierraLeone</td>
</tr>
<tr>
<td>14</td>
<td>UK</td>
<td>UK</td>
<td>UK</td>
<td>UK</td>
</tr>
<tr>
<td>15</td>
<td>USA</td>
<td>USA</td>
<td>USA</td>
<td>USA</td>
</tr>
<tr>
<td>16</td>
<td>Yugoslavia</td>
<td>Yugoslavia</td>
<td>Yugoslavia</td>
<td>Yugoslavia</td>
</tr>
</tbody>
</table>

Example 9
Deleting Blanks or Other Characters

If you had numbers entered as character values (235,235), you could use SQZ$ to remove the commas and then convert the character values to numeric values:

$$\begin{align*}
\text{LET} & \quad \text{chrvar2$} = \text{SQZ$(chrvar$, ",")}
\text{LET} & \quad \text{numvar} = \text{VAL(chrvar2$)}
\end{align*}$$

These commands convert the character value 235,235, originally stored in CHRVAR$, to the number 235235 and store it in NUMVAR.

Examples of SQZ$ include:

- SQZ$(city$) By default blanks are removed. New York becomes NewYork.
- SQZ$(name$,"n") Remove all n’s.
- SQZ$(name$,",") Remove all commas.
- SQZ$(airport$,"") Remove all apostrophes, as in O’Hare.
Example 10
Extracting and Inserting Characters

The following commands illustrate the mechanics of the MID$, PUT$, RPD$, and LPD$ functions. These functions operate on the values of the variable CHARDATA$ shown on the left in the table below:

```plaintext
USE CHARDATA
LET Last4$ = MID$(CHARDATA$, 9, 4)
LET Insert_X$ = SUB$(CHARDATA$, 'efgh', 'XXXX')
LET Insert_Y$ = PUT$(CHARDATA$, 'YYY', 5, 4)
LET RTPad_X$ = RPD$(CHARDATA$, 'X')
LET LFTPad_Y$ = LPD$((RGT$(CHARDATA$)), 'Y')
LET Midd$ = MID$(CHARDATA$, 3, 4)
LET Subs$ = SUB$(CHARDATA$, "abc", "DATA")
LET L = LEN(CHARDATA$)
```

LIST CHARDATA$  Last4$  Insert_X$  Insert_Y$  RTPad_X$

<table>
<thead>
<tr>
<th>Case</th>
<th>CHARDATA$</th>
<th>Last4$</th>
<th>Insert_X$</th>
<th>Insert_Y$</th>
<th>RTPad_X$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>abcdefghijkl</td>
<td>ijkl</td>
<td>abcdXXXXijkl</td>
<td>abcdYYYYijkl</td>
<td>abcdghijkl</td>
</tr>
<tr>
<td>2</td>
<td>abc fghijkl</td>
<td>jkkl</td>
<td>abc fkghijkl</td>
<td>abc YYYijkl</td>
<td>abc ghijkl</td>
</tr>
<tr>
<td>3</td>
<td>cdefghijkl</td>
<td>kkl</td>
<td>cdXXXXijkl</td>
<td>cdefYYYYkl</td>
<td>cdefghijklklXX</td>
</tr>
<tr>
<td>4</td>
<td>abcdefgh</td>
<td>klkl</td>
<td>abcdXXXX</td>
<td>abcdYYYY</td>
<td>abcdefghXXX</td>
</tr>
<tr>
<td>5</td>
<td>123,456.78</td>
<td>78</td>
<td>123,456.78</td>
<td>123,YYYY78</td>
<td>123,456.78XX</td>
</tr>
</tbody>
</table>

Case | LFTPad_Y$ | Midd$ | Subs$
----+-----------+-------+------
1   | abcdefghijkl | cdef  | DATAdefghijkl |
2   | abc fghijkl  | c f   | DATA fghijkl |
3   | cdefghijkl   | efgh  | cdefghijkl   |
4   | abcdefgh     | cdef  | DATAdefgh |
5   | 123,456.78   | 3,45  | 123,456.78   |

Remembering that values of character variables can contain up to 256 characters, we make the following observations for the first case:

- **LAST4$** contains the last four characters (ijkl) of the string abcdefghijkl. SYSTAT starts at the ninth character (i) and extracts four characters.
- For **INSERT_X$**, SYSTAT replaces the string efg with XXXX.
- To create **INSERT_Y$**, SYSTAT starts at the fifth character (e) of abcdefghijkl and replaces four characters with YYY.
- **MIDD$** contains the four characters of the string starting from the third character.
L returns the length of the string.
SYSTAT adds X’s to the right side of the value (see Data editor).
Since the string variables by default are left-justified leaving no empty space to the left, we make use of the RGTS functions to first right justify them. Notice that the Y’s are appended to the left side of the last 72 characters (see Data editor).

Example 11
Converting Numbers from European to American Notation

You can use SYSTAT’s character functions to convert numbers from European notation to American notation. When writing numbers, Europeans use commas where Americans use periods (decimal points) and vice versa. In SYSTAT, numbers in European notation must be stored as character values. However, you can use the SUB$, SQZ$, and VAL functions to convert European numbers to American numbers stored as numeric values.

Suppose that the variable EUR_NUMS contains numbers in European notation.

**EUR_NUMS**
365.452,137
256,4
8957
2.734.102,56

To convert these numbers to American notation, the input is:

```plaintext
LET NO_PRD$ = SQZ$(EUR_NUM$, '.')
LET AM_NUM$ = SUB$(NO_PRD$, ',', '.')
LET AM_NUM = VAL(AM_NUM$)
LIST EUR_NUM$, NO_PRD$, AM_NUM$, AM_NUM / FORMAT=',
'<<<<<<<<<<<<   <<<<<<<<<<<<   <<<<<<<<<<<  #######.###'
```

The output is:

<table>
<thead>
<tr>
<th>Case</th>
<th>EUR_NUMS</th>
<th>NO_PRD$</th>
<th>AM_NUM$</th>
<th>AM_NUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>365.452,137</td>
<td>365452,137</td>
<td>365452.137</td>
<td>365452.137</td>
</tr>
<tr>
<td>2</td>
<td>256,4</td>
<td>256,4</td>
<td>256.4</td>
<td>256.400</td>
</tr>
<tr>
<td>3</td>
<td>8957</td>
<td>8957</td>
<td>8957</td>
<td>8957.000</td>
</tr>
<tr>
<td>4</td>
<td>2.734.102,56</td>
<td>2734102,56</td>
<td>2734102.56</td>
<td>2.734E+006</td>
</tr>
</tbody>
</table>
**Example 12**

**Extracting First Names**

Suppose your file contains a variable `FULLNAME$` with both the first and last names of each subject:

`FULLNAME$
Scout Finch
Jane Eyre
Tom Sawyer
Billy Budd

You can extract the first name of each subject and store the result in the variable `FIRST$`. There are two ways to create `FIRST$`.

**Using MID$.** You can use IND and MID$ together to create `FIRST$`:

```
LET INDEX  = IND(FULLNAME$,' ')
LET FIRST$ = MID$(FULLNAME$,1,INDEX–1)
```

The first LET statement finds the position of the blank between the first and last name and stores the position in the variable `INDEX`. The second LET statement starts with the first character of `FULLNAME$`, extracts `INDEX–1` characters, and stores the resulting string in the variable `FIRST$`. So, for the first case, `Scout Finch`, `INDEX` gets the value 6, since the blank is the sixth character. Then the MID$ function extracts the first five (`INDEX–1`) characters (Scout) of `FULLNAME$`, and stores the string in `FIRST$`.

Note that you can create `FIRST$` with only one LET statement by embedding the IND function in the MID$ function as follows:

```
LET FIRST$ = MID$(FULLNAME$,1,IND(FULLNAME$,' ')–1)
```

**Using PUT$.** Alternatively, you can create `FIRST$` by using the PUT$ function to replace the letters in each last name with blanks:

```
LET INDEX  = IND(FULLNAME$,' ')
LET FIRST$ = PUT$(FULLNAME$,' ',INDEX,13–INDEX)
```

So, for the first case, `INDEX` gets the value 6 as before. Then the PUT$ function starts with the sixth character and replaces the next seven (`13–INDEX`) characters with blanks. The result, `Scout`, is stored in the variable `FIRST$`. 
As with the MID$ function, you can save a step by embedding the IND function in the PUT$ function:

\[
\text{LET FIRST$ = PUT$(FULLNAME$,' ', IND(FULLNAME$,' '), 13-IND(FULLNAME$,' '))}
\]

**Example 13**

**Concatenating Character Strings**

Suppose a file called `MYDATA` contains the following data:

<table>
<thead>
<tr>
<th>AGES</th>
<th>SEX$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adult</td>
<td>Female</td>
</tr>
<tr>
<td>Child</td>
<td>Male</td>
</tr>
<tr>
<td>Adult</td>
<td>Male</td>
</tr>
<tr>
<td>Teen</td>
<td>Female</td>
</tr>
<tr>
<td>Child</td>
<td>Female</td>
</tr>
<tr>
<td>Child</td>
<td>Male</td>
</tr>
<tr>
<td>Adult</td>
<td>Female</td>
</tr>
<tr>
<td>Child</td>
<td>Female</td>
</tr>
<tr>
<td>Teen</td>
<td>Male</td>
</tr>
</tbody>
</table>

You can use CAT$ to combine these values into a new variable named `SEX_AGE$`. To make the new values easier to read, separate the `SEX` and `AGES` values with a blank. You can do this by first appending an underscore to the beginning of the `AGES` value, concatenating the `SEX$` and `AGES` variables, and using the SUB$ function to replace the underscores with blanks. We finish by sorting the values by `SEX_AGE$` and listing the results:

```
USE MYDATA
LET AGE$ = CAT$('_',AGE$)
LET SEX_AGE$ = CAT$(SEX$,AGE$)
LET SEX_AGE$ = SUB$(SEX_AGE$,'_',' ')
SORT SEX_AGE$
LIST SEX$ AGE$ SEX_AGE$ / FORMAT='
<<<<<<  <<<<<<  <<<<<<<<<<<<'
```
Chapter 4

The output is:

<table>
<thead>
<tr>
<th>Case</th>
<th>SEX$</th>
<th>AGE$</th>
<th>SEX_AGE$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Female _Adult</td>
<td>Female Adult</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Female _Adult</td>
<td>Female Adult</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Female _Child</td>
<td>Female Child</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Female _Child</td>
<td>Female Child</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Female _Teen</td>
<td>Female Teen</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Male _Adult</td>
<td>Male Adult</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Male _Child</td>
<td>Male Child</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Male _Child</td>
<td>Male Child</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Male _Teen</td>
<td>Male Teen</td>
<td></td>
</tr>
</tbody>
</table>

The results are as expected. To save the data, type DSAVE filename. Now we can use SEX_AGE$ as a grouping variable and request plots, statistics, or other statistical analyses separately for the six groups.

Example 14
Using Soundex to Code Names

Let's say you want to check for duplicate records in a large file with format as in the file named SOUNDEx listed below, but you wonder about alternative spellings. First create a variable containing the Soundex code for the LASTNAM$ variable:

```
LET CODE$ = SND$(lastnam$)
```

Then sort the file by this new variable:

```
LASTNAMS | INCOME  | AGE | SEX$ | CODE$
---------|--------|-----|------|-------
CASE 1    Barrett       376.300 | 22.000 | F   | B630  
CASE 2    Barrott       872.100 | 34.000 | F   | B630  
CASE 3    Howell        987.200 | 36.000 | F   | H400  
CASE 4    MacCarthy     765.100 | 45.000 | M   | M263  
CASE 5    MacLoud       987.300 | 40.000 | F   | M243  
CASE 6    McCarthy      765.000 | 24.000 | M   | M263  
CASE 7    McCarty       367.100 | 38.000 | F   | M263  
CASE 8    McLoud        987.300 | 40.000 | F   | M243  
CASE 9    McLeod        542.300 | 32.000 | M   | M243  
CASE 10   Stephens      683.200 | 30.000 | F   | S315  
CASE 11   Stevens       743.200 | 55.000 | M   | S315  
CASE 12   Wilkenson     999.300 | 48.000 | F   | W425  
CASE 13   Wilkinson     235.100 | 48.000 | M   | W425  
```
By viewing the file, you could then easily pick out duplicate records. If you are looking for a specific duplicate record, instead of creating the `CODE$` variable, you can select all records with alternative spellings of the same name:

```sql
SELECT SND$('stephens') = SND$(LASTNAM$)
```

Since the Soundex code for both names (Stephens and Stevens) is S315, both cases are selected. Now, we just list the selected cases to compare `AGE`, `SEX`, and `INCOME` to be sure that we have unique records, or a duplicate spelling of the same record:

Data for the following results were selected according to `SELECT SND$('STEPHENS') = SND$(LASTNAM$)`

<table>
<thead>
<tr>
<th>Case</th>
<th>LASTNAM$</th>
<th>INCOME</th>
<th>AGE</th>
<th>SEX$</th>
<th>CODE$</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>STEPHENS</td>
<td>683.200</td>
<td>30.000</td>
<td>F</td>
<td>S315</td>
</tr>
<tr>
<td>11</td>
<td>STEVENS</td>
<td>743.200</td>
<td>55.000</td>
<td>M</td>
<td>S315</td>
</tr>
</tbody>
</table>

You will discover that the records for Stephens and Stevens are unique.

Soundex uses the first letter of the word to begin the code, so words such as Carson and Karson have different codes—C625 and K625. Also, Soundex is not useful for languages other than English.

**Example 15**

*Generating a New File of Random Data*

You can generate a new file with random numbers. First specify `NEW` and then use `REPEAT` to specify the number of cases you want:

```
NEW
RSEED 359
REPEAT 200
LET A=ZRN()
DSAVE filename
```
Example 16
Standardizing Age

This example uses the CHILDREN data set:

<table>
<thead>
<tr>
<th>SEX$</th>
<th>AGE</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>Female</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>Female</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>Female</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Female</td>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>Female</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>Female</td>
<td>5</td>
<td>13</td>
</tr>
<tr>
<td>Female</td>
<td>6</td>
<td>14</td>
</tr>
<tr>
<td>Male</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>Male</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Male</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>Male</td>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>Male</td>
<td>6</td>
<td>9</td>
</tr>
<tr>
<td>Male</td>
<td>4</td>
<td>11</td>
</tr>
<tr>
<td>Male</td>
<td>5</td>
<td>12</td>
</tr>
</tbody>
</table>

We create and list a variable AGESTAND that contains the standardized values of AGE.

The input is:

```
USE CHILDREN
SORT SEX$
LET AGESTAND=AGE
STANDARDIZE AGESTAND
LIST AGE AGESTAND
```

The output is:

<table>
<thead>
<tr>
<th>Case</th>
<th>AGE</th>
<th>AGESTAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5.000000</td>
<td>-0.237</td>
</tr>
<tr>
<td>2</td>
<td>6.000000</td>
<td>0.593</td>
</tr>
<tr>
<td>3</td>
<td>5.000000</td>
<td>-0.237</td>
</tr>
<tr>
<td>4</td>
<td>3.000000</td>
<td>-1.898</td>
</tr>
<tr>
<td>5</td>
<td>5.000000</td>
<td>-0.237</td>
</tr>
<tr>
<td>6</td>
<td>5.000000</td>
<td>-0.237</td>
</tr>
<tr>
<td>7</td>
<td>6.000000</td>
<td>0.593</td>
</tr>
<tr>
<td>8</td>
<td>6.000000</td>
<td>0.593</td>
</tr>
<tr>
<td>9</td>
<td>4.000000</td>
<td>-1.068</td>
</tr>
<tr>
<td>10</td>
<td>6.000000</td>
<td>0.593</td>
</tr>
<tr>
<td>11</td>
<td>8.000000</td>
<td>2.254</td>
</tr>
<tr>
<td>12</td>
<td>6.000000</td>
<td>0.593</td>
</tr>
<tr>
<td>13</td>
<td>4.000000</td>
<td>-1.068</td>
</tr>
<tr>
<td>14</td>
<td>5.000000</td>
<td>-0.237</td>
</tr>
</tbody>
</table>
AGESTAND now has standardized age values with a mean of 0 and a standard deviation of 1. Remember that standardizing does not change the shape of the distribution. If the data are highly skewed or bimodal before standardizing, they will be so after. Standardizing simply moves the location and rescales the spread of your values.

**Example 17**

**Computing F Statistic**

In other examples, we use the LONGLEY data to fit a multiple regression model with six independent variables and use forward stepping to select a subset of three predictors. The residual sum of squares for the six-variable model is 836,424.056 with 9 df; for the three-variable model, the value is 1,323,360.743 with 12 df. Let’s construct an F statistic to compare the two models (that is, to test $b_4 = b_5 = b_6 = 0$):

\[
\text{CALC } \frac{(1323360.743 - 836424.056)/3}{(836424.056/9)}
\]

The calculator returns 1.746.

Finding the p-value associated with F. This expression requests the p-value for $F = 1.746$ with numerator df = 3 and denominator df = 9:

\[
\text{CALC } 1 - \text{FCF}(1.746, 3, 9)
\]

The calculator returns the probability 0.227. The inclusion of the extra three variables does not significantly improve the fit of the model.

**Example 18**

**Trimming Cases for Selected Variable(s)**

To trim the 0.20 proportion of the largest cases for the variables $X$ and $Y$ by the Separate method in the RAINFALL data file, the input is:

\[
\text{USE RAINFALL} \\
\text{TRIM } X \ Y / \text{TRPROP} = 0.20 \ \text{TREGION} = \text{UPPER} \ \text{METHOD} = \text{SEPARATE}
\]
The result is:

<table>
<thead>
<tr>
<th>Case Number</th>
<th>X</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>23.900</td>
<td>41.000</td>
</tr>
<tr>
<td>2</td>
<td>43.300</td>
<td>52.000</td>
</tr>
<tr>
<td>3</td>
<td>36.300</td>
<td>18.700</td>
</tr>
<tr>
<td>4</td>
<td>40.600</td>
<td>.</td>
</tr>
<tr>
<td>5</td>
<td>57.000</td>
<td>40.000</td>
</tr>
<tr>
<td>6</td>
<td>52.500</td>
<td>29.200</td>
</tr>
<tr>
<td>7</td>
<td>46.100</td>
<td>51.000</td>
</tr>
<tr>
<td>8</td>
<td>.</td>
<td>17.600</td>
</tr>
<tr>
<td>9</td>
<td>.</td>
<td>46.600</td>
</tr>
<tr>
<td>10</td>
<td>23.700</td>
<td>.</td>
</tr>
</tbody>
</table>

Here, trimming by Separate method trimmed the 0.20 proportion of the largest observations from each of the X and Y variables separately. The largest 0.20 proportion of X, i.e. the 2 largest observations, 142 and 112.6, get trimmed, and the largest 0.20 proportion of Y, i.e. the 2 largest observations, 55 and 57, get trimmed.

To trim the same data by the Listwise method, the input is:

```
USE RAINFALL
TRIM X Y / TRPROP= 0.20 TREGION = UPPER METHOD = LISTWISE
```

The result is:

<table>
<thead>
<tr>
<th>Case Number</th>
<th>X</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>23.900</td>
<td>41.000</td>
</tr>
<tr>
<td>2</td>
<td>43.300</td>
<td>52.000</td>
</tr>
<tr>
<td>3</td>
<td>36.300</td>
<td>18.700</td>
</tr>
<tr>
<td>4</td>
<td>40.600</td>
<td>55.000</td>
</tr>
<tr>
<td>5</td>
<td>57.000</td>
<td>40.000</td>
</tr>
<tr>
<td>6</td>
<td>52.500</td>
<td>29.200</td>
</tr>
<tr>
<td>7</td>
<td>46.100</td>
<td>51.000</td>
</tr>
<tr>
<td>8</td>
<td>142.000</td>
<td>17.600</td>
</tr>
<tr>
<td>9</td>
<td>112.600</td>
<td>46.600</td>
</tr>
<tr>
<td>10</td>
<td>23.700</td>
<td>57.000</td>
</tr>
</tbody>
</table>
Here, trimming by the Listwise method first trimmed the 0.20 proportion of the largest observations from each of the $X$ and $Y$ variables separately and then excluded cases listwise i.e. completely excluded all those cases, which had observations excluded for at least one of $X$ or $Y$ previously. Thus, cases 8 and 9 which are trimmed by the Separate method for $X$ only, are also excluded for $Y$ and, cases 4 and 10 which are trimmed by the Separate method for $Y$ only are also excluded for $X$. Only the shaded cases will be used in further analysis.

**Example 19**

**Stacking Variables**

Consider CRIMESTAT data set, we have information on the number of particular type of crimes (CRIMES$) done by MALES and Females. The two variables MALES and Females can be stacked on one column for any statistical analysis. For this the input is:

```
USE CRIMESTAT
STACK MALES FEMALES
```

The result is:

```
Case | GROUP$ | VARIABLE   | CRIME$            
-----+--------+------------+-------------------
1    | MALES  | 12904.000  | murder            
2    | MALES  | 28865.000  | rape              
3    | MALES  | 105401.000 | robbery           
4    | MALES  | 211228.000 | assault           
5    | MALES  | 326959.000 | burglary          
6    | MALES  | 744423.000 | larceny           
7    | MALES  | 97835.000  | auto              
8    | MALES  | 13129.000  | arson             
9    | MALES  | 416735.000 | battery           
10   | MALES  | 46286.000  | forgery           
11   | MALES  | 151773.000 | fraud             
12   | MALES  | 5624.000   | embezzle          
13   | MALES  | 181600.000 | vandal            
14   | MALES  | 134210.000 | weapons           
15   | MALES  | 29584.000  | vice              
16   | MALES  | 74602.000  | sex               
17   | MALES  | 562754.000 | drugs             
18   | MALES  | 21995.000  | gambling          
19   | MALES  | 35553.000  | family            
20   | MALES  | 1208416.000| dui               
21   | MALES  | 726214.000 | drunk             
22   | MALES  | 435198.000 | disorderly        
23   | MALES  | 24592.000  | vagrancy          
24   | MALES  | 53808.000  | runaway           
25   | FEMALES| 1815.000   | murder            
26   | FEMALES| 303.000    | rape              
27   | FEMALES| 8639.000   | robbery           
28   | FEMALES| 32926.000  | assault           
29   | FEMALES| 26753.000  | burglary          
30   | FEMALES| 334053.000 | larceny           
31   | FEMALES| 10093.000  | auto              
32   | FEMALES| 2003.000   | arson             
33   | FEMALES| 75937.000  | battery           
34   | FEMALES| 23181.000  | forgery
```
Chapter 4

Example 20

Number of cases, number of variables, and number of columns in the file

You can find the case number, number of cases, and number of variables in the active data file; find the column number of the specified variable in the data file. Let us consider WORLDMM data set, the input is:

USE WORLDMM.SYZ
LET NCASES=NCASE()
LET NVARI=NVAR()
LET CASES=CASE()
LET COLM=COLUMN(DEATH_RT)
LIST NCASES NVARI CASES COLM

The output is:

<table>
<thead>
<tr>
<th>Case ID</th>
<th>NCASES</th>
<th>NVARI</th>
<th>CASES</th>
<th>COLM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Finland</td>
<td>30.000</td>
<td>11.000</td>
<td>1.000</td>
<td>3.000</td>
</tr>
<tr>
<td>France</td>
<td>30.000</td>
<td>11.000</td>
<td>2.000</td>
<td>3.000</td>
</tr>
<tr>
<td>Spain</td>
<td>30.000</td>
<td>11.000</td>
<td>3.000</td>
<td>3.000</td>
</tr>
<tr>
<td>UK</td>
<td>30.000</td>
<td>11.000</td>
<td>4.000</td>
<td>3.000</td>
</tr>
<tr>
<td>Italy</td>
<td>30.000</td>
<td>11.000</td>
<td>5.000</td>
<td>3.000</td>
</tr>
<tr>
<td>Sweden</td>
<td>30.000</td>
<td>11.000</td>
<td>6.000</td>
<td>3.000</td>
</tr>
<tr>
<td>Hungary</td>
<td>30.000</td>
<td>11.000</td>
<td>7.000</td>
<td>3.000</td>
</tr>
<tr>
<td>Germany</td>
<td>30.000</td>
<td>11.000</td>
<td>8.000</td>
<td>3.000</td>
</tr>
<tr>
<td>Gambia</td>
<td>30.000</td>
<td>11.000</td>
<td>9.000</td>
<td>3.000</td>
</tr>
<tr>
<td>Iraq</td>
<td>30.000</td>
<td>11.000</td>
<td>10.000</td>
<td>3.000</td>
</tr>
<tr>
<td>Ethiopia</td>
<td>30.000</td>
<td>11.000</td>
<td>11.000</td>
<td>3.000</td>
</tr>
<tr>
<td>Guinea</td>
<td>30.000</td>
<td>11.000</td>
<td>12.000</td>
<td>3.000</td>
</tr>
<tr>
<td>Mali</td>
<td>30.000</td>
<td>11.000</td>
<td>13.000</td>
<td>3.000</td>
</tr>
<tr>
<td>Libya</td>
<td>30.000</td>
<td>11.000</td>
<td>14.000</td>
<td>3.000</td>
</tr>
<tr>
<td>Somalia</td>
<td>30.000</td>
<td>11.000</td>
<td>15.000</td>
<td>3.000</td>
</tr>
<tr>
<td>Sudan</td>
<td>30.000</td>
<td>11.000</td>
<td>16.000</td>
<td>3.000</td>
</tr>
<tr>
<td>Turkey</td>
<td>30.000</td>
<td>11.000</td>
<td>17.000</td>
<td>3.000</td>
</tr>
<tr>
<td>Algeria</td>
<td>30.000</td>
<td>11.000</td>
<td>18.000</td>
<td>3.000</td>
</tr>
<tr>
<td>Yemen</td>
<td>30.000</td>
<td>11.000</td>
<td>19.000</td>
<td>3.000</td>
</tr>
<tr>
<td>Argentina</td>
<td>30.000</td>
<td>11.000</td>
<td>20.000</td>
<td>3.000</td>
</tr>
<tr>
<td>Barbados</td>
<td>30.000</td>
<td>11.000</td>
<td>21.000</td>
<td>3.000</td>
</tr>
<tr>
<td>Bolivia</td>
<td>30.000</td>
<td>11.000</td>
<td>22.000</td>
<td>3.000</td>
</tr>
<tr>
<td>Brazil</td>
<td>30.000</td>
<td>11.000</td>
<td>23.000</td>
<td>3.000</td>
</tr>
<tr>
<td>Canada</td>
<td>30.000</td>
<td>11.000</td>
<td>24.000</td>
<td>3.000</td>
</tr>
<tr>
<td>Chile</td>
<td>30.000</td>
<td>11.000</td>
<td>25.000</td>
<td>3.000</td>
</tr>
</tbody>
</table>
References


Chapter 5

List, Sort, and Select

Laszlo Engelman

This chapter describes these items on the Data menu:

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>List Cases</td>
<td>Lists all or selected variables in a data file.</td>
</tr>
<tr>
<td>Sort File</td>
<td>Sorts the cases in a file according to values of one or more numeric or string variables.</td>
</tr>
<tr>
<td>Select Cases</td>
<td>Specifies a condition to select a subset of cases for analysis. For example, using $AGE &gt; 21$ restricts subsequent analyses to adults, omitting any case where the value of $AGE \leq 21$.</td>
</tr>
</tbody>
</table>

**List Cases Dialog Box**

Often during the steps of an analysis, you need to stop to view data values. For example, you may want to see values of specific variables side by side for a few selected cases, check that transformations or recoding worked as intended, or scan other data values for cases with extreme residuals.

To open the List Cases dialog box, from the menus choose:

Data
List Cases...
List Cases displays the values of selected variables in the order they are selected. To display cases you can select from options: All cases, or Cases 1 through n where n is the number you specify. If no variables are selected, all variables are listed. You can also use an ID variable to select a numeric or string variable whose values are listed before other values for each case. When you want to list more than five variables across the panel, specify a wide page format (PAGE WIDE). This enables you to print up to eight variables (plus a case number or label) on each line, in both modes of output display (CLASSIC OFF or ON).

The following additional options are available:

**Format.** Specifies a column width to use for all variables. Any variable value shorter than this width is right-justified. Any value longer than the width is truncated. In addition, define the number of decimal places to print for numeric variables.

**Labels.** Displays category labels instead of values.

**Picture Format**

To display your data, you can specify a picture format using the FORMAT option on the LIST command. You use symbols to tell SYSTAT exactly how to format your listing. This gives you a greater control over the appearance of your output. The picture format also enables you to:

- Specify the number of characters or digits in each field.
List, Sort, and Select

- Specify the number of spaces between variables.
- Specify the number of digits of numeric data to display after the decimal point for each variable individually.
- Choose whether to left- or right-justify the values within a field.
- Insert characters, such as vertical bars (|), to separate columns.

You must enclose your symbol strings in quotation marks. Use a #, >, <, or $ for each digit or character. Numeric variables are always right-justified. For string variables, the symbol you use determines whether it is left- or right-justified:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>blank</td>
<td>Separate fields</td>
</tr>
<tr>
<td>&gt;</td>
<td>Right-justify</td>
</tr>
<tr>
<td>&lt;</td>
<td>Left-justify</td>
</tr>
<tr>
<td>$</td>
<td>Do not justify</td>
</tr>
<tr>
<td>.</td>
<td>Decimal point</td>
</tr>
<tr>
<td>#</td>
<td>Digit</td>
</tr>
<tr>
<td>Y</td>
<td>Years</td>
</tr>
<tr>
<td>M</td>
<td>Months</td>
</tr>
<tr>
<td>D</td>
<td>Days</td>
</tr>
<tr>
<td>h</td>
<td>Hours</td>
</tr>
<tr>
<td>m</td>
<td>Minutes</td>
</tr>
<tr>
<td>s</td>
<td>Seconds</td>
</tr>
</tbody>
</table>

If you include any other symbol (such as a vertical bar) in the FORMAT statement, it is printed “as is” for every case. Use the date symbols (Y, M, D, h, m, and s) to format dates in your output.

**Additional Features for Listing Cases**

When listing data, there are other options you may want to use. For example, you can:

- Display eight variables per panel instead of five (the default) by specifying a wide output format (PAGE WIDE).

You can also use the PRINT command to list data values without case numbers and variable names.

**Sort Dialog Box**

To open the Sort dialog box, from the menus choose:

Data
   Sort File...
Chapter 5

Sort orders all the cases in a file according to the variable you select. For example, you could sort cases by gender, and subsequently by age within each gender. In addition, you can specify whether values are sorted in ascending or descending order. If you do not specify any variables, SYSTAT sorts by the first variable in the file.

- **Ascending/Descending.** Sorts cases in either ascending order (1, 2, 3... or a, b, c...) or descending order (3, 2, 1...).
- **Save file.** Specifies an output file for the sorted data. If this option is selected and a filename is not specified, SYSTAT prompts you to specify a filename. If you do not select this option, SYSTAT saves the sorted data in a temporary file.

**Sort Order**

By default, SYSTAT orders cases in an ascending order, (1, 2, ..., n) for numeric variables and, as follows, for string variables:

```
! "$ % &'() *+,-./ 0123456789 :;<==>@ABCDEFGHIJKLMNOPQRSTUVWXYZ[
\]^_abcdefghijklmnpqrstuvwxyz{||~```

---

**Data: Sort File**

The image shows a data sort file interface with available and selected variables, an order option for ascending or descending, and a save file option.
Words are sorted alphabetically with words in upper case preceding those in lower case.

- If you sort a string variable containing numeric values, those values are sorted from left to right, rather than small to big: 1, 12, 150, 2, 31, 4000, 5.4, etc.

- Cases with missing data for the sort variable are placed at the beginning of the file.

**Select Cases Dialog Box**

To open the Select Cases dialog box, from the menus choose:

Data
Select Cases…

Select Cases restricts subsequent analyses to cases that meet the conditions you specify. Unselected cases remain in the data file, but are excluded from subsequent
analyses until Select is turned off. For example, you could restrict your analysis to respondents of a certain age, gender, or both.

**Rules for expressions.** You can use any mathematically valid combination of variables, numbers, functions, and operators. You can also use any combination of selecting, pasting, and typing necessary to build the test condition. Finally, you can specify any number of conditions, connecting them with a logical AND or OR. Use parentheses if needed for logic or clarity.

- If the expression contains any character values, they must be enclosed in single or double quotation marks. Character values are case sensitive.
- Arguments for functions must be inside parentheses, for example, \( \text{LOG}(\text{WEIGHT}) \) and \( \text{SQR}(\text{INCOME}) \).

The following options are available:

- **Mode of input.** Gives an option to specify selection condition by selecting available variables (in the expression) and operators or by typing the selection condition.
- **Complete.** Selects cases with no values missing.
- **Turn off.** Turns off case selection so that all cases are used in the subsequent analyses. You can also turn off case selection by closing SYSTAT, opening a new data file, or typing `SELECT` in the command area.

You can also select cases in graphs using the region and lasso tools available in the selection tool of the Graph editor. Selection can be toggled using the invert case selection icon in the data toolbar.

**Extract Dialog Box**

To open the Extract dialog box, from the menus choose:

Data
  Extract…
List, Sort, and Select

Executes transformations and allows you to select a subset of variables with or without an additional case selection condition. So, it will enable even if SELECT is not active.

- **Available variable(s).** Displays all variables of active data file.
- **Selected variable(s).** The list of variables which are subset of active data file which can be extracted to.
- **Select cases.** When SELECT condition is specified through Select Cases dialog box, the edit box shows the specified condition, on selected variables.
- **Extract to.** You can specify the filename to extract the data which satisfy the above conditions.

**Using Commands**

To list cases:

```
LIST varlist / FORMAT=(m, n) or 'picture format'  N = n LABEL
```

To sort cases:

```
SORT varlist / D or A or a list of D's and A's
```
To select cases:

```
SELECT exprn1 AND exprn2 OR exprn3...
```

or

```
SELECT COMPLETE()
```

or

```
SELECT CASE() = expression
```

To extract selected case(s) of variable(s) specified in `varlist`:

```
EXTRACT filename / VARIABLES = {varlist}
```

**Examples**

**Example 1**

**Listing All Variables and Cases**

To list all values in the `MINIWRLD` data file, the input is:

```
USE MINIWRLD
LIST
```

The first three cases of `MINIWRLD` are shown below:

<table>
<thead>
<tr>
<th>Case ID</th>
<th>COUNTRY$</th>
<th>POP_1990</th>
<th>URBAN</th>
<th>BIRTH_RT</th>
<th>BABYMORT</th>
<th>GDP_CAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>France</td>
<td>France</td>
<td>56.358</td>
<td>73.000</td>
<td>14.000</td>
<td>6.000</td>
<td>14542.657</td>
</tr>
<tr>
<td>Greece</td>
<td>Greece</td>
<td>10.028</td>
<td>65.000</td>
<td>11.000</td>
<td>10.000</td>
<td>5614.184</td>
</tr>
<tr>
<td>Switzerland</td>
<td>Switzerland</td>
<td>6.742</td>
<td>58.000</td>
<td>12.000</td>
<td>5.000</td>
<td>17723.499</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Case ID</th>
<th>EDUC</th>
<th>HEALTH</th>
<th>MIL</th>
<th>GOV$</th>
<th>LIFEEXPM</th>
<th>LIFEEXPF</th>
<th>LITERACY</th>
</tr>
</thead>
<tbody>
<tr>
<td>France</td>
<td>648.069</td>
<td>728.249</td>
<td>432.780</td>
<td>Democracy</td>
<td>73.000</td>
<td>82.000</td>
<td>99.000</td>
</tr>
<tr>
<td>Greece</td>
<td>115.000</td>
<td>158.700</td>
<td>260.400</td>
<td>Democracy</td>
<td>75.000</td>
<td>80.000</td>
<td>95.000</td>
</tr>
<tr>
<td>Switzerland</td>
<td>853.538</td>
<td>1209.077</td>
<td>330.769</td>
<td>Democracy</td>
<td>75.000</td>
<td>83.000</td>
<td>99.000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Case ID</th>
<th>GROUP$</th>
<th>B_TO_D</th>
</tr>
</thead>
<tbody>
<tr>
<td>France</td>
<td>Europe</td>
<td>1.556</td>
</tr>
<tr>
<td>Greece</td>
<td>Europe</td>
<td>1.222</td>
</tr>
<tr>
<td>Switzerland</td>
<td>Europe</td>
<td>1.333</td>
</tr>
</tbody>
</table>

The individual cases are too long to fit on one line (80 characters), so each case spreads across three lines. You can specify a wider output format using `PAGE WIDE`.

You can also display separate panels for selected variables:

```
LIST COUNTRY$ .. BABYMORT
LIST GDP_CAP .. GOV$
```
Example 2
Listing Cases Subsets for Selected Variables

Here we list only cases 1 through 10 of the variables COUNTRY$, LITERACY, URBAN, and BIRTH_RT in the MINIWRLD data file.

The input is:

```
USE MINIWRLD
LIST COUNTRY$ LITERACY URBAN BIRTH_RT / N = 10
```

The output is:

<table>
<thead>
<tr>
<th>Case ID</th>
<th>COUNTRY$</th>
<th>LITERACY</th>
<th>URBAN</th>
<th>BIRTH_RT</th>
</tr>
</thead>
<tbody>
<tr>
<td>France</td>
<td>France</td>
<td>99.000</td>
<td>73.000</td>
<td>14.000</td>
</tr>
<tr>
<td>Greece</td>
<td>Greece</td>
<td>95.000</td>
<td>65.000</td>
<td>11.000</td>
</tr>
<tr>
<td>Switzerland</td>
<td>Switzerland</td>
<td>99.000</td>
<td>58.000</td>
<td>12.000</td>
</tr>
<tr>
<td>Spain</td>
<td>Spain</td>
<td>97.000</td>
<td>91.000</td>
<td>11.000</td>
</tr>
<tr>
<td>UK</td>
<td>UK</td>
<td>99.000</td>
<td>76.000</td>
<td>14.000</td>
</tr>
<tr>
<td>Hungary</td>
<td>Hungary</td>
<td>99.000</td>
<td>54.000</td>
<td>12.000</td>
</tr>
<tr>
<td>Iraq</td>
<td>Iraq</td>
<td>55.000</td>
<td>68.000</td>
<td>46.000</td>
</tr>
<tr>
<td>Pakistan</td>
<td>Pakistan</td>
<td>26.000</td>
<td>28.000</td>
<td>43.000</td>
</tr>
<tr>
<td>Ethiopia</td>
<td>Ethiopia</td>
<td>55.200</td>
<td>14.000</td>
<td>45.000</td>
</tr>
<tr>
<td>Afghanistan</td>
<td>Afghanistan</td>
<td>12.000</td>
<td>16.000</td>
<td>44.000</td>
</tr>
</tbody>
</table>

Example 3
ID Variables

In this example using the MINIWRLD data file, we use an ID variable, COUNTRY$, to identify the rows of the listing. We also use FORMAT to specify one decimal place in each field.

The input is:

```
USE MINIWRLD
IDVAR COUNTRY$
LIST LITERACY URBAN BIRTH_RT / N = 10 FORMAT = 1
```
Example 4
Group Labels

In this example using the SURVEY2 data file, we add group labels to a listing.

The input is:

```
USE SURVEY2
LABEL MARITAL / 1="NEVER",2="MARRIED",3="DIVORCED",4="SEPARATED"
LABEL EDUCATN / 1,2="DROPOUT",3="HS, GRAD",4,5="COLLEGE",6,7="DEGREE"
LIST MARITAL EDUCATN AGE SEX / N = 12 LABEL
```

The output is:

```
Case | MARITAL | EDUCATN | AGE   | SEX  
-----|---------|---------|-------|------
 1   | DIVORCED| COLLEGE| 58.000| Male 
 2   | MARRIED | HS GRAD | 45.000| Female
 3   | DIVORCED| HS GRAD | 50.000| Female
 4   | SEPARATED | HS GRAD | 33.000| Female
 5   | MARRIED | HS GRAD | 24.000| Male 
 6   | MARRIED | DROPOUT | 58.000| Female
 7   | NEVER   | HS GRAD | 22.000| Male 
 8   | MARRIED | DROPOUT | 30.000| Male 
 9   | MARRIED | HS GRAD | 57.000| Female
10  | MARRIED | DROPOUT | 39.000| Male 
11  | MARRIED | HS GRAD | 23.000| Female
12  | SEPARATED | DROPOUT | 55.000| Female
```
Example 5

Picture Format: Displaying Many Variables in One Panel

To illustrate the picture format, let us use an example from a 1980 survey of depression. Often it is beneficial to see many variables in a single panel so that cases can be compared. To do this, use the FORMAT option to specify a picture format.

The input is:

```
USE SURVEY
SORT ID
LET id$ = SQZ$(STR$(ID), '.000')
IDVAR id$
LET v1 = SEX
LET v2 = AGE
LET v3 = MARITAL
LET v4 = EDUCATN
LET v5 = EMPLOY
LET v6 = INCOME
LET v7 = RELIGION
LET v8 = BLUE
LET v9 = DEPRESS
LET v10 = LONELY
LET v11 = CRY
LET v12 = SAD
LET v13 = FEARFUL
LET v14 = CASE
LET v15 = DRINK
PUSH CLASSIC
CLASSIC ON
NODE 'LIST SEX .. FEARFUL CASE DRINK'
LIST v1 .. v15 / FORMAT = '# ## # # # ## # # # # # # # ### #'
NODE
POP CLASSIC
```

The first 10 cases of the output are shown below:

<table>
<thead>
<tr>
<th>Case ID</th>
<th>V1</th>
<th>V2</th>
<th>V3</th>
<th>V4</th>
<th>V5</th>
<th>V6</th>
<th>V7</th>
<th>V8</th>
<th>V9</th>
<th>V10</th>
<th>V11</th>
<th>V12</th>
<th>V13</th>
<th>V14</th>
<th>V15</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>68</td>
<td>5</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>58</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>15</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>45</td>
<td>3</td>
<td>3</td>
<td>9</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>50</td>
<td>3</td>
<td>3</td>
<td>9</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>33</td>
<td>4</td>
<td>3</td>
<td>35</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>24</td>
<td>3</td>
<td>1</td>
<td>11</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>58</td>
<td>2</td>
<td>2</td>
<td>5</td>
<td>11</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>22</td>
<td>1</td>
<td>3</td>
<td>9</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>47</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>23</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>30</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>35</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Example 6
Picture Format: Displaying Dates

With the FORMAT option of LIST, you can list values in a specified date or time format. For this example, we use the SICKDATE data file, which lists the date each patient’s illness was diagnosed (DIAGDATE) and the date each died (MORTDATE). These dates are listed in day-of-the-century format.

<table>
<thead>
<tr>
<th>NAME$</th>
<th>DIAGDATE</th>
<th>MORTDATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jones</td>
<td>32153</td>
<td>33151</td>
</tr>
<tr>
<td>Smith</td>
<td>31255</td>
<td>32351</td>
</tr>
<tr>
<td>Williams</td>
<td>30251</td>
<td>32512</td>
</tr>
<tr>
<td>Jackson</td>
<td>29351</td>
<td>30251</td>
</tr>
<tr>
<td>Moore</td>
<td>28351</td>
<td>29351</td>
</tr>
<tr>
<td>Iverson</td>
<td>29351</td>
<td>30251</td>
</tr>
<tr>
<td>Brown</td>
<td>27351</td>
<td>32512</td>
</tr>
<tr>
<td>Long</td>
<td>26351</td>
<td>28351</td>
</tr>
<tr>
<td>Nelson</td>
<td>28351</td>
<td>30251</td>
</tr>
<tr>
<td>Dennison</td>
<td>24351</td>
<td>25351</td>
</tr>
</tbody>
</table>

To demonstrate how the FORMAT option displays these dates, we use three different formats:

- MM/DD/YY
- MMM-DD,YYYY
- DD.MMM,YY

The input is:

```
USE SICKDATE
LIST DIAGDATE DIAGDATE DIAGDATE MORTDATE MORTDATE MORTDATE/FORMAT =,
'MM/DD/YYYY  MMM-DD,YYYY DD.MM,YY' 
```

The output is:

<table>
<thead>
<tr>
<th>Case</th>
<th>DIAGDATE</th>
<th>DIAGDATE</th>
<th>DIAGDATE</th>
<th>MORTDATE</th>
<th>MORTDATE</th>
<th>MORTDATE</th>
</tr>
</thead>
</table>
Example 7
Simple Sort

To sort the children on SEX$ using the CHILDREN file, the input is:

```
USE CHILDREN
SORT SEX$
LIST
```

The output is:

```
Case | SEX$      | AGE      | N
-----+----------+----------+-------
 1    | Female   | 5.000000 | 1.000000
 2    | Female   | 6.000000 | 4.000000
 3    | Female   | 5.000000 | 5.000000
 4    | Female   | 3.000000 | 8.000000
 5    | Female   | 5.000000 | 10.000000
 6    | Female   | 5.000000 | 13.000000
 7    | Female   | 6.000000 | 14.000000
 8    | Male     | 6.000000 | 2.000000
 9    | Male     | 4.000000 | 3.000000
10   | Male     | 6.000000 | 6.000000
11   | Male     | 8.000000 | 7.000000
12   | Male     | 6.000000 | 9.000000
13   | Male     | 4.000000 | 11.000000
14   | Male     | 5.000000 | 12.000000
```

Cases where SEX$ equals Female come before those where SEX$ equals Male.
(N represents the original position of each case.)

Example 8
Nested Sort

If you select more than one sort variable, SYSTAT does a nested sort. For example, if you select SEX$ and AGE from the CHILDREN data file in that order, SYSTAT first sorts the males and females into two groups and then sorts each group from youngest to oldest.

The input is:

```
USE CHILDREN
SORT SEX$ AGE
LIST
```
The output is:

<table>
<thead>
<tr>
<th>Case</th>
<th>SEX$</th>
<th>AGE</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Female</td>
<td>3.000000</td>
<td>8.000000</td>
</tr>
<tr>
<td>2</td>
<td>Female</td>
<td>5.000000</td>
<td>1.000000</td>
</tr>
<tr>
<td>3</td>
<td>Female</td>
<td>5.000000</td>
<td>5.000000</td>
</tr>
<tr>
<td>4</td>
<td>Female</td>
<td>5.000000</td>
<td>10.000000</td>
</tr>
<tr>
<td>5</td>
<td>Female</td>
<td>5.000000</td>
<td>13.000000</td>
</tr>
<tr>
<td>6</td>
<td>Female</td>
<td>6.000000</td>
<td>4.000000</td>
</tr>
<tr>
<td>7</td>
<td>Female</td>
<td>6.000000</td>
<td>14.000000</td>
</tr>
<tr>
<td>8</td>
<td>Male</td>
<td>4.000000</td>
<td>3.000000</td>
</tr>
<tr>
<td>9</td>
<td>Male</td>
<td>4.000000</td>
<td>11.000000</td>
</tr>
<tr>
<td>10</td>
<td>Male</td>
<td>5.000000</td>
<td>12.000000</td>
</tr>
<tr>
<td>11</td>
<td>Male</td>
<td>6.000000</td>
<td>2.000000</td>
</tr>
<tr>
<td>12</td>
<td>Male</td>
<td>6.000000</td>
<td>6.000000</td>
</tr>
<tr>
<td>13</td>
<td>Male</td>
<td>6.000000</td>
<td>9.000000</td>
</tr>
<tr>
<td>14</td>
<td>Male</td>
<td>8.000000</td>
<td>7.000000</td>
</tr>
</tbody>
</table>

Within each gender group, SYSTAT arranges the cases so that the values for \( AGE \) go from smallest to largest.

**Example 9**

**Sorting in Ascending and Descending Order**

You can sort data in ascending and descending order in sequence given. Specify D and A for each variable, the way you want to get data sorted. For example:

```
USE CHILDREN
SORT SEX$ AGE / D A
LIST
```

sorts the file in descending order for \( SEX\) (males precede females) and then, within each sex, in ascending order for \( AGE \) (from youngest to oldest).

The output is:

<table>
<thead>
<tr>
<th>Case</th>
<th>SEX$</th>
<th>AGE</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Male</td>
<td>4.000000</td>
<td>3.000000</td>
</tr>
<tr>
<td>2</td>
<td>Male</td>
<td>4.000000</td>
<td>11.000000</td>
</tr>
<tr>
<td>3</td>
<td>Male</td>
<td>5.000000</td>
<td>12.000000</td>
</tr>
<tr>
<td>4</td>
<td>Male</td>
<td>6.000000</td>
<td>2.000000</td>
</tr>
<tr>
<td>5</td>
<td>Male</td>
<td>6.000000</td>
<td>6.000000</td>
</tr>
<tr>
<td>6</td>
<td>Male</td>
<td>6.000000</td>
<td>9.000000</td>
</tr>
<tr>
<td>7</td>
<td>Male</td>
<td>8.000000</td>
<td>7.000000</td>
</tr>
<tr>
<td>8</td>
<td>Female</td>
<td>3.000000</td>
<td>8.000000</td>
</tr>
<tr>
<td>9</td>
<td>Female</td>
<td>5.000000</td>
<td>1.000000</td>
</tr>
<tr>
<td>10</td>
<td>Female</td>
<td>5.000000</td>
<td>5.000000</td>
</tr>
<tr>
<td>11</td>
<td>Female</td>
<td>5.000000</td>
<td>10.000000</td>
</tr>
<tr>
<td>12</td>
<td>Female</td>
<td>5.000000</td>
<td>13.000000</td>
</tr>
<tr>
<td>13</td>
<td>Female</td>
<td>6.000000</td>
<td>4.000000</td>
</tr>
<tr>
<td>14</td>
<td>Female</td>
<td>6.000000</td>
<td>14.000000</td>
</tr>
</tbody>
</table>
**Example 10**  
*Selecting Subset of Cases*

Using the *USSTATES* data file, select states in the Pacific and Mountain divisions that have low rainfall.

The input is:

```plaintext
USE USSTATES
SELECT (RAIN < 15) AND (DIVISION$ = 'Pacific' OR,
                        DIVISION$ = 'Mountain')
LIST RAIN STATE$
```

The output is:

<table>
<thead>
<tr>
<th>Case</th>
<th>RAIN</th>
<th>STATE$</th>
</tr>
</thead>
<tbody>
<tr>
<td>38</td>
<td>11.000</td>
<td>MT</td>
</tr>
<tr>
<td>39</td>
<td>11.000</td>
<td>ID</td>
</tr>
<tr>
<td>42</td>
<td>10.000</td>
<td>NM</td>
</tr>
<tr>
<td>43</td>
<td>7.000</td>
<td>AZ</td>
</tr>
<tr>
<td>44</td>
<td>14.000</td>
<td>UT</td>
</tr>
<tr>
<td>45</td>
<td>8.000</td>
<td>NV</td>
</tr>
</tbody>
</table>

**Example 11**  
*Selecting Cases with Equal Values of Some Variables*

You can select cases for which values of some specified pairs of variables are equal. For example, the commands:

```plaintext
NEW
INPUT A B C D
2 2 2 2
3 3 3 4
5 5 1 1
6 7 0 1
8 9 1 0
~
SELECT (A = B) AND (B = C)
LIST
```

selects all the cases where the “(A = B) and (B = C)” condition is satisfied i.e. all the cases for which the values of all the variables $A$, $B$, and $C$ are equal.

Thus the output is:

<table>
<thead>
<tr>
<th>Case</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.000</td>
<td>2.000</td>
<td>2.000</td>
<td>2.000</td>
</tr>
<tr>
<td>2</td>
<td>3.000</td>
<td>3.000</td>
<td>3.000</td>
<td>4.000</td>
</tr>
</tbody>
</table>
Now the commands:

```
SELECT (A = B) and (B = C) and (C = D)
LIST
```

selects cases where “(A = B) and (B = C) and (C = D)”. Here, you can see that case 2 has not got selected as in the earlier case. In this case the output is:

```
Case | A    | B    | C    | D
-----+-------+-------+-------+-------
1    | 2.000 | 2.000 | 2.000 | 2.000
```

However, SELECT A = B = C does not compare A, B, C simultaneously.

The commands:

```
SELECT A=B=C
LIST
```

results in the following:

The first two variables i.e., A and B are compared and if they are equal for a case then the value '1' is being stored; otherwise '0' is stored. And then if the next variable C equals the stored value then that case is selected. Hence, for the above command the output is:

```
Case | A    | B    | C    | D
-----+-------+-------+-------+-------
3    | 5.000 | 5.000 | 1.000 | 1.000
4    | 6.000 | 7.000 | 0.000 | 1.000
```

Similarly, the case for SELECT A = B = C = D.

First, A and B are compared and if they are equal then '1' is stored; otherwise '0' is stored. Then, if the values of the next variable C equals the stored value (i.e. '0' or '1') then '1' is stored; otherwise '0' is stored for further comparison. Finally, if the last variable D equals the stored value then the case is selected; otherwise not.

Thus for the commands:

```
SELECT A=B=C=D
LIST
```

The output is:

```
Case | A    | B    | C    | D
-----+-------+-------+-------+-------
3    | 5.000 | 5.000 | 1.000 | 1.000
4    | 6.000 | 7.000 | 0.000 | 1.000
5    | 8.000 | 9.000 | 1.000 | 0.000
```
Chapter 6

Grouping Variables and By Groups

For each case, a grouping variable contains a value that identifies group membership. For example, for the string variable \textit{SEX$\$,} the values might be Male and Female, or for the numeric variable \textit{SEX}, the codes might be 1 and 2. The values of grouping variables are used to define categories, cells, subpopulations, or groups of cases for:

- Each factor in a frequency table.
- Analyzing where group means are compared, such as \(t\) tests, analysis of variance, or discriminant analysis.
- Stratifying an analysis—for example, computing descriptive statistics separately for males and females using SYSTAT’s By Groups feature.
- Univariate graphical displays, such as bar charts, pie charts, dot plots, and profile displays.
- Symbols or names that label plot points in bivariate and 3-D scatterplots.

The values of grouping variables can be entered and stored as a column in your data file or generated while executing any SYSTAT procedure via the \texttt{LABEL}, \texttt{ORDER}, and \texttt{RECODE} commands or the \texttt{COD}, \texttt{INC}, and \texttt{CUT} functions.

This chapter describes how to:

- Specify categorical variables.
- Assign labels.
- Order categories and labels for displays and analyses.
- Recode values of numeric or string variables.
- Specify cutpoints to define intervals along a continuous variable.
Identify when the value of a numeric or string variable matches one of a set of codes that you list (1 = yes, 0 = no).

Define By Groups variables for stratified analyses.

*Categorical Variables Dialog Box*

To treat values of a numeric or character variable as categories, click CAT on the Status bar, or from the menus choose:

Data
  Categorical Variables...

SYSTAT treats each value of the selected variables as a discrete category. Thus, if you identify a quantitative variable as categorical, each unique value appears on the plot scale, the values are equally spaced (for example, if the numbers are 1, 3, and 15, they are spaced as 1, 2, 3), and the minimum and maximum data values are limits for the plot scale.

To set individual variables as categorical, use the Category column in the Variable editor and select the check box in the column. You can also set a variable as a categorical at the stage of variable definition itself by using the same procedure as explained above.

**Missing values.** Specifies that cases with a missing value for the categorical variable be included as an additional category.
**Turn off.** Removes any category specifications currently in effect.

**Saving category information.** By default, when you request data to be saved to a SYSTAT data file, any category information that you set will also be saved. The implication of this is that the next time you open the saved data file and run an analysis or draw a graph involving the saved categorical variables, they will be processed as categorical variables without any explicit declaration. If you do not want this to happen, that is, if you want variables to be treated as categorical only after you declare them as such, then uncheck **Save category variable information to data file** in the Data tab of the Edit:Options dialog box.

**NCAT Function**

Use the NCAT function to know the number of category levels in a variable. The syntax is:

```
NCAT (arg1, arg2)
```

`arg1` is a numeric or string variable name. The function returns the number of category levels in `arg1`. `arg2` is an optional argument which can take a value of 0 or 1. If you wish to include a missing value as a category, then use 1 as its value. Else use 0 (default).

**Value Labels Dialog Box**

You can use the Value Labels dialog box or the LABEL command to:

- Assign a character name to a value for use as a label in the output.
- Order categories for graphical displays and statistical analyses.
- Assign new labels for string variables.

To open the Value Labels dialog box, from the menus choose:

Data
  Value Labels...
In the Value Labels dialog box, select a variable and then specify the values and corresponding labels related to that variable. You can define value labels for any number of variables in this manner by simply selecting that variable and then entering values and labels.

You can also specify value labels for a given variable using the Value Label column in the Variable editor.

- Click the Variable editor tab in the Viewspace.
- Click the desired cell in the Value Label column.
- Click the ellipsis button. The Value Labels dialog appears.
- Enter the data values with their desired labels.
- Press OK.

To remove a previous label specification, click the ellipsis button. The Value Labels dialog appears and select Turn off.

**Displaying value labels in the output.** By default, value labels are displayed in the output for all variables for which value labels have been defined; the values themselves are displayed for other variables. You can control the display of value labels using the
Value labels display setting in the Edit :Options dialog or the LDISPLAY command or, from the menus choose:

Edit
  Output Format
    Value Label Display...

The Label menu item will be checked by default. Use the Data setting to suppress the display of value labels, and Both to display value labels as well as data values.

Saving labels with the data. Any value labels that you define will also be saved to the file if you request saving the data after defining the labels. The value label information will be associated with the corresponding variable information and not as a separate variable.

You can also create a variable containing the value labels defined for another variable using the LAB$ function \([LET var$ = LAB$(var)]\), and then save the data file. In this method of saving, however, SYSTAT will not associate the new variable as being the value label equivalent of the old variable.

Use of quotation marks. With the Value Labels dialog box, you do not need to type quotation marks. With commands, if the name of a category or interval is a number, you can omit the quotation marks.

**Category for Missing Codes**

If you want to include a category (group) for cases with missing codes, and you want to label it, your option list should include a period (.) for missing numbers or a space surrounded by quotation marks (" " or ") for missing character data. For example,

\[
\text{LABEL gnp}\$ / ' '=Missing, 'D'=Developed,'U'=Emerging
\]

forms three economic categories for the countries in the OURWORLD file.

**Order of Display Dialog Box**

By default, SYSTAT orders numeric category codes or labeled values in the ascending order of their magnitude, and string category codes or labeled strings in the alphabetical order. You can use Order of Display on the Data menu or the ORDER command to specify how SYSTAT should sort categories or labels for output including table factors, statistical analyses, and graphical displays.
To open the Order of Display dialog box, from the menus choose:

Data
Order of Display...

Select sort. Specify one of the following options for ordering categories:

- **None.** Categories or labels are ordered as SYSTAT first encounters them in the data file.
- **Ascending.** Numeric category codes or labels are ordered from smallest to largest, and string codes or labels, alphabetically. This is the default.
- **Descending.** Numeric category codes or labels are ordered from largest to smallest, and string codes or labels, backward alphabetically.
- **Ascending frequency or Descending frequency.** Categories or labels are ordered by the frequency of cases within each, placing the category or label with the largest (or smallest) frequency first. Use Ascending frequency for an ascending sort and Descending frequency for a descending sort.

Enter sort. Specifies a custom order for codes or labels. Values must be separated by spaces or commas, with string values enclosed in quotation marks (for example, 1, 3, 2, or ‘low,’ ‘high’).
**Sort applies to.** Indicates whether the sort applies to data values or value labels. By default, the sort applies to data values even if value labels are defined. Select Labels if you want the sort to be based on value labels.

**Missing values.** Includes an additional category for missing values if the value of `var` or `var$` is missing.

**Use default order.** Returns to the default order.

ORDER is not a filter. For example, if the data for a variable contain six unique values and you identify three of them using the SORT option, then the remaining (unspecified) values are still considered for categorizing and are automatically sorted in the file order when they are shown in the output.

---

**Recode Dialog Box**

You can use the Recode dialog box or the RECODE command to:

- Group cases into categories
- Recode the values of existing variables
- Create new variables with recoded values

To recode values of numeric or string variables, from the menus choose:

Data  
Transform  
      Recode...
Chapter 6

Select one or more variables in the Available variable(s) list, and press Add to add them to the Recode From column.

**Recode From.** This column accepts either numeric or string variables one on each row.

**Recode To.** By default, the values of the selected variable(s) will be replaced by the new values that you specify. To create a new variable with the recoded values of a selected variable, specify a variable name in the corresponding row of the Recode To column.

All the Recode From variables are recoded based on the old and new values that you specify in the columns provided.

**Old Value.** Specify the value that you want to recode. You can specify multiple values separated by spaces, commas or the ellipsis notation (..). The ellipsis notation will allow you to specify a range of values starting from a given value upto another given value. If any value that you enter here does not exist in the selected variable(s), it will be ignored while recoding. So, it is not necessary that you should only enter values that exist in one of the selected variable(s). Do not enclose strings in quotes; the dialog
takes care of inserting quotes whenever required. Press the ENTER key or the button to add new rows.

**New Value.** Specify the desired recoded value; specify a single number or string as applicable. Strings may include spaces. Do not enclose strings in quotes; the dialog takes care of inserting quotes whenever required.

**Save file.** You can save the file with the recoded values to a SYSTAT data file.

---

**COD, INC and CUT Functions**

The Recode dialog allows you to recode single, multiple or a range of values. For specific kinds of recoding, you can also use the functions COD, INC and CUT. With these, you can:

- Recode values of numeric or string grouping variables: COD.
- Identify when the value of a numeric or string variable matches (1 = yes, 0 = no) one of the specific codes you list: INC.
- Define intervals along a continuous variable: CUT.
- Separate the values of a string variable alphabetically: CUT.

These functions are specified using `LET...` and `IF... THEN LET` statements.

Unlike LABEL and ORDER, these functions modify the actual data values. If you save the data after specifying COD, INC, or CUT, the results are stored in the data file.

---

**Value Recoding: COD**

Use the COD function to recode values of a grouping variable.

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>COD(var,num1,num2,...)</td>
<td>Values of <code>var</code> are replaced: <code>num1</code> becomes 1, <code>num2</code> becomes 2, etc.</td>
</tr>
<tr>
<td>COD(var$,'text1','text2',...)</td>
<td>Values of <code>var$</code> are replaced: <code>text1</code> becomes 1, <code>text2</code> becomes 2, etc.</td>
</tr>
</tbody>
</table>
The COD function always produces consecutive integers (starting with 1) that correspond, respectively, to the original codes specified in the statement. Periods (SYSTAT’s marker for missing data) are stored in the recoded variable for every case whose original value is not among the values specified.

**Comparing Values: INC**

The INC (for included) function compares each value of a numeric or string variable with a list of values that you specify. If a match is found, SYSTAT returns a 1; if a match is not found, a 0.

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>INC(var,num1,num2,…)</td>
<td>A 1 (true) if the value in var matches one of the numbers num1, num2, …; 0 (false) otherwise.</td>
</tr>
<tr>
<td>INC(var$,'text1','text2',…)</td>
<td>A 1 (true) if the value in var$ matches text1, text2, …; 0 (false) otherwise.</td>
</tr>
</tbody>
</table>

**Defining Intervals: CUT**

Use CUT to define intervals on a quantitative (continuous) variable or to divide the values of a string variable alphabetically.

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUT(var,num1,num2,…)</td>
<td>Use to define intervals along a continuous variable. Values in var less than or equal to num1 get value 1. Values greater than num1 and less than or equal to num2 get value 2, etc.</td>
</tr>
<tr>
<td>CUT(var$,'text1','text2',…)</td>
<td>Use to group variables alphabetically. Values in var$ less than text1 (alphabetically), including text1, get value 1. Values between text1 and text2, including text2, get value 2, etc.</td>
</tr>
</tbody>
</table>

**By Groups Dialog Box**

You can use By Groups on the Data menu or the BY command to request separate results for each level of one or more grouping variables. For example, you could obtain separate graphs and statistical analyses for male children, female children, male teens, female teens, and so on. Here, SEX and AGES$ (categorized as child, teen, etc.) would be the grouping variables.
To open the By Groups dialog box, click BY on the status bar or from the menus choose:

Data
   By Groups...

You can select up to 10 grouping variables. After specifying grouping variables, run your analyses as normal; all subsequent SYSTAT procedures and graphs perform analyses separately for each unique combination of the group codes.

Exclude missing. Excludes categories defined by missing values from the analysis. If this option is not selected, a separate category is created for cases with missing values for the grouping variable, and a separate analysis is carried out for them.

Turn off. Turns off by groups so that subsequent analyses are performed across all cases. You can also turn off by groups by quitting SYSTAT or opening a new data file.

By Groups versus Plot Groups

When By Groups is used with a graphical display, each group (or each unique combination of the grouping variables) is displayed in a separate chart, and the scale is set using only the values within the group. For example, if $SEX$ is the grouping variable, separate charts are produced for males and females. The scale for each chart is defined by the maximum and minimum values within each gender.
However, for many graphical displays, GROUP option is available. With this option, displays for all subgroups appear on a single chart and share a common scale.

To obtain separate charts that share a common scale, use the minimum and maximum values of the complete sample to define the scale, rather than using the minimum and maximum values within each particular group.

**Using Commands**

To treat variables as categorical, use

```
CATEGORY varlist / ADD or REPLACE MISS
```

ADD will add varlist to the existing category set and is the default setting. REPLACE will replace the existing category set by varlist. The optional MISS will include missing values as a category in each of the category variables.

To display a list of the currently declared categorical variables, use

```
CATEGORY / ?
```

To turn off category declarations, execute

```
CATEGORY varlist / OFF
```

If you do not specify varlist, all category declarations are turned off.

Use this syntax for labeling numeric values:

```
LABEL varlist / n1='text1', n2='text2',
or
n1,n2,...='text1',
n3,n4,...='text2',
or
n1 .. n2='text1',
n3 .. n4='text2',
```

For string values, specify:

```
LABEL varlist$ / 'oldtext1'=newtext1,
'oldtext2'=newtext2,
```
You can also use VALLABEL instead of LABEL.

Options for LABEL statements include:

\[
\begin{align*}
\text{n1}= & \text{'}text1, n2=\text{'}text2, \ldots \\
n1,n2, \ldots= & \text{'}text1', \\
n3,n4, \ldots= & \text{'}text2', \ldots \\
n1 .. n2= & \text{'}text1' \\
n3 .. n4= & \text{'}text2', \ldots \\
\end{align*}
\]

The values \(n1, n2 \ldots\) in the data file are labeled \(text1, text2\ldots\) respectively.

The values \(n1, n2 \ldots\) are labeled \text{'}text1', the values \(n3, n4 \ldots\) are labeled \text{'}text2', etc. (The values \(n_i\) do not have to be consecutive values or specified in order).

The values \(n1\) and \(n2\) and all the values between them are \text{'}text1'; the values from \(n3\) to \(n4\) are labeled \text{'}text2'; and so on. If \(n1=n3\), it will be considered in the interval where it is the upper limit.

Assign new names to string values.

To order categories or value labels:

\[
\begin{align*}
\text{ORDER varlist or varlist} & \text{ / MISS,} \\
& \text{DATA or LABEL,} \\
& \text{SORT=} \{\text{option}\}
\end{align*}
\]

where \text{option} is NONE, ASC, DESC, FASC, FDES, numlist, or charvallist. By default, the sort is based on data values unless you specify \text{LABEL}. Specify \text{ORDER \text{var (or \text{var$)}}} to return to the default order.

Use this syntax for recoding numeric values:

\[
\begin{align*}
\text{RECODE varlist} / \ n1= & m1, n2=m2, \\
& \text{or} \\
& n1,n2,\ldots=m1, \\
& n3,n4,\ldots=m2, \\
& \text{or} \\
& \ldots n0=m0 \\
& n1 .. n2=m1, \\
& n3 .. n4=m2,\ldots, \\
& nk .. \ =mk,\ldots,
\end{align*}
\]

The second set of options allows you to put two or more groups into a larger group. The third set allows you to discretize a continuous variable, that is, define intervals along a continuous variable.

For string values, specify:

\[
\begin{align*}
\text{RECODE varlist$} / \ \text{’oldtext1’}= & \text{newtext1,} \\
& \text{’oldtext2’}=\text{newtext2,} \ldots
\end{align*}
\]
To store recoded numeric values into a new numeric variable, issue:

```
RECODE newvar = oldvar / n1=m1, n2=m2,
or
n1, n2,...=m1,
n3, n4,...=m2,
or
.. n0=m0
n1 .. n2=m1,
n3 .. n4=m2,...,
nk .. =mk
```

To store recoded numeric values into a new string variable, issue:

```
RECODE newvar$ = oldvar / n1='text1', n2='text2',...
or
n1, n2,...='text1',
n3, n4,...='text2',...
or
.. n0='text0'
n1 .. n2='text1',
n3 .. n4='text2',...
nk .. = 'textk'
```

To store recoded string values into a new numeric variable, issue:

```
RECODE newvar = oldvar$ / 'text1'=n1,
   'text2'=n2,...
```

To store recoded string values into a new numeric variable, issue:

```
RECODE newvar$ = oldvar$ / 'oldtext1'=newtext1,
   'oldtext2'=newtext2,...
```

In each of the above RECODE statements, value(s) given in the left side of an option are recoded to the value in the right side of the option.

To request separate results for each level of one or more grouping variables, use the BY command before you request a graph or analysis:

```
BY grpvar1, grpvar2, ...
```

For example, if the variable `SEX` has code 1 for males and code 2 for females, and `CITY$` has the codes Chicago, New York, and Los Angeles, specify

```
BY sex, city$
```

to form six groups (Chicago males, Chicago females, New York males, New York females, and so on).
Examples

Example 1
Recoding and Labeling Categories and Intervals

Suppose your SYSTAT file contains these variables with the following data values:

- **MARITAL** 1, 2, 3, 4
- **MARITAL$** Never, Married, Divorced, or Separated
- **EMPLOY** 1, 2, 3, 4, 5, 6
- **EDUCATN** 1, 2, 3, 4, 5, 6, 7
- **AGE** 18, 19, ..., 83
- **SEX$** M, F, and a blank for missing
- **DOSES$** None, Low, Medium, and High

You can use the RECODE command to group cases, and the LABEL command to order the resulting groups, and create labels for the output.

- The numeric codes 1 through 4 stored in the variable **MARITAL** assign each subject to one of four marital status categories. The labels Never through Separated will identify these groups in graphical displays and statistical analyses.

  ```
  LABEL MARITAL / 1='Never', 2='Married', 3='Divorced', 4='Separated'
  ```

- Subjects with code 4 will be used in subsequent displays or analyses that use the variable **MARITAL** but without any label, and the order of the categories in, for example, a bar chart will be Married, Divorced, Never and 4.

  ```
  LABEL MARITAL / 2='Married', 3='Divorced',1='Never'
  ORDER MARITAL / LABEL SORT = {'Married', 'Divorced', 'Never'}
  ```

- **SYSTAT** will use only those subjects with **EMPLOY** codes 1, 2, and 3 when **EMPLOY$** is specified.

  ```
  RECODE EMPLOY$ = EMPLOY / 1='Full time', 2='Part time', 3='Unemployed'
  ```

- The seven codes of **EDUCATN** for educational level are collapsed into four categories of the variable **EDUCATN$** with the assigned names.

  ```
  RECODE EDUCATN$ = EDUCATN / 1,2='HS Dropout', 3='HS grad', 4,5='College', 6,7='Degree +'
  ```
The age of each subject is used to assign the person to one of four age groups. That is, the ages 29, 45, and 60 are used as cutpoints along the continuous variable \( AGE \) to form four intervals. The second specification is a shortcut and works when the intervals are identified in the same order as they fall across the range of the distribution. SYSTAT forms the category as it encounters each specification. Thus, if we specify:

\[
\text{RECODE } \text{AGE$} = \text{AGE} / \\
\quad \ldots 29 = '18 \text{ to } 29', \\
\quad 30 \ldots 45 = '30 \text{ to } 45', \\
\quad 46 \ldots 60 = '46 \text{ to } 60', \\
\quad 60 \ldots = 'Over 60'
\]

or

\[
\text{RECODE } \text{AGE$} = \text{AGE} / \\
\quad \ldots 29 = '18 \text{ to } 29', \\
\quad \ldots 45 = '30 \text{ to } 45', \\
\quad \ldots 60 = '46 \text{ to } 60', 60 \\
\quad \ldots = 'Over 60'
\]

SYSTAT assigns 30-year-olds to the first group and anyone 30.00001 or older to the next group.

\[
\text{RECODE } \text{AGE$} = \text{AGE} / \quad \ldots 30 = 'label1', 30..='label2'
\]

The character codes \( M \) and \( F \) define two subpopulations—they are labeled \( Male \) and \( Female \), respectively.

\[
\text{LABEL } \text{SEX$} / 'M'=\text{Male}, 'F'=\text{Female}
\]

When this \text{LABEL} statement is specified, missing values, corresponding to those subjects who did not report their sex, will be labeled as “Missing” in any output that includes missing values.

\[
\text{LABEL } \text{SEX$} / 'M'=\text{Male}, 'F'=\text{Female}, ' '=' \text{Missing}
\]

For character codes, you may need to eliminate a category and/or specify a specific order. Here, \text{SELECT} is used to delete the subjects who had no drug (\textit{none}) and order the doses from high to low for use in graphs and analyses.

\[
\text{SELECT } \text{DOSE$} <> 'none' \\
\text{ORDER } \text{DOSE$} / \text{SORT} = \{ 'high', 'medium', 'low' \}
\]

\text{RECODE} in some of the above examples and \text{SELECT} both select a subset of cases; however, the subset identified via \text{SELECT} applies to all subsequent displays and analyses; that for \text{RECODE} applies only to those subsequent procedures where the same recoded variables are specified.
Example 2
Labeling Multiple Variables

If you have several variables with the same codes, you can specify grouping information for them in one statement using commands. For example, if your data file has the same doses (6.25, 12.5, and 25) for three drugs (where each drug is a variable), specify labels as follows:

```
LABEL DRUG_A DRUG_B DRUG_C / 6.25='low dose',
       12.5='medium dose',
       25='high dose'
```

Alternatively, if the three drug variables are stored consecutively in the data file, use the ellipsis notation (..):

```
LABEL DRUG_A .. DRUG_C / 6.25='low dose',
       12.5='medium dose',
       25='high dose'
```

Or, as an example using character data, consider a study about smoking cessation in which researchers questioned subjects about their desire to have a cigarette in different situations (after eating, at a party, and after a stressful event). To each question, the subjects responded “always,” “sometimes,” or “never”. For quick data entry, you could name the questions 1 through 10 and enter the responses as a, s, and n. Then specify

```
LABEL Q1$ .. Q10$ / 'a'=always 's'=sometimes 'n'=never
```

to tell SYSTAT to use the full response to label the output.

Example 3
Collapsing Categories

If you assign the same label to two codes (or groups of codes), the codes form a single category. The CANCER data file presents data from a breast cancer study in which 764 women have tumors classified in one of four categories. Two of the categories describe malignant tumors, and the other two, benign tumors. To find out how the overall malignant or benign status relates to survival, the RECODE command is used to collapse the four categories into two:
Chapter 6

The proportion of women with benign tumors who remained alive at the end of the study is significantly greater than the proportion with malignant tumors (p-value = 0.003).

Example 4
Recoding Numeric Variables

Suppose the variable OCCUPATN contains 30 or 40 numeric occupation codes including: 21 (bricklayer), 32 (carpenter), 53 (day laborer), and 65 (plumber). For a particular analysis, you need to study these four jobs and want to recode the values as 1 (day laborer), 2 (carpenter), 3 (plumber), and 4 (bricklayer). Use either the RECODE command as illustrated earlier or the COD function to do this:

```
RECODE NEW_JOB = OCCUPATN / 53=1,32=2,65=3,21=4
```

or

```
LET NEW_JOB = COD(OCCUPATN, 53,32,65,21)
```

The output is:

```
XTAB
USE CANCER
FREQ NUMBER
RECODE TUMOR$ / 'MinMalig''MaxMalig'=Malignant,
'MinBengn' 'MaxBengn'=Benign
PLENGTH NONE / FREQ CHISQ
TABULATE SURVIVE$ * TUMOR$
```

Case frequencies determined by value of variable NUMBER

Counts

SURVIVE$(rows) by TUMOR$(columns)

<table>
<thead>
<tr>
<th></th>
<th>Benign</th>
<th>Malignant</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alive</td>
<td>323</td>
<td>231</td>
<td>554</td>
</tr>
<tr>
<td>Dead</td>
<td>97</td>
<td>113</td>
<td>210</td>
</tr>
<tr>
<td>Total</td>
<td>420</td>
<td>344</td>
<td>764</td>
</tr>
</tbody>
</table>

Chi-square tests of association for SURVIVE$ and TUMOR$

```
Test Statistic | Value | df | p-value
----------------+-------|----|--------
Pearson Chi-square | 9.026 | 1.000 | 0.003
```

Number of Valid Cases: 72

The proportion of women with benign tumors who remained alive at the end of the study is significantly greater than the proportion with malignant tumors (p-value = 0.003).
The values of \textit{OCCUPATN} and the corresponding values of \textit{NEW_JOB} are shown below:

\begin{tabular}{cc}
\textit{OCCUPATN} & \textit{NEW_JOB} \\
53 & 1 \\
32 & 2 \\
65 & 3 \\
21 & 4 \\
\text{All else} & .
\end{tabular}

\textbf{Example 5}  
\textbf{Recoding Characters to Numbers}

The \textit{COD} function also works for string variables. Suppose the string variable \textit{OCCUPATN$} has 30 or 40 string codes. The values of \textit{NEW_JOB} will be 1, 2, 3, and 4 if we specify the following:

\begin{verbatim}
LET NEW_JOB = COD(OCCUPATN$, 'day laborer','carpenter',
                 'plumber','brick layer')
\end{verbatim}

The equivalent \texttt{RECODE} command is:

\begin{verbatim}
RECODE NEW_JOB= OCCUPATN$ / 'day laborer'=1,
                 'carpenter'=2,
                 'plumber'=3,
                 'brick layer'=4
\end{verbatim}

Always remember to use quotation marks to surround all string codes that you specify in the argument of the \textit{COD} function or the option of the \texttt{RECODE} command.

\textbf{Example 6}  
\textbf{Matching Values}

Valid \texttt{INC} statements include:

\begin{verbatim}
LET DRIVER = INC(JOB,63,47,85)
LET DRIVER = INC(JOB$, 'truck driver','bus driver',
                 'cab driver')
\end{verbatim}

In the first statement, if \textit{JOB} contains a 63, 47, or 85, \textit{DRIVER} will contain a 1. \textit{DRIVER} contains a 0 for all other values of \textit{JOB}. In the second statement, if \textit{JOBS$} contains ‘truck driver,’ ‘bus driver,’ or ‘cab driver,’ the new variable \textit{DRIVER} will contain a 1. All other values for \textit{JOBS$} receive a 0 for \textit{DRIVER}. 
Example 7
Categorizing Variables

Sometimes you need to categorize a quantitative variable, such as $AGE$. For example, this statement uses each subject’s age to assign him or her to one of four age groups:

\[
\text{LET AGE_GRP = CUT(AGE, 12, 19, 64)}
\]

The values of $AGE$ and the corresponding values of the new variable $AGE\_GRP$ are shown below:

<table>
<thead>
<tr>
<th>$AGE$</th>
<th>$AGE_GRP$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 &lt; ages &lt;= 12</td>
<td>1</td>
</tr>
<tr>
<td>12 &lt; ages &lt;= 19</td>
<td>2</td>
</tr>
<tr>
<td>19 &lt; ages &lt;= 64</td>
<td>3</td>
</tr>
<tr>
<td>64 &lt; ages</td>
<td>4</td>
</tr>
</tbody>
</table>

The numbers you list in the CUT function (12, 19, and 64) are the upper limits of the intervals. For example, a 64-year-old subject falls into the third category, and someone who reports their age as 64.01 is in the fourth category. Note that $n$ cutpoints define $n + 1$ intervals, so you always have one more interval than the number of cutpoints.

Alternatively, you could use RECODE to do the same task as the above CUT function:

\[
\text{RECODE AGE$ = AGE / .. 12 = 'child',}
\]
\[
\text{.. 19 = 'teen',}
\]
\[
\text{.. 64 = 'adult',}
\]
\[
64 .. = 'senior'
\]

Example 8
Alphabetical Intervals

The CUT function also divides string variables alphabetically into intervals. Suppose the variable $NAME$ contains the following values:

Richards, Carson, Young, Stern, Parker, Buck, Smith, Martin, Howe

The CUT function:

\[
\text{LET GROUP = CUT(NAME$, 'Howe', 'Richards')}
\]
assigns codes alphabetically to the variable GROUP:

<table>
<thead>
<tr>
<th>NAMES</th>
<th>GROUP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Richards</td>
<td>2</td>
</tr>
<tr>
<td>Carson</td>
<td>1</td>
</tr>
<tr>
<td>Young</td>
<td>3</td>
</tr>
<tr>
<td>Stern</td>
<td>3</td>
</tr>
<tr>
<td>Parker</td>
<td>2</td>
</tr>
<tr>
<td>Buck</td>
<td>1</td>
</tr>
<tr>
<td>Smith</td>
<td>3</td>
</tr>
<tr>
<td>Martin</td>
<td>2</td>
</tr>
<tr>
<td>Howe</td>
<td>1</td>
</tr>
</tbody>
</table>

The new variable GROUP contains a 1 for Howe and the names preceding Howe alphabetically, a 2 if the name falls between Howe and Richards (including Richards), and a 3 if it follows Richards.

**Example 9**

**Use of NCAT Function**

This function returns a numeric value which gives the number of categories in a given data variable. Consider the example:

```plaintext
USE AFIFI
TMP x~ = NCAT(DRUG)
PRINT 'The no of categories in drug is', x~
PRINT 'The no of categories in disease is', NCAT(DISEASE)
```

In the output, the number of categories in Drug and Disease are displayed as follows.

The no of categories in Drug is 4.000
The no of categories in Disease is 3.000
The BASIC programming in SYSTAT is a global feature to facilitate transformation and complex data management tasks requiring a programming language. SYSTAT BASIC allows you to:

- read ASCII text files with multiple lines per case, multiple cases per record, records of unequal length or with a fixed format.
- enter data case by case at the command prompt.
- enter or read matrices with or without a diagonal and specify their type as correlation, covariance, similarity, dissimilarity, and so on.
- delete cases that meet or fail to meet a specified condition.
- list data without variable names or case numbers.
- transform variables conditionally or unconditionally.

For more complicated programming tasks, you can:

- use arrays (subscripted variables).
- use FOR... NEXT and WHILE.. ENDWHILE loops.
- include ELSE, PRINT, and DELETE in IF...THEN statements.
- specify commands that operate on subgroups of cases (beginning and end of group and file).
- create your own functions and macros.

This chapter presents applications of SYSTAT transformation commands. While there are simpler ways to accomplish many of these tasks, the programs in this chapter are selected to illustrate the full range of SYSTAT’s capabilities.
Data Entry and BASIC

BASIC in SYSTAT provides more flexibility for reading ASCII data files than the IMPORT command. You can read files with:

- more than one line of data per case.
- fixed-format.
- special data structures, such as multiple cases per line.
- varying numbers of variables per case.

Here is an overview of the commands for reading a text file:

\[
\text{GET inputfilename} \\
\text{INPUT varlist} \\
\text{DSAVE outputfilename}
\]

\textit{inputfilename} is the name of the ASCII file with a .DAT extension to be imported; \textit{varlist} is a list of variable names; and \textit{outputfilename} is the name of the SYSTAT file that is created. SYSTAT uses the number of variable names you list to determine how many values to read for each case. Here, \texttt{INPUT} is a HOT command; it triggers the importing of the data into SYSTAT.

\textbf{Entering data.} Rather than use a GET statement to read an existing file, you can type small data sets in the Interactive tab of the Commandspace as follows:

\[
\text{INPUT varlist} \\
\text{[type data here]}
\]

Type a tilde (~) or ENDINPUT to end data input.

To save the file use the command:

\[
\text{DSAVE output filename}
\]

\textbf{Free- or fixed-format input.} You can use free-format or fixed-format to input data with BASIC commands or to format data in an ASCII text file.
Free-Format Input

Free-format input is easier to use than fixed-format because it reads delimited data.

- Each data value is separated by a tab, a comma, and/or one or more spaces.
- Each new case begins on a new line. A single case can extend over several lines, but the next case must start on a new line. (If you have several cases per record, use a backslash after the varlist.)
- Character values that contain blanks, commas, or special characters are enclosed in single (' ') or double (" ") quotation marks.
- Missing numeric data are recorded as a period (.), and missing character data are recorded as a blank enclosed in single (') or double (" ") quotation marks. They can also be indicated by typing two commas (,,) with no characters between them.

The data file should be ASCII text without nonprinting characters such as page breaks, margin indicators, or control characters. The data can be aligned or unaligned:

<table>
<thead>
<tr>
<th>Sloan</th>
<th>male</th>
<th>31</th>
<th>158.5</th>
<th>Sloan</th>
<th>male</th>
<th>31</th>
<th>158.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>'Mike Johnson'</td>
<td>male</td>
<td>45</td>
<td>165</td>
<td>'Mike Johnson'</td>
<td>male</td>
<td>45</td>
<td>165</td>
</tr>
<tr>
<td>Smythe</td>
<td>female</td>
<td>37</td>
<td>126.5</td>
<td>Smythe</td>
<td>female</td>
<td>37</td>
<td>126.5</td>
</tr>
</tbody>
</table>

SYSTAT counts the number of variable names listed in the INPUT command to determine how many data values to read for each case. After reading the specified number of values, SYSTAT moves to the next line to begin reading the next case.

Use INPUT to name the variables in the order they are read into SYSTAT:

```
INPUT varlist
```

where varlist is a list of variable names. You can identify a range of variables in varlist using subscript notation, such as:

```
INPUT var(1) .. var(10)  
or
INPUT var(1..10)
```

Variable names can be up to 256 characters long. Names of string variables must end with a dollar sign ($). Subscripts can be used for numeric or string variable names (for example, QUESTION(1),...,QUESTION(35)) as long as the total length of the variable name does not exceed 256 characters.
Example 1
Inputting Data at the Command Prompt

You can create a data file by typing data directly in the Interactive tab of the Commandspace. For example, to create a SYSTAT data file called MYFILE, type:

```
INPUT A B C
```

Now enter data, one case to a line, pressing the Enter key at the end of each line:

```
1 2 3
4 5 6
7 8 9
~
```

```
DSAVE myfile
```

The character, a tilde (~), tells SYSTAT to end data input.

Typing data at the command prompt is particularly useful if you are creating a command file and want to include the data in that file rather than in a separate data file.

Example 2
Reading an ASCII Text File: GET

Suppose you want to read an ASCII text file named INFILE.DAT that has the following data:

```
1 2 3
4 5 6
7 8 9
```

However, you then want to save this file as a SYSTAT file named MYFILE1.SYZ that has three variables named A, B, and C. The input is as follows:

```
GET INFILE
INPUT A B C
DSAVE MYFILE1
```
Example 3
Reading Multiple Lines per Case

To demonstrate how to read ASCII files with multiple lines per case, we use the MINIWRLD.DAT file. This file contains 15 variables for each country and has two lines of data per case. The variables are:

- Name of the country.
- Population in 1990.
- Percentage of the population living in cities.
- Number of births per 1000 people per year.
- Number of infants (per 1000 live births) who die during their first year.
- Gross domestic product per capita.
- Average expenditure for education (per person).
- Average expenditure for health (per person).
- Average expenditure for military (per person).
- Type of government.
- Years of life expectancy for males.
- Years of life expectancy for females.
- Percentage of population who can read.
- Type of country.
- Ratio of birth to death rates.
The following illustration shows the file in a text editor.

France  56.358  73  14   6  14524.657  648.069  728.249  432.780  Democracy
73  82  99.00 Europe  1.556
Greece  10.028  65  11  10  5614.184  115.000  158.700  260.400  Democracy
75  80  95.00 Europe  1.222
Switzerland  6.742  58  12  5  17723.499  853.538 1209.077  330.769  Democracy
75  83  99.00 Europe  1.333
Spain  39.269  91  11  6  10153.121 166.675  224.923  117.964  Democracy
75  82  97.00 Europe  1.375
UK     57.366  76  14  7  14259.401 474.488  479.064  447.473  Democracy
73  79  99.00 Europe  1.273
Hungary 10.569  54  12 15  6112.397 234.340  196.792  146.321  OneParty
67  75  99.00 Europe  0.923
Iraq    18.782  68  46  67 1863.509  87.875  19.000  760.000  OneParty
66  68  55.00 Islamic  6.571
Pakistan 114.649  28  43  110  376.801  7.655  6.97  23.268  Military
56  57  26.00 Islamic  3.071
Ethiopia 51.667  14  45  116  127.742  4.943  1.185  10.137  Military
49  52  55.20 Islamic  3.000
Afghanistan 15.862  16  44  154  189.128  1.00  .  .  OneParty
47  46  12.00 Islamic  2.444
Brazil  152.505  68  26  69  2472.049 59.449  22.729  16.260  Democracy
62  68  76.00 'New World'  3.714
Canada  26.538  76  14  7  19353.214 1052.813  936.211  316.797  Democracy
74  81  99.00 'New World'  2.000
Venezuela 19.698  76  28  27  2944.446 193.989  78.652  45.562  Democracy
71  77  85.60 'New World'  7.000
ElSalvador  5.310  39  34  49  1035.808 17.647  7.647  29.020  Military
62  68  65.00 'New World'  4.857

To store these data in the SYSTAT file called MINIWRLD.SYZ, the input is:

```
GET MINIWRLD
INPUT COUNTRY$ POP_1990 URBAN BIRTH_RT BABYMORT,
GDP_CAP EDUC-HEALTH MIL GOV$ LIFEEXPM,
LIFEEXPF LITERACY GROUP$ B_TO_D
DSAVE MINIWRLD
```

**Example 4**

**Reading Records of Unequal Length**

For each new case, SYSTAT reads as many data values as are named in the `INPUT` statement, one value per variable, even if it has to read several lines of data to do so. The following example illustrates what happens when the rows have records that vary in length. The data in the text file `MYDATA1.DAT` are:

<table>
<thead>
<tr>
<th>10</th>
<th>20</th>
<th>30</th>
<th>40</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>60</td>
<td></td>
<td></td>
</tr>
<tr>
<td>70</td>
<td>80</td>
<td></td>
<td></td>
</tr>
<tr>
<td>90</td>
<td>100</td>
<td>110</td>
<td>120</td>
</tr>
</tbody>
</table>
To create a SYSTAT file named *MYFILE2.SYZ*, type:

```
GET MYDATA1
INPUT A B C D
DSAVE MYFILE2
```

To view the contents of the SYSTAT file, type:

```
LIST
```

The output is:

```
Case |    A      |    B      |    C      |    D      |
-----|----------|----------|----------|----------|
 1   | 10.000   | 20.000   | 30.000   | 40.000   |
 2   | 50.000   | 60.000   | 70.000   | 80.000   |
 3   | 90.000   | 100.000  | 110.000  | 120.000  |
```

Line by line, this is how SYSTAT processes the file:

- **Case 1.** SYSTAT reads 10, 20, 30, and 40 as the first case of the new data file.
- **Case 2.** Next, SYSTAT reads two data values from the second line (50 and 60) as variables *a* and *b* for case 2. It still must fill variables *c* and *d*, so it continues to the next line and reads 70 and 80.
- **Case 3.** Finally, SYSTAT reads 90, 100, 110, and 120 as case 3 of the file.

**Evaluating Missing Data**

Missing character data values appear in SYSTAT commands and output as blanks. Missing numeric data values appear as periods (.).

Evaluation of missing numeric data in arithmetic expression:

All numeric arithmetic expressions involving missing values propagate missing values. For example, if you have two variables *X* and *Y* in a SYSTAT file and you sum them as follows:

```
LET SUM = X + Y
```

the value for the new variable SUM is missing for every case where either *X* or *Y* or both are missing.

```
LET K = SUM(X, Y)
```
Chapter 7

The multi-variable functions AVG, SUM, MIN, MAX, and STD automatically exclude missing values from computations.

<table>
<thead>
<tr>
<th>Y</th>
<th>X</th>
<th>SUM</th>
<th>K</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>.</td>
<td>.</td>
<td>2</td>
</tr>
<tr>
<td>.</td>
<td>3</td>
<td>.</td>
<td>3</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
</tbody>
</table>

Evaluation of missing numeric data in relational expression:

- `X < .` is MISSING always
- `X > .` is MISSING always
- `X = .` is TRUE if X is missing and FALSE otherwise (non-missing)
- `X != .` is FALSE if X is missing and TRUE otherwise (non-missing)
- `X <= .` is MISSING always
- `X >= .` is MISSING always

Evaluation of missing numeric data in logical expression:

- `X AND .` is FALSE if X is false and MISSING otherwise
- `X OR .` is TRUE if X is true and MISSING otherwise
- `NOT .` is MISSING

Note: The CALCULATE command not only evaluates arithmetic expressions but also relational and logical expressions. The evaluation of relational and logical expressions in the CALCULATE command gives the above results. When the same evaluation is done in IF and WHILE statements, the resulting missing value is treated as false.

**Example 5**

**Coding Missing Values**

SYSTAT uses a period to represent missing numeric data and a space to represent missing character data.

- Code missing numeric data as periods in your data. Do not code missing numeric data as characters such as `na, m, *, ?` because SYSTAT does not read character data into a numeric field. If missing numeric data are left as blank spaces, SYSTAT moves the next value in the file where the missing value should be. All subsequent data are also moved. (See the following example.)
Code missing character data as a blank space enclosed in single (' ') or double quotation marks (" ").

This example demonstrates what happens when you do not code missing values as periods.

The values in the MYDATA2.DAT text file are:

```
100   200   300
400   600
700   800   900
```

To create a SYSTAT file named NEWFILE.SYZ, use the following input:

```
GET MYDATA2
INPUT A B C
DSAVE NEWFILE
```

The SYSTAT file NEWFILE.SYZ does not contain the correct number of cases or values, as is shown here:

```
Case | A       | B       | C       |
-----|---------|---------|---------|
1    | 100.000 | 200.000 | 300.000 |
2    | 400.000 | 600.000 | 700.000 |
```

Instead of three cases, SYSTAT produces two, omitting 800 and 900. SYSTAT reads the first line of data correctly, but treats the missing value in the second line as a space delimiter separating the values 400 and 600. Therefore, SYSTAT reads 600 as variable \( b \). It completes case 2 by reading 700, the first value of the next line. SYSTAT is ready to start a new case, so it moves to the next line of original data. There are no more lines of data to read, so SYSTAT closes the file.

To read these data correctly, use a period to mark where data are missing. Data in the MYDATA2.DAT file should be as follows:

```
100   200   300
400   .   600
700   800   900
```
Example 6
Reading Multiple Cases per Record

If you want SYSTAT to read more than one case per line, append a backslash (\) to your INPUT statement. The backslash forces SYSTAT to use all the data in the row, even if it has to start a new case. It also forces SYSTAT to start a new case whenever it starts reading a new row of values. Without a backslash, SYSTAT ignores extra values in a row and moves to the next line to find data for the next case.

This example shows how to use the backslash to read data where you have more than one case per line of original data. Data in the text file MYDATA3.DAT are:

<table>
<thead>
<tr>
<th>Tom</th>
<th>23</th>
<th>Jerry</th>
<th>51</th>
<th>Marilyn</th>
<th>50</th>
<th>Lynne</th>
<th>18</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mark</td>
<td>22</td>
<td>Andrew</td>
<td>8</td>
<td>Henry</td>
<td>70</td>
<td>Chris</td>
<td>23</td>
</tr>
</tbody>
</table>

To create the SYSTAT file MULTIPLE.SYZ, type:

```
GET MYDATA3
INPUT NAME$ AGE \n
DSAVE MULTIPLE
```

With the backslash, SYSTAT reads the entire line of original data even though each line fills up four cases in the SYSTAT file. Without the backslash, SYSTAT reads only the first two values from each line and generates only two cases. To view the contents of the SYSTAT file, type:

```
LIST
```

The listing follows:

<table>
<thead>
<tr>
<th>Case</th>
<th>NAME$</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Tom</td>
<td>23.000</td>
</tr>
<tr>
<td>2</td>
<td>Jerry</td>
<td>51.000</td>
</tr>
<tr>
<td>3</td>
<td>Marilyn</td>
<td>50.000</td>
</tr>
<tr>
<td>4</td>
<td>Lynne</td>
<td>18.000</td>
</tr>
<tr>
<td>5</td>
<td>Mark</td>
<td>22.000</td>
</tr>
<tr>
<td>6</td>
<td>Andrew</td>
<td>8.000</td>
</tr>
<tr>
<td>7</td>
<td>Henry</td>
<td>70.000</td>
</tr>
<tr>
<td>8</td>
<td>Chris</td>
<td>23.000</td>
</tr>
</tbody>
</table>
Example 7
Reading Incomplete Records: Backslash(\)

This example shows how to use the backslash to read records that do not have an equal number of values per case. The data in the text file MYDATA4.DAT are:

```
To create the SYSTAT file UNEQUAL.SYZ, the input follows:

GET MYDATA4
INPUT A B C D \ 
DSAVE UNEQUAL
```

The resulting SYSTAT data file looks like this:

<table>
<thead>
<tr>
<th>Case</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.000</td>
<td>2.000</td>
<td>3.000</td>
<td>.</td>
</tr>
<tr>
<td>2</td>
<td>4.000</td>
<td>5.000</td>
<td>6.000</td>
<td>7.000</td>
</tr>
<tr>
<td>3</td>
<td>8.000</td>
<td>9.000</td>
<td>.</td>
<td></td>
</tr>
</tbody>
</table>

Fixed-Format Input

With fixed-format input, you tell SYSTAT exactly where the values for each variable are located in the data records. Values for a variable must be in the same place for every record.

A fixed-format input statement consists of two parts. The first part provides the variable names that will appear in the SYSTAT file. The second part contains the format that tells SYSTAT where to read values for each variable. The number of items specified in the format must match the number of input variables. Enclose the variable names and the input format in separate sets of parentheses:

```
INPUT (age,sex$,income) (#3,$6,#8)
```

SYSTAT checks the number of items you specify in the format against the number of variables. If they do not match, an error message is displayed.
Formats

Formats control a pointer that tells SYSTAT where to read the next variable value. These formats specify the location and width of fields that contain values. Leading and trailing blanks are ignored for numeric data. All characters within the formatted field, except leading blanks, are read into a character value. Character strings are left-justified.

Format items for fixed-format input include the following:

- `#n` Read a numeric value in the next `n` columns.
- `$n` Read a character value in the next `n` columns.
- `>` Move the pointer one column to the right.
- `<` Move the pointer one column to the left.
- `^n` Move the pointer to column `n`.
- `/` Move the pointer to the first column on the next record.
- `%%n` Move the pointer to the first column on the `n`th record.
- `n*r` Repeat `r n` times, where `r` is any of the above.

Some examples follow:

- `>>>` Move the pointer three columns to the right.
- `3*>` Move the pointer three columns to the right.
- `^10` Move to column 10 of the current record.
- `#4` Read the numeric value in the next four columns beginning at the current column.
- `$5` Read the character value in the next five columns beginning at the current column.
- `^3` Move the pointer to column 3.
- `>>>>>` Move the pointer five columns to the right.
- `5*>` Move the pointer five columns to the right.
- `%2` Move the pointer to column 1 of the second record. (You cannot move back to an earlier record.)
- `/` Move the pointer to column 1 of the next record.
- `//` Move the pointer to column 1 two records ahead. (For example, if you are starting on the first record, `%3` and `//` mean the same thing.)
- `#3` Read the numeric value in the next three columns beginning at the current column.
- `2*$3` Read one character value in three columns and then another in the next three columns.

You can use a backslash (\) to keep the pointer at the current position—rather than move to the next row—to begin the next case. This feature allows you to read files with multiple cases per record.
The following points are to be noted when using the backslash (\):

- The backslash (\) cannot be used with the frontslash (/) in the same INPUT formatted statement since, together, they don't give any meaningful structure.
- The backslash (\) cannot be used with the % option in the same INPUT formatted statement since, together, they don't give any meaningful structure.

Note: Use % and ^ rather than / and >, so that you know precisely which record and column you are on. Furthermore, if you have seven records per observation, you need a %7 at the end of your format to ensure that the pointer is positioned correctly for the next observation, even if you read nothing from the seventh field.

Example 8
Fixed Format

Suppose that you have an ASCII file TESTDATA.DAT like the one below:

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>012345678901234567890123456</td>
<td>1232 BILLY 0 1 1 0 BACDD BCEAD</td>
<td></td>
</tr>
<tr>
<td></td>
<td>7384 SUSAN 1 1 0 1 1 BDAAE DDEAE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2837 TIM 1 1 1 0 1 CBDAE DDBCA</td>
<td></td>
</tr>
<tr>
<td></td>
<td>7484 TOM 0 0 1 0 1 BCDEC AEDDC</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5678 WAYNE 1 1 0 0 0 ADEAA DACBB</td>
<td></td>
</tr>
</tbody>
</table>

The two lines at the top of the example help you count columns, while the first variable in the file is a four-column ID number. The second is the first name of a student. The next five variables are answers to true–false questions and are separated by spaces. The last five variables on the first line are answers to multiple-choice questions and are not delimited. The variables on the second line are answers to five more multiple-choice questions.

SYSTAT reads the data into a SYSTAT file called TEST.SYZ. Because the INPUT statement takes up more than one line, a comma is used to continue it onto the next line:

```
GET TESTDATA
INPUT (ID, NAME$, Q(1) .. Q(5), Q$(6) .. Q$(15)),
(#4,$6,5*$2,>,5*$1,%2,5*$1)
DSAVE TEST
```
Here is how each variable is read by its format description:

- **#4** Read numeric value from first four columns into variable ID.
- **$6** Read character value from next six columns into NAMES.
- **5*#2** Read five consecutive two-column numeric values into \( Q(1), Q(2), Q(3), Q(4), \) and \( Q(5) \).
- **>** Move pointer one space to the right.
- **5*$1** Read five consecutive one-column character values into \( QS(6), QS(7), QS(8), QS(9), \) and \( QS(10) \).
- **%2** Move pointer to second line of input record.
- **5*$1** Read five consecutive one-column character values into \( QS(11), QS(12), QS(13), QS(14), \) and \( QS(15) \).

**Example 9**

**Inputting Fixed Format Data with Backslash ( \ )**

This example tells us how to read files with multiple cases per record using backslash with a specified format. For example, consider the following input:

```
NEW
INPUT ( A B C \ ) (#4, #3, #2)
123456789642
9753146281
~
```

Then it will read the data as follows:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>567</td>
<td>89</td>
</tr>
<tr>
<td>642</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>9753</td>
<td>146</td>
<td>28</td>
</tr>
<tr>
<td>1</td>
<td>.</td>
<td>.</td>
</tr>
</tbody>
</table>

**Entering Triangular Matrices**

When you input data, it takes the cases-by-variables (rectangular) form by default. SYSTAT also enables you to enter triangular matrices, such as those produced by CORR.

Use the TYPE command to indicate what type of matrix you are entering. Valid types are RECTANGULAR, SSCP, COVARIANCE, CORR, DISSIMILARITY, and SIMILARITY. The default is RECTANGULAR.
COVARIANCE designates a covariance matrix and CORR, a correlation matrix. When CORR outputs a triangular matrix to a SYSTAT file, it automatically sets the type. If you LIST a triangular matrix, the upper triangular portion contains missing values, since the matrices are symmetric and only half the values are needed by the statistical routines.

For some types of data, the values on the diagonal are undefined or constant. In these cases, you can input only the values below the diagonal and leave the diagonal missing.

Use DIAGONAL ABSENT to signal that the diagonal values are missing. If you do not use this command, DIAGONAL PRESENT is assumed. If you input a correlation matrix with DIAGONAL ABSENT, SYSTAT sets the diagonal elements to 1.0; otherwise, the diagonal elements are set to missing values.

The TYPE and DIAGONAL commands are cold commands and hence should be given before INPUT if you enter data using INPUT and ENDINPUT commands. If you read data from an ASCII file, write them in between GET and INPUT commands.

Example 10

Covariance Matrices

This example reads a covariance matrix from an ASCII file named MYDATA5.DAT and saves it in a SYSTAT file named TURTLE.SYZ. MYDATA5 contains the following data records:

451.39
271.17 171.73
168.70 103.29 66.65

To create the SYSTAT file, the input is:

GET MYDATA5
TYPE COVARIANCE
INPUT LENGTH, WIDTH, HEIGHT
DSAVE TURTLE
Example 11
Matrices with Missing Diagonals: DIAGONAL

The following example reads a correlation matrix with no diagonal elements. The matrix is a text file named MYCORR.DAT that looks like this:

```
.96
.90  .91
.72  .11  .99
.09  .42  .89  .98
.59  .51  .73  .90  .93
.88  .79  .52  .82  .89  .97
```

To store this matrix as the SYSTAT file COLORS.SYZ, type:

```
GET MYCORR
TYPE CORR
DIAGONAL ABSENT
INPUT RED, ORANGE, YELLOW, GREEN, BLUE, INDIGO, VIOLET
DSAVE COLORS
```

Notice that there are only six rows and columns to fill seven variables. The diagonal elements are set to 1.0.

Troubleshooting ASCII Files

You must be sure that your ASCII file does not contain any nonprinting ASCII characters, such as page breaks, control characters, and column markers. SYSTAT can read numbers, alphabetical and keyboard characters, delimiters (spaces, commas, or tabs that separate consecutive values from each other), and carriage returns.

Numeric fields must contain only numeric data; therefore, exclude variable labels or column headings when using GET to read from an ASCII file. Note that SYSTAT can import ASCII data files containing variable names with Open on the File menu.

Most text editors, such as Microsoft Word and WordPerfect, can save files in an ASCII text format without nonprinting characters. If you do not know whether your file is in an ASCII format, use the TYPE command in DOS to view it. If you see any strange characters, your file was not saved in ASCII format, and SYSTAT will not be able to read it.
Errors and Error Messages

The following are some of the error messages encountered when reading ASCII files:

Empty file error. Make sure that you spelled the filename correctly and that it is in the current directory. If it is not, either copy it to the current directory or specify the complete filename (path plus full filename).

Long INPUT statement. If your INPUT statement is too long for one line, type a comma at the end of the first line and press Enter. Continue typing the statement on the next line. Do this for as many lines as you need.

Data lost or in the wrong columns. If SYSTAT places data incorrectly or data are lost when you read them, make sure that you correctly specify missing values in your data file. If you are using free-format input, enter missing numeric values as periods (.) and missing character values as blanks enclosed in quotation marks (" "). If you are using fixed-format input, you can leave missing values as blank spaces. If you are using free-format input to read a file that does not have the same number of values in every record, add the backslash (\) at the end of your INPUT statement.

Unexpected data error. Make sure that the ASCII file includes no field headings or variable labels, and make sure that variable types match data types. Do not put character data under a numeric variable or vice versa. Make sure that you correctly specify missing values. If you are using free-format input, enter missing numeric values as periods (.) and missing character values as blanks enclosed in quotation marks (" "). If you are using free-format input to read a file that does not have the same number of values in every record, add the backslash (\) to the end of your INPUT statement. If you are using fixed-format input, make sure that you specify the variable types in the format section correctly.

Non-ASCII character warning. Check for nonprinting characters in your file. Such characters include control characters, tab markers, margin and page-break indicators, and so on.

Nonmatching number of variables error. The number of variables defined in the format of your INPUT statement does not match the number of variables named in the variable list.

Input past end of record error. The format of your INPUT statement tells SYSTAT to read your ASCII file records further than allowed. This message should rarely occur.
Variable Types

SYSTAT handles two types of variables: file (data) variable and temporary variable.

**File (Data) Variables.** File variables appear in data file and can hold multiple values. Values are assigned to file variables through the LET command.

\[
\text{LET var} = \text{value}
\]

**Temporary Variables.** The variables created in memory temporarily for doing data-related operations are called temporary variables. A temporary variable always holds a single value. The TMP command enables you to assign a value to a temporary variable.

\[
\text{TMP tempvar~} = \text{value} \\
\text{TMP tempvar$~} = \text{string}
\]

Temporary variable names should compulsorily end with a tilde (~) sign. Assigning a temporary variable to a file variable is possible, but the reverse is not possible; you will be able to assign specific values of a file variable with the help of DATA or DATA$ functions. Once created, temporary variables are available for further processing till you clear them from memory using the CLEAR VARIABLES command or till you close the session.

\[
\text{CLEAR VARIABLES} = \text{tempvarlist}
\]

**Example 12**

**Creating Temporary Variables**

The following example includes the creation and manipulation of temporary variables.

```
NEW
REPEAT 1
TMP X~ = 5
TMP Y~ = 2
LET Z = X~ + Y~
TMP W~ = DATA (Z, 1)
TMP A$~ = "DATA"
TMP B$~ = "MANAGEMENT"
LET C$ = CAT$(A$~, B$~)
PRINT A$~, B$~
LIST Z, C$
CLEAR VARIABLES = X~, Y~, W~, A$~, B$~
```
The output is:

```
DATA MANAGEMENT
Case   Z   C$
-------+-------------------------
 1   | 7.000   DATA MANAGEMENT
```

**Data Output**

**Export Data to an ASCII Text File**

You may need to write data in SYSTAT files as text when you want to use them in a word processor or in another program. You can use the PUT command, the PRINT command, or the LIST command to store data in a text file.

If you want a command conversion procedure, use PUT:

```
USE filename
PUT outputfilename
```

where USE identifies the SYSTAT input file and PUT assigns a filename for the output ASCII file. PUT saves your data in a text file using 12 columns for each value, with values separated by commas. Character values (strings) are enclosed in double quotation marks (" "). When there are many variables, each case wraps such that no line of the ASCII file exceeds 132 columns in length. Use FORMAT n to control the number of decimal places appearing for numeric values. Variable names are not included in the ASCII file. SYSTAT automatically adds a .DAT.

**Example 13**

**Converting a Data File to a Text File**

The following commands convert the SYSTAT file MINIWRLD to a text file:

```
USE MINIWRLD
PUT TEXTFILE
```
You can use a word processor to view TEXTFILE.DAT. The first case in the file appears as follows:

"France", 56.358, 73.000, 14.000, 6.000, 14542.657, 648.069, 728.249, 432.780, "Democracy", 73.000, 82.000, 99.000, "Europe", 1.556

While a hard return usually appears at the end of the text, the text for the first line "wraps around" here for display purposes. Notice that PUT encloses character values in quotation marks. This does not happen if you use PRINT.

**Printing Data and Text**

You can print data and text using the PRINT command. PRINT prints numeric and string constants, values of file and temporary variables, and values of numeric expressions. The comma separator is mandatory.

```
PRINT number text filevarlist tmpvarlist expression
```

The PRINT statement is executed for each case if it contains a file variable.

**Example 14**

**Printing File and Temporary Variables**

In the following example, the first PRINT statement is executed only once without depending on the data file, whereas the second PRINT statement is executed for each case since it contains a file variable RED.

```
USE COLOR
TMP u~ = 10
PRINT "The value of u~ is ", u~
PRINT RED, u~
```

The output is:

The value of u~ is 10.000

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.470</td>
<td>10.000</td>
<td></td>
</tr>
<tr>
<td>0.700</td>
<td>10.000</td>
<td></td>
</tr>
<tr>
<td>0.400</td>
<td>10.000</td>
<td></td>
</tr>
<tr>
<td>0.330</td>
<td>10.000</td>
<td></td>
</tr>
<tr>
<td>0.490</td>
<td>10.000</td>
<td></td>
</tr>
</tbody>
</table>
Example 15

Saving Data as Text without Quotes: PRINT

If you do not want commas as delimiters and quotation marks around character data, use the PRINT command to send the output to a file.

```
USE MINIWRLD
IDVAR COUNTRY$
PUSH CLASSIC
CLASSIC ON
OUTPUT SUBSET2
PRINT COUNTRY$, LITERACY, URBAN, BIRTH_RT, GOV$
OUTPUT *
POP CLASSIC
```

The OUTPUT * command closes the text file. The resulting file, SUBSET2.DAT, looks like this:

```
France        99.000   73.000   14.000   Democracy
Greece        95.000   65.000   11.000   Democracy
Switzerland   99.000   58.000   12.000   Democracy
Spain         97.000   91.000   11.000   Democracy
UK            99.000   76.000   14.000   Democracy
Hungary       99.000   54.000   12.000   OneParty
Iraq          55.000   68.000   46.000   OneParty
Pakistan      26.000   28.000   43.000   Military
Ethiopia      55.200   14.000   45.000   Military
Afghanistan   12.000   16.000   44.000   OneParty
Brazil        76.000   68.000   26.000   Democracy
Canada        99.000   76.000   14.000   Democracy
Venezuela     85.600   76.000   28.000   Democracy
ElSalvador    65.000   39.000   34.000   Military
```

Conditional and Looping Statements

IF...THEN Statement

With the IF...THEN statement, you can execute actions conditionally.

```
IF condition THEN action
```

The action that follows THEN can be any legal statement including LET, TMP, PRINT, DELETE, and IF...THEN. For example, legal IF...THEN statements are:

```
IF condition THEN LET variable = expression
IF condition THEN TEMP variable = expression
IF condition THEN PRINT expression(s)
IF condition THEN DELETE
IF condition THEN IF condition THEN...
```
If *condition* involves file variables, or action is LET, PRINT with file variable or DELETE, then SYSTAT executes the action one case at a time.

Any IF statement can also be written in multiple lines provided the resultant action is closed with an ENDIF command:

```
IF condition THEN
  action
END IF
```

In general, *action* may be a single statement including LET, TMP, PRINT, and IF...THEN, or a block of statements enclosed between FOR and NEXT. If *condition* does not involve file variables, then *action* may even be any of the data, graph or module related commands like SORT, RANK, BAR, CSTAT, TABULATE, etc. For module related commands, the module should be loaded before the IF statement can be executed.

**ELSE Statement**

An IF...THEN statement only checks for one condition. You may want, however, to execute many conditional statements at once. If you are testing consecutive related conditions on the same variable, SYSTAT provides an ELSE statement to accompany IF...THEN. In its simplest form, IF...THEN... ELSE take the format:

```
IF condition THEN action1 ELSE action2
```

where *action1* and *action2* could be one of LET, TEMP, PRINT, DELETE and IF...THEN. SYSTAT executes *action2* only when the preceding IF condition is false. Another IF...THEN statement can follow ELSE, enabling you to string together a number of related conditional statements:

```
IF condition1 THEN action1,
ELSE IF condition2 THEN action2,
ELSE IF condition3 THEN action3,
ELSE action
```

In this case, SYSTAT executes the statement following ELSE only when all preceding IF conditions are false. When a preceding condition is true, SYSTAT ignores subsequent ELSE statements.
You can also use a multiline IF...THEN ... ELSE statement:

```
IF condition THEN
  action1
ELSE
  action2
ENDIF
```

Here, action1 and action2 are similar to action in the multiline IF statement explained above. If action2 is a single BASIC command then it may immediately follow ELSE on the same line.

**Example 16**  
**IF...THEN...ELSE Statement**

The following examples compare two sets of transformation commands. The first uses only IF...THEN statements to assign values to a new variable called RATE$ based on values for CARDIO:

```
USE USSTATES
  IF CARDIO < 275 THEN LET RATE$ = 'LOW'
  IF CARDIO >= 275 AND CARDIO < 325,
     THEN LET RATE$ = 'AVERAGE'
  IF CARDIO >= 325 THEN LET RATE$ = 'HIGH'
  IF CARDIO = . THEN LET RATE$ = 'MISSING'
DSAVE NEWDATA
```

Using IF...THEN and ELSE makes the commands simpler:

```
USE USSTATES
  IF CARDIO < 275 THEN LET RATE$ = 'LOW',
  ELSE IF CARDIO < 325 THEN LET RATE$ = 'AVERAGE',
  ELSE IF CARDIO >= 325 THEN LET RATE$ = 'HIGH',
  ELSE LET RATE$ = 'MISSING'
ENDIF
DSAVE NEWDATA
```

SYSTAT executes these commands once for each case. The order of the IF and ELSE statements is important. The ELSE depends on the truth of the IF conditions before it. SYSTAT executes an ELSE statement only if all preceding conditions are false.

After running these commands, check the values for the first 10 cases for CARDIO and RATE$:

```
USE NEWDATA
LIST STATE$, CARDIO, RATE$ / N=10
```
Chapter 7

The output follows:

<table>
<thead>
<tr>
<th>Case</th>
<th>STATES</th>
<th>CARDIO</th>
<th>RATE$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ME</td>
<td>330.400</td>
<td>HIGH</td>
</tr>
<tr>
<td>2</td>
<td>NH</td>
<td>278.300</td>
<td>AVERAGE</td>
</tr>
<tr>
<td>3</td>
<td>VT</td>
<td>289.400</td>
<td>AVERAGE</td>
</tr>
<tr>
<td>4</td>
<td>MA</td>
<td>343.900</td>
<td>HIGH</td>
</tr>
<tr>
<td>5</td>
<td>RI</td>
<td>353.600</td>
<td>HIGH</td>
</tr>
<tr>
<td>6</td>
<td>CT</td>
<td>318.800</td>
<td>AVERAGE</td>
</tr>
<tr>
<td>7</td>
<td>NY</td>
<td>376.200</td>
<td>HIGH</td>
</tr>
<tr>
<td>8</td>
<td>NJ</td>
<td>356.100</td>
<td>HIGH</td>
</tr>
<tr>
<td>9</td>
<td>PA</td>
<td>400.600</td>
<td>HIGH</td>
</tr>
<tr>
<td>10</td>
<td>OH</td>
<td>344.600</td>
<td>HIGH</td>
</tr>
</tbody>
</table>

Example 17
Multiline IF...THEN Statement

The mean value for the variable PULMONAR from the data set USSTATES is approximately 34.8 with a standard deviation of 7.7. Suppose you want to set RATE$ to HIGH and RATE to 1 if PULMONAR is more than one standard deviation above average.

USE USSTATES
IF PULMONAR > 34.8 THEN FOR
LET RATE$ = 'HIGH'
LET RATE = 1
NEXT
ENDIF
LIST PULMONAR, RATE$, RATE

Here are the first 10 cases:

<table>
<thead>
<tr>
<th>Case</th>
<th>PULMONAR</th>
<th>RATE$</th>
<th>RATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>46.100</td>
<td>HIGH</td>
<td>1.000</td>
</tr>
<tr>
<td>2</td>
<td>33.500</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>45.200</td>
<td>HIGH</td>
<td>1.000</td>
</tr>
<tr>
<td>4</td>
<td>34.400</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>34.300</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>29.300</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>31.000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>29.300</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>36.000</td>
<td>HIGH</td>
<td>1.000</td>
</tr>
<tr>
<td>10</td>
<td>37.500</td>
<td>HIGH</td>
<td>1.000</td>
</tr>
</tbody>
</table>

For the cases that meet the condition PULMONAR > 34.8, SYSTAT executes the two transformations between FOR and NEXT. For those cases that do not meet the condition, SYSTAT assigns missing values to RATE$ and RATE. Missing values for string variables appear as blanks.
Example 18
Multiline IF...THEN...ELSE Statement

This example shows the IF...THEN...ELSE format with the FOR...NEXT statement and the following assignments:

Where PULMONAR is | Let RATE$ = | Let RATE =
---|---|---
< 20.8 | LOW | 1
\geq 20.8 \text{ and }< 34.8 | MID | 2
\geq 34.8 | HIGH | 3

The input is:

```
USE USSTATES
IF PULMONAR < 20.8 THEN FOR
   LET RATE$ = 'LOW'
   LET RATE = 1
NEXT
ELSE IF PULMONAR < 34.8 THEN FOR
   LET RATE$ = 'MID'
   LET RATE = 2
NEXT
ELSE IF PULMONAR \geq 34.8 THEN FOR
   LET RATE$ = 'HIGH'
   LET RATE = 3
NEXT
ENDIF
LIST PULMONAR, RATE$, RATE
```

Here are the first 10 cases:

<table>
<thead>
<tr>
<th>Case</th>
<th>PULMONAR</th>
<th>RATE$</th>
<th>RATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>46.100</td>
<td>HIGH</td>
<td>3.000</td>
</tr>
<tr>
<td>2</td>
<td>33.500</td>
<td>MID</td>
<td>2.000</td>
</tr>
<tr>
<td>3</td>
<td>45.200</td>
<td>HIGH</td>
<td>3.000</td>
</tr>
<tr>
<td>4</td>
<td>34.400</td>
<td>MID</td>
<td>2.000</td>
</tr>
<tr>
<td>5</td>
<td>34.300</td>
<td>MID</td>
<td>2.000</td>
</tr>
<tr>
<td>6</td>
<td>29.300</td>
<td>MID</td>
<td>2.000</td>
</tr>
<tr>
<td>7</td>
<td>31.000</td>
<td>MID</td>
<td>2.000</td>
</tr>
<tr>
<td>8</td>
<td>29.300</td>
<td>MID</td>
<td>2.000</td>
</tr>
<tr>
<td>9</td>
<td>36.000</td>
<td>HIGH</td>
<td>3.000</td>
</tr>
<tr>
<td>10</td>
<td>37.500</td>
<td>HIGH</td>
<td>3.000</td>
</tr>
</tbody>
</table>

If, for a case, \textit{PULMONAR} is less than 20.8, SYSTAT executes the associated FOR...NEXT statements, setting the values of \textit{RATE$} to \textit{LOW} and \textit{RATE} to 1. It does not execute the subsequent ELSE statements but moves on to the next case.
If $PULMONAR$ is greater than or equal to 20.8 but less than 34.8, SYSTAT executes the first ELSE statement. SYSTAT sets $RATES$ to $MID$ and $RATE$ to 2 and does not execute the second ELSE statement.

If $PULMONAR$ is greater than 34.8, SYSTAT executes the last ELSE statement and sets $RATES$ to $HIGH$ and $RATE$ to 3.

**FOR...NEXT Statement**

The FOR...NEXT statement is used to execute a block of statements a specified number of times.

The syntax for a FOR...NEXT statement is:

```
FOR index = n1 TO n2 STEP n3
  statement1
  statement2
  .
  .
  .
NEXT
```

The index is a temporary numeric variable, $n1$ and $n2$ are the initial and final values of the index, and $n3$ is the amount the index is changed each time through the loop. $n1$, $n2$ and $n3$ can be numerical expressions. STEP is optional; by default, $n3$ takes the value 1 when $n1 < n2$ and -1 when $n1 > n2$.

Adding tilde(~) at the end of index is not mandatory in FOR...NEXT loop.

When execution of the FOR...NEXT loop begins, SYSTAT evaluates $n1$, $n2$, and $n3$. It then assigns $n1$ to index. Before it runs the statement block, it compares index to $n2$. If index is already past the $n2$ value, the FOR...NEXT loop terminates, otherwise the statement block runs. The index can be changed from within the loop, so can $n1$, $n2$ and $n3$. While changes to index affect the number of times the loop is repeated, changes to $n1$, $n2$ and $n3$ do not.
The following are some illustrations:

```
TMP S~ = 100
FOR K = 1 TO 5
  PRINT S~   !! S~ is a temporary variable
NEXT
USE RAINFALL
FOR I = 1 TO 3
  PRINT X    !! X is a file variable
NEXT
```

If any statement contains a file variable, then the FOR...NEXT loop is executed for each case. In the first illustration, since the PRINT statement does not contain any file variable, SYSTAT simply prints the S~ value 5 times. In the second illustration, the PRINT statement is executed 3 times for each case.

The STEP option adjusts the size of the increment. For the following script, SYSTAT increments the index i by two each time.

```
FOR i = 1 TO 10 STEP 2
  PRINT i
NEXT
```

The values of i are printed as 1, 3, 5, 7, and 9 consecutively.

FOR...NEXT loops without an index are executed once.

```
FOR _statement(s)_
NEXT
```

**Nesting.** You can nest a FOR...NEXT loop within another. However, each loop must have a unique index.

```
FOR index1 = ...
  FOR index2 = ...
    FOR index3 = ...
      ...
    NEXT
  NEXT
NEXT
```

Always match every FOR with a NEXT.
Example 19

Unpacking Records

This example shows how to reorganize several repeated measures on a single record as one measure per record and how to generate a sequence number for each value.

Begin with the TRIAL.SYZ data file that has two records:

10 20 30 40 50 Male
11 21 31 41 51 Female

The following examples generate a file with 10 cases. First, you create a temporary ASCII text file with 10 cases, each containing one data value plus its sequence number and label:

SYSTAT’s FOR... NEXT statement lists the five data values, and the values of SEX$ and I (the index) as sequence numbers. The OUTPUT * command closes the text file. You read the data from the text file TEMP into the SYSTAT file NEWTRIAL as follows:

The following are the values in the SYSTAT file NEWTRIAL:

<table>
<thead>
<tr>
<th>X</th>
<th>I</th>
<th>SEX</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10.000</td>
<td>1.000</td>
</tr>
<tr>
<td>2</td>
<td>20.000</td>
<td>2.000</td>
</tr>
<tr>
<td>3</td>
<td>30.000</td>
<td>3.000</td>
</tr>
<tr>
<td>4</td>
<td>40.000</td>
<td>4.000</td>
</tr>
<tr>
<td>5</td>
<td>50.000</td>
<td>5.000</td>
</tr>
<tr>
<td>6</td>
<td>11.000</td>
<td>1.000</td>
</tr>
<tr>
<td>7</td>
<td>21.000</td>
<td>2.000</td>
</tr>
<tr>
<td>8</td>
<td>31.000</td>
<td>3.000</td>
</tr>
<tr>
<td>9</td>
<td>41.000</td>
<td>4.000</td>
</tr>
<tr>
<td>10</td>
<td>51.000</td>
<td>5.000</td>
</tr>
</tbody>
</table>
**Example 20**

**Random Walks**

A random walk is a two-dimensional trace of a path in which the direction taken at each step is given by a random variable. The following one is a program for generating a random walk in two dimensions.

```
NEW
REPEAT 1000
RSEED 123
LET R=URN()
TMP X~=-0
TMP Y~=-0
FOR
IF R<.25 THEN TMP X~=-X~+1,
ELSE IF R<.5 THEN TMP Y~=-Y~+1,
ELSE IF R<.75 THEN TMP X~=-X~+1,
ELSE TMP Y~=-Y~+1
LET X =X~
LET Y =Y~
NEXT
PLOT X * Y / LINE, AXES=None, SCALE=None, SIZE=0
DELETE COLUMNS = R
CLEAR VARIABLES = X~, Y~
```

The output is:
In the plot, we omit the axes to emphasize the shape itself. The elongated (non-globular) shape is typical of random walks and is somewhat counter-intuitive. Rudnick and Gaspari (1987) discuss the shapes of random walks and show several examples that look like SYSTAT's display.

**WHILE...ENDWHILE Statement**

A WHILE...ENDWHILE statement is used to repeatedly execute a block of statements as long as a specified condition is true.

The syntax for a WHILE ...ENDWHILE statement is:

```
WHILE (condition)
  statements
ENDWHILE
```

Here, condition can only involve temporary variables. If condition is true, all of the statements run until ENDWHILE is encountered. Control then returns to WHILE and condition is again checked. If condition is still true, the process is repeated. The loop is executed for each case if any statement contains a file variable.

Use a WHILE...ENDWHILE statement when you want to repeat a set of statements an indefinite number of times, as long as a condition remains true. If you want to repeat the statements a fixed number of times, the FOR...NEXT statement is usually a better choice.

**Conditional WHILE...ENDWHILE Statements**

WHILE...ENDWHILE can be used with multiline IF...THEN...ELSE statements:

```
IF condition THEN
  WHILE
    ...
  ENDWHILE
ELSE
  WHILE
    ...
  ENDWHILE
ENDIF
```
**Example 21**

**Using WHILE...ENDWHILE Loop**

The following example prints the values of two temporary variables as long as their difference is greater than 0.3.

```plaintext
RSEED 879
TMP OldValue~ = 0
TMP NewValue~ = 1
WHILE(ABS(NewValue~ - OldValue~) > 0.3)
    TMP oldValue~ = NewValue~
    TMP NewValue~ = URN(0,1)
    PRINT "Old Value = ", OldValue~, "New Value = ", NewValue~
ENDWHILE
```

The output is:

```
Old Value =  1.000 New Value =  0.036
Old Value =  0.036 New Value =  0.899
Old Value =  0.899 New Value =  0.321
Old Value =  0.321 New Value =  0.365
```

**Subscripted Variables**

A subscripted variable in SYSTAT looks like this:

```
varname(subscript)
```

where *subscript* should be an integer. Subscripted variables are typically created through the `DIM` command.

**Dimensioning Space for New Variables - The DIM command**

To add new subscripted variables to a file, use the `DIM` command.

```
DIM varname(size)
```

A `DIM` statement reserves space for new subscripted variables. You may also define string subscripted variables by suffixing `varname` with the `$` sign. For example, the following `DIM` statement creates new variables `X(1), X(2), X(3), X(4),` and `X(5).

```
DIM X(5)
```
Similarly,

```plaintext
DIM X$(3)
```

creates three string subscripted variables X$(1), X$(2) and X$(3).

You can use a FOR...NEXT loop to access a set of subscripted variables.

**FOR...NEXT Loops with Subscripted Variables**

Suppose you have a file containing the variables X(1...10) and you want to calculate the natural log of each. You could either enter 10 separate LET commands or use the FOR...NEXT looping construct to do this. The input is as follows:

```plaintext
FOR n = 1 TO 10
    LET x(n) = LOG(x(n))
NEXT
```

SYSTAT runs through the loop 10 times, increasing the value of n by one each time. Thus, the value of n increases successively: 1, 2, 3, 4, 5, 6, 7, 8, 9, and 10. This FOR...NEXT statement is the same as:

```plaintext
LET X(1) = LOG(X(1))
LET X(2) = LOG(X(2))
LET X(3) = LOG(X(3))
LET X(4) = LOG(X(4))
LET X(5) = LOG(X(5))
LET X(6) = LOG(X(6))
LET X(7) = LOG(X(7))
LET X(8) = LOG(X(8))
LET X(9) = LOG(X(9))
LET X(10) = LOG(X(10))
```

Alternatively, you could use the @ shortcut and specify:

```plaintext
LET (X(1 .. 10)) = LOG(@)
```

If you do not want to change the values in x by replacing them with their logs, you can assign the transformed values to a new variable Y.

The following program creates new variables Y(1...10) whose values are the natural logarithms of the corresponding values in X(1...10).

```plaintext
DIM Y(10)
FOR n = 1 TO 10
    LET Y(n) = LOG(X(n))
NEXT
```
Without the DIM statement, SYSTAT would not understand the \( y \) variable subscript in the LET statement and would respond with an error message. You cannot redimension existing or previously defined arrays of subscripted variables using DIM. To do this, use DEFVAR.

\[
\text{DEFVAR } A(n1..n2)
\]

where \( n1 \) and \( n2 \) are positive integers. This creates numeric subscripted variables \( A(n1) \) to \( A(n2) \). String subscripted variables can be similarly created.

The following script creates an array of 10 subscripted variables:

\[
\begin{align*}
\text{DIM } & \text{ X(4)} \\
\text{DEFVAR } & \text{ X(5..10)}
\end{align*}
\]

Here, DEFVAR has augmented the original set of 4 subscripted variables with 6 more.

An array of subscripted variables can be deleted at once using the DELELE COLUMNS command.

\[
\text{DELETE COLUMNS } = \text{ X(1..10)}
\]

**Dummy Subscripted Variables - The ARRAY command**

If your variable names are not already subscripted, you can use the ARRAY command to assign dummy subscripts temporarily for the purpose of doing transformations inside a FOR...NEXT loop. Once their purpose is over, you can delete them using the CLEAR ARRAY command.

\[
\text{ARRAY dummyarrayname / varlist}
\]

For example:

\[
\begin{align*}
\text{USE COLOR} \\
\text{ARRAY } & \text{ X / RED GREEN BLUE}
\end{align*}
\]

The ARRAY command aliases each existing variable with a subscripted variable having the root \( X \).

<table>
<thead>
<tr>
<th>Original</th>
<th>Alias</th>
</tr>
</thead>
<tbody>
<tr>
<td>RED</td>
<td>X(1)</td>
</tr>
<tr>
<td>GREEN</td>
<td>X(2)</td>
</tr>
<tr>
<td>BLUE</td>
<td>X(3)</td>
</tr>
</tbody>
</table>
SYSTAT treats each variable specified as an element of the array X.
You can specify the same variable in more than one array. For example,

\begin{verbatim}
ARRAY U / RED GREEN YELLOW
ARRAY V / GREEN BLUE CYAN
\end{verbatim}

**Example 22**

**Computing Means of Subscripted Variables**

The following commands compute the average of the variables \(X(1)\) through \(X(10)\) for each case. The statement

\begin{verbatim}
IF X(i) <> .
\end{verbatim}

checks for missing data. (Of course, you can calculate the mean more easily with the multivariable function AVG.)

\begin{verbatim}
USE MYFILE
DIM X(10)
LET SUMX = 0
LET N = 0
FOR I = 1 TO 10
    IF X(I) <> . THEN BEGINBLOCK
        LET SUMX = SUMX + X(I)
        LET N = N + 1
    ENDBLOCK
NEXT
IF N <> 0 THEN LET MEAN = SUMX/N
ELSE LET MEAN = .
ENDIF
DSAVE NEWDATA
\end{verbatim}

The commands beginning with LET and ending with ELSE are executed once for each case. At the start of each case, LET SUMX=0 and LET N=0 set the variables \(SUMX\) and \(n\) to 0. \(SUMX\) sums the nonmissing values across each case, and \(n\) counts the nonmissing values for each case.

The FOR...NEXT loop runs the variables \(X(1...10)\) through two conditional transformations. In the first, if \(X(1)\) is not missing, its value is added to \(SUMX\). In the second, again if \(X(1)\) is not missing, the count variable \(n\) is increased by one.

Upon completion of the FOR...NEXT loop, another conditional transformation tests whether \(n\) is not equal to 0. If \(n\) is not equal to 0, then the calculation \(MEAN=SUMX/N\) is executed. If \(n\) equals 0 (because the values of \(X(1...10)\) for the current case are all missing), dividing by \(n\) would cause an error. Therefore, SYSTAT executes the ensuing ELSE statement and sets \(MEAN\) to missing (\(.\)).
**Example 23**  
**Computing Means of Un subscripted Variables: ARRAY**

To average variables that are not subscripted, use the ARRAY command to alias the variables with a subscripted variable and then use the same logic employed for subscripted variables.

The example below averages the values of the average expenditures for health, education, and the military \((HEALTH, EDUC, \text{ and } MIL)\) using the OURWORLD data file and tests for missing values.

```basic
USE OURWORLD
ARRAY DOLLARS / HEALTH  EDUC  MIL
LET SUMMONEY = 0
LET N = 0
FOR I = 1 TO 3
  IF DOLLARS(I) <> . THEN BEGIN
    LET SUMMONEY = SUMMONEY + DOLLARS(I)
    LET N = N + 1
  END
ENDFOR
IF N <> 0 THEN LET MEAN = SUMMONEY/N ELSE LET MEAN = .
DSAVE NEWDATA
```

**Example 24**  
**Fitting Normal Distribution with FOR…NEXT Statement**

Suppose you want to generate five independent columns each consisting of a random sample of size 50 from a normal distribution and test the goodness of fit. Then the input is as follows:

```basic
NEW
REPEAT 50
RSEED 123
DIM X(5)
LET (x(1..5))=ZRN(0,1)
FITDIST
FOR i = 1 to 5
  LET X = X(i)
  Continuous X / dist = Z
NEXT
```
The result of fitting normal distribution to X(1) is given below:

Variable Name : X
Distribution : Normal

Estimated Parameter(s)
Location or Mean (mu) : 0.198
Scale or SD (sigma) : 0.999

Estimation of Parameter(s): Maximum Likelihood Method

Test Results
<table>
<thead>
<tr>
<th>Lower Limit</th>
<th>Upper Limit</th>
<th>Observed</th>
<th>Expected</th>
</tr>
</thead>
<tbody>
<tr>
<td>-0.914</td>
<td>-0.508</td>
<td>6</td>
<td>6.630</td>
</tr>
<tr>
<td>-0.508</td>
<td>-0.101</td>
<td>10</td>
<td>5.361</td>
</tr>
<tr>
<td>-0.101</td>
<td>0.306</td>
<td>5</td>
<td>7.122</td>
</tr>
<tr>
<td>0.306</td>
<td>0.713</td>
<td>6</td>
<td>8.032</td>
</tr>
<tr>
<td>0.713</td>
<td>1.119</td>
<td>5</td>
<td>7.692</td>
</tr>
<tr>
<td>1.119</td>
<td>0.306</td>
<td>12</td>
<td>8.909</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>50</td>
<td>50.000</td>
</tr>
</tbody>
</table>

Chi-square Test Statistic: 7.246
Degrees of Freedom: 4
p-value 0.123

Kolmogorov-Smirnov Test Statistic: 0.098
Lilliefors Probability: 0.250

Shapiro-Wilk Test Statistic: 0.977
p-value: 0.435
**Temporary Arrays - The ARRAY Command**

You may also use the ARRAY command to create an array of temporary variables of specified size in memory.

```
ARRAY varname~(size)
```

You can clear the array from memory using CLEAR ARRAY command.

**Example 25**

**Creating and Accessing Temporary Array**

The following example shows how to create and access an array of temporary variables:

```
ARRAY X~(5)
RSEED 1000
FOR K =1 TO 5
  TMP X~(K) = ZRN(0,1)
PRINT X~(K)
NEXT
CLEAR ARRAY = X~
```

The output is:

```
0.476
0.338
-0.350
-0.107
-0.295
```

**Example 26**

**Random Subsamples without Replacement**

This example illustrates two methods of taking random samples of cases without replacement from data files. The first extracts a percentage of cases from a file; the second, a specific number of cases.

To pick a random sample of approximately three-fourths of a file (*USSTATES*), type:

```
USE USSTATES
IF URN () > .75 THEN DELETE
```

To vary the sample size, change the 0.75 proportion to another number between 0 and 1. The output is a file containing only the selected cases.
As a variation of the output of this method, both the selected and deselected cases can be kept in the same file, with a newly created \textit{COUNT} variable taking values 1 (selected) or 0 (deselected). This \textit{COUNT} variable can be invoked in statistical procedures as the Frequency variable to use only the selected random subsample for cross-validation and other purposes.

\begin{verbatim}
USE USSTATES
  IF URN () > .75 THEN LET COUNT = 0,
  ELSE LET COUNT = 1
\end{verbatim}

The second method picks a random sample of size exactly \textit{ns~} without replacement from a file of size \textit{nf~} cases, using an algorithm due to Bebbington (1975). To select such a random sample of size 10 from the \textit{USSTATES} file, and output only the selected cases, type:

\begin{verbatim}
USE USSTATES
RSEED 1000
TMP nf~ = NCASE()
TMP ns~ = 10
LET RAND = URN()
FOR
  IF RAND > (ns~/nf~) THEN DELETE,
  ELSE TMP ns~ = ns~ - 1
  TMP nf~ = nf~ - 1
NEXT
DELETE COLUMNS = RAND
CLEAR VARIABLES = nf~, ns~
LIST STATE$
\end{verbatim}

The output is of the following kind:

<table>
<thead>
<tr>
<th>Case</th>
<th>STATE$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>IN</td>
</tr>
<tr>
<td>2</td>
<td>IL</td>
</tr>
<tr>
<td>3</td>
<td>MI</td>
</tr>
<tr>
<td>4</td>
<td>IA</td>
</tr>
<tr>
<td>5</td>
<td>MO</td>
</tr>
<tr>
<td>6</td>
<td>WV</td>
</tr>
<tr>
<td>7</td>
<td>AZ</td>
</tr>
<tr>
<td>8</td>
<td>NV</td>
</tr>
<tr>
<td>9</td>
<td>OR</td>
</tr>
<tr>
<td>10</td>
<td>CA</td>
</tr>
</tbody>
</table>
Example 27
Random Subsample with Replacement

The following program picks a sample of size $ns$ with replacement from a file of size $nf$ cases. The file used is RAINFALL. The output consists of all cases in the file with their frequency of occurrence in the sample (0 if not selected and 1 or more if selected). Notice that the frequency can be greater than 1 since sampling is with replacement.

```
USE RAINFALL
RSEED 1000
TMP ns~ = 6
TMP nf~ = 10
ARRAY ctr~(10)  !! Defining a temporary array ctr~ of size nf~
FOR i=1 to nf~
  TMP ctr~(i)=0
NEXT
FOR i=1 TO ns~
  TMP rand~ = durn(nf~)
  TMP ctr~(rand~)=ctr~(rand~)+1
NEXT
FOR i~ = 1 to nf~
  IF CASE()=i~ then FOR
    LET count=ctr~(i~)
    TMP i~ = nf~
  NEXT
NEXT
CLEAR ARRAY= ctr~
CLEAR VARIABLE = ns~, nf~, rand~
LIST
```

The output is of the following kind:

<table>
<thead>
<tr>
<th>Case</th>
<th>X</th>
<th>Y</th>
<th>COUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>23.900</td>
<td>41.000</td>
<td>0.000</td>
</tr>
<tr>
<td>2</td>
<td>43.300</td>
<td>52.000</td>
<td>0.000</td>
</tr>
<tr>
<td>3</td>
<td>36.300</td>
<td>18.700</td>
<td>0.000</td>
</tr>
<tr>
<td>4</td>
<td>40.600</td>
<td>55.000</td>
<td>2.000</td>
</tr>
<tr>
<td>5</td>
<td>57.000</td>
<td>40.000</td>
<td>1.000</td>
</tr>
<tr>
<td>6</td>
<td>52.500</td>
<td>29.200</td>
<td>0.000</td>
</tr>
<tr>
<td>7</td>
<td>46.100</td>
<td>51.000</td>
<td>2.000</td>
</tr>
<tr>
<td>8</td>
<td>142.000</td>
<td>17.600</td>
<td>0.000</td>
</tr>
<tr>
<td>9</td>
<td>112.600</td>
<td>46.600</td>
<td>1.000</td>
</tr>
<tr>
<td>10</td>
<td>23.700</td>
<td>57.000</td>
<td>0.000</td>
</tr>
</tbody>
</table>

The above program creates a new variable called COUNT which represents the frequency of each case in the sample. This COUNT variable can be invoked in statistical procedures as the Frequency variable to use only the selected random subsample for cross-validation and other purposes. For example, the following commands compute statistics only using the selected sample.
Chapter 7

FREQ COUNT
CSTATISTICS

The output is:

Case frequencies determined by value of variable COUNT

|       X        | Y       |
|-------------------+------------------|
| N of Cases         | 6        6       |
| Minimum            | 40.600   40.000  |
| Maximum            | 112.600  55.000  |
| Arithmetic Mean    | 57.167   49.767  |
| Standard Deviation | 27.809   5.714   |

If the nonselected cases are not desired in the output file, add a command:

IF COUNT = 0 THEN DELETE

Example 28
Calendar for the Year 2009-2010

The following program generates a calendar for the year 2009-2010.

PUSH ALL
CLASSIC ON
FORMAT 12, 0
NEW
ARRAY DAY$~(7)
ARRAY DATE$~(7)
FOR YEAR = 2009 TO 2010
FOR MONTH = 1 TO 12
TMP BASE~ = DOC(YEAR, MONTH, 1) - 1
TMP LAST~ = BASE~ + 31
TMP FIRST~ = INT((BASE~ + 1)/7) * 7
TMP MONTH$~ = STR$(BASE~ + 1, 'MMMMMMMM')
PRINT CNT$(CAT$(UPR$(MONTH$~), RGT$(STR$(YEAR), 5)), 33)
PRINT '*-------------------------------*'
FOR i = 1 TO 7
TMP DAY$~(i) = MID$(DOW$(i - 1), 1, 3)
NEXT
PRINT DAY$~(1), '', DAY$~(2), '', DAY$~(3), '',
DAY$~(4), '', DAY$~(5), '', DAY$~(6), '', DAY$~(7)
FOR i = FIRST~ TO LAST~ STEP 7
FOR l = i TO i + 6
TMP T~ = l + 1 - i
IF l - BASE~ < 10 THEN,
TMP DATE$~(T~) = CAT$(l, STR$(l - BASE~)),
ELSE TMP DATE$~(T~) = STR$(l - BASE~)
IF (DAT(l, 'M')<>MONTH) THEN TMP DATE$~(T~) = '.
NEXT
PRINT DATE$~(1), ' ', DATE$~(2), ' ', DATE$~(3), ' ',
DATE$~(4), ' ', DATE$~(5), ' ', DATE$~(6), ' ',
DATE$~(7)
NEXT
NEXT
PRINT '-------------------------------*
NEXT
CLEAR ARRAYS = DAY$, DATE$
CLEAR VARIABLES = T$
POP ALL

The calendar follows:

SYSTAT Created a temp variable "T~".

JANUARY 2009
*-------------------------------*
Sun Mon Tue Wed Thu Fri Sat
. . . . 1 2 3
4 5 6 7 8 9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30 31

FEBRUARY 2009
*-------------------------------*
Sun Mon Tue Wed Thu Fri Sat
1 2 3 4 5 6 7
8 9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
. . . . . .

MARCH 2009
*-------------------------------*
Sun Mon Tue Wed Thu Fri Sat
1 2 3 4 5 6 7
8 9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31 . . . .

APRIL 2009
*-------------------------------*
Sun Mon Tue Wed Thu Fri Sat
. . . 1 2 3 4
5 6 7 8 9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 . .

MAY 2009
*-------------------------------*
Sun Mon Tue Wed Thu Fri Sat
. . . 1 2 3 4
5 6 7 8 9 10 11
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
31 . . . . . .

JUNE 2009
*-------------------------------*
Sun Mon Tue Wed Thu Fri Sat
. 1 2 3 4 5 6
7 8 9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 . . . .
Functions

Apart from the built-in functions that SYSTAT provides, you may also define your own functions and use them in your program.

A function is a series of BASIC statements enclosed by the FUNCTION and ENDFUNC commands. The function performs a task and then returns control to the calling statement. When it returns control, it also returns a value to the calling statement.

The syntax for defining a function is as follows:

```
FUNCTION TMP funcname(TMP arg1, TMP arg2, ...)  
statement 1  
statement 2  
.  
.  
RETURN expression  
ENDFUNC
```

The `funcname` should be different from SYSTAT library functions and other existing functions. You can define functions which return only numeric values. Within a function, only temporary variables are allowed for any manipulation. You can pass temporary numeric variables or numeric constants as argument values in the calling function.

The CLEAR FUNCTION command is used to clear the user-defined functions from memory.

```
CLEAR FUNCTION = funcname1, funcname2,...
```
Example 29  
Defining and Using Functions

The following program prints the number of combinations of \( n \) things taken \( r \) at a time using the user-defined functions \( FACT() \) and \( COMB() \).

```plaintext
FUNCTION TMP FACT(TMP x~)
TMP p~ = 1
IF x~ > 0 THEN
FOR i = 1 TO x~
TMP p~ = p~ * i
NEXT
ENDIF
RETURN p~
ENDFUNC

FUNCTION TMP COMB(TMP n~, TMP r~)
RETURN FACT(n~)/(FACT(n~-r~)*FACT(r~))
ENDFUNC

PRINT "The value of COMB(4,2) is ", COMB(4,2)
CLEAR FUNCTION = FACT, COMB
```

The output is:

```
The value of COMB(4,2) is 6.000
```
Macro

A macro is a series of statements enclosed by the DEFMACRO and ENDMACRO commands. A macro performs a task and then returns control to the calling statement. Macros are used for several purposes. Suppose you want to execute the same code in many different places in a program. Macros allow you to execute a series of commands any number of times with one copy of the code.

The syntax for defining a macro is as follows:

```
DEFMACRO macname(arg1,arg2,...)
  statement 1
  statement 2
  ...
ENDMACRO
```

The arguments are optional (the parentheses are mandatory though) in which case the macro will just represent a block of static commands. For example,

```
DEFMACRO MYSTAT()
  CSTAT / MEAN SD MEDIAN MODE
ENDMACRO
```

Alternatively, file variables may be specified as arguments.

To invoke a macro in a program, use the CALLMACRO command:

```
CALLMACRO macname(arg1,arg2,...)
```

A macro must be defined in your program before it is called.

Use the CLEAR MACRO command to clear one or more previously defined macros from memory.

```
CLEAR MACRO = macname1, macname2 ...
```
Example 30
Defining and Using Macros

The following program generates various summary charts and scatterplots by calling the same macro for various data files.

```
DEFMACRO GRAPHS(X,Y)
  BAR X
  LINE X
  PIE X
  PROFILE X
  PYRAMID X
  PLOT Y/SYMBOL = X
ENDMACRO
USE Afifi
CALLMACRO GRAPHS(DRUG,SYSINCR)
USE iris
CALLMACRO GRAPHS(SPECIES, SEPALLEN)
CALLMACRO GRAPHS(SPECIES, PETALLEN)
```

Programming Examples

The examples in this section show more applications of SYSTAT transformation commands, including statistical calculations and data management procedures. There are simpler ways to accomplish many of these tasks, but the programs in this chapter are selected to illustrate the full range of SYSTAT capabilities through programming.

Select, List, and Save Cases

Use IF...THEN DELETE statements to select a subset of cases and store them in a SYSTAT data file, or use PUT or PRINT to save them in a text file. (You can also save selected cases using the SELECT command with EXTRACT.)

```
USE MYFILE
IF condition THEN DELETE
DSAVE subfile
```

or

```
USE MYFILE
IF condition THEN DELETE
PUT subtext
```
Example 31
Deleting Selected Cases

The following commands drop certain cases and list the variable \textit{COUNTRY}\textsuperscript{$$}$ to show the cases SYSTAT retained:

\begin{verbatim}
USE MINIWRLD
IF CASE() <= 9 THEN DELETE
LIST COUNTRY$
\end{verbatim}

SYSTAT responds:

<table>
<thead>
<tr>
<th>Case</th>
<th>COUNTRY$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Afghanistan</td>
</tr>
<tr>
<td>2</td>
<td>Brazil</td>
</tr>
<tr>
<td>3</td>
<td>Canada</td>
</tr>
<tr>
<td>4</td>
<td>Venezuela</td>
</tr>
<tr>
<td>5</td>
<td>ElSalvador</td>
</tr>
</tbody>
</table>

Example 32
Saving Selected Cases to a New File

The following commands save only those cases in \textit{MINIWRLD} where \textit{GDP\_CAP} is greater than 10,000. The results are stored in \textit{NEWFILE}.

The input is:

\begin{verbatim}
USE MINIWRLD
IDVAR COUNTRY$
IF GDP\_CAP < 10000 THEN DELETE
DSAVE NEWFILE
LIST
\end{verbatim}

The output follows:

<table>
<thead>
<tr>
<th>Case ID</th>
<th>COUNTRY$</th>
<th>POP_1990</th>
<th>URBAN</th>
<th>BIRTH_RT</th>
<th>BABYMORT</th>
<th>GDP_CAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>France</td>
<td>France</td>
<td>56.358</td>
<td>73.000</td>
<td>14.000</td>
<td>6.000</td>
<td>14,542.657</td>
</tr>
<tr>
<td>Switzerland</td>
<td>Switzerland</td>
<td>6.742</td>
<td>58.000</td>
<td>12.000</td>
<td>5.000</td>
<td>17,723.499</td>
</tr>
<tr>
<td>Spain</td>
<td>Spain</td>
<td>39.269</td>
<td>91.000</td>
<td>11.000</td>
<td>6.000</td>
<td>10,153.121</td>
</tr>
<tr>
<td>UK</td>
<td>UK</td>
<td>57.366</td>
<td>76.000</td>
<td>14.000</td>
<td>7.000</td>
<td>14,259.401</td>
</tr>
<tr>
<td>Canada</td>
<td>Canada</td>
<td>26.538</td>
<td>76.000</td>
<td>14.000</td>
<td>7.000</td>
<td>19,353.214</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Case ID</th>
<th>EDUC</th>
<th>HEALTH</th>
<th>MIL</th>
<th>GOVS</th>
<th>LIFEEXPM</th>
<th>LIFEEXPF</th>
</tr>
</thead>
<tbody>
<tr>
<td>France</td>
<td>648.069</td>
<td>728.249</td>
<td>432.780</td>
<td>Democracy</td>
<td>73.000</td>
<td>82.000</td>
</tr>
<tr>
<td>Switzerland</td>
<td>853.538</td>
<td>1,209.077</td>
<td>330.769</td>
<td>Democracy</td>
<td>75.000</td>
<td>83.000</td>
</tr>
<tr>
<td>Spain</td>
<td>166.675</td>
<td>224.923</td>
<td>330.769</td>
<td>Democracy</td>
<td>75.000</td>
<td>82.000</td>
</tr>
<tr>
<td>UK</td>
<td>474.488</td>
<td>479.064</td>
<td>447.473</td>
<td>Democracy</td>
<td>73.000</td>
<td>79.000</td>
</tr>
<tr>
<td>Canada</td>
<td>1,052.813</td>
<td>936.211</td>
<td>316.797</td>
<td>Democracy</td>
<td>74.000</td>
<td>81.000</td>
</tr>
</tbody>
</table>
Instead of using an IF...THEN DELETE statement to delete cases that are less than 10,000, you could have used SELECT:

```
SELECT GDP_CAP >= 10000
EXTRACT NEWFILE
```

**Example 33**

**Omitting Variable Names and Case Numbers**

The PRINT command is similar to LIST, except that case numbers and variable names are not printed. Also, PRINT can be used with IF THEN... DELETE, allowing you to print the first $k$ cases. Type:

```
USE MINIWRLD
IF CASE() > 3 THEN DELETE
PRINT COUNTRY$, LITERACY, URBAN, BIRTH_RT
```

SYSTAT responds:

<table>
<thead>
<tr>
<th>Case ID</th>
<th>GDP_CAP</th>
<th>LITERACY</th>
<th>URBAN</th>
<th>BIRTH_RT</th>
</tr>
</thead>
<tbody>
<tr>
<td>France</td>
<td>99.000</td>
<td>73.000</td>
<td>14.000</td>
<td></td>
</tr>
<tr>
<td>Greece</td>
<td>95.000</td>
<td>65.000</td>
<td>11.000</td>
<td></td>
</tr>
<tr>
<td>Switzerland</td>
<td>99.000</td>
<td>58.000</td>
<td>12.000</td>
<td></td>
</tr>
</tbody>
</table>

Case numbers and variables are not printed. SYSTAT prints numeric and character values in 12-column, right-justified fields. Blanks pad the left side of numeric fields. With PAGE set to WIDE, you can display up to 10 variables per line. If you use FORMAT to set the field width to fewer than 12 characters per field, you can display more than 10 variables per line.

**Example 34**

**Using PRINT to Save Selected Cases**

You can select specific cases by using the IF...THEN PRINT command. The example uses the OUTPUT command to store the values of GROUP$, COUNTRY$, LITERACY, BIRTH_RT, and LIFEXPF (for Islamic countries only) in the text file `TEXTFILE.DAT`:

```
SELECT GDP_CAP >= 10000
EXTRACT NEWFILE
USE MINIWRLD
IF CASE() > 3 THEN DELETE
PRINT COUNTRY$, LITERACY, URBAN, BIRTH_RT
```

<table>
<thead>
<tr>
<th>Case ID</th>
<th>GDP_CAP</th>
<th>LITERACY</th>
<th>URBAN</th>
<th>BIRTH_RT</th>
</tr>
</thead>
<tbody>
<tr>
<td>France</td>
<td>99.000</td>
<td>73.000</td>
<td>14.000</td>
<td></td>
</tr>
<tr>
<td>Greece</td>
<td>95.000</td>
<td>65.000</td>
<td>11.000</td>
<td></td>
</tr>
<tr>
<td>Switzerland</td>
<td>99.000</td>
<td>58.000</td>
<td>12.000</td>
<td></td>
</tr>
</tbody>
</table>
Example 35

Printing the First Three Cases

If you want to see only some of the cases, you can specify the number of cases by using the IF condition THEN DELETE option. The following example sends the first three cases of MINIWRLD to the NEWFILE.DAT file:

```basic
USE MINIWRLD
PUSH CLASSIC
CLASSIC ON
OUTPUT NEWFILE
IF CASE() > 3 THEN DELETE
PRINT COUNTRY$, LITERACY, URBAN, BIRTH_RT
OUTPUT *
POP CLASSIC
```
Subgroup Processing: \texttt{BOG()}, \texttt{EOG()}, \texttt{BOF()}, \texttt{EOF()}

You can specify SYSTAT commands that operate on subgroups of cases in a file. To do this, you must first identify variables in your file which define subgroups. SYSTAT has four special functions available for processing subgroups:

- \texttt{BOF()} returns 1 if beginning-of-file, else it returns 0.
- \texttt{EOF()} returns 1 if end-of-file, else it returns 0.
- \texttt{BOG()} returns 1 if beginning-of-BY group, else it returns 0.
- \texttt{EOG()} returns 1 if end-of-BY group, else it returns 0.

Before using \texttt{BOG()} or \texttt{EOG()}, you must use a \texttt{BY} statement to identify the variables that define subgroups in your data. To clear a previous \texttt{BY} command, type \texttt{BY} with no arguments.

You can use \texttt{BOG()}, \texttt{EOG()}, \texttt{BOF()}, and \texttt{EOF()} within conditional expressions in \texttt{IF...THEN} statements. For example, the statement

\begin{verbatim}
IF BOG() THEN statement
\end{verbatim}

causes SYSTAT to execute the \texttt{statement} every time it encounters a new value in a \texttt{BY} variable. This is because the \texttt{BOG()} returns 1 ("true") for every case that begins a new group (and 0 otherwise).

Example 36
\textit{EOF()}: Printing the Last Case in a File

The following procedure prints the value of \texttt{CARDIO} for the last case in the \texttt{USDATA} file. \texttt{EOF} is 0 for every case but the last, where its value is 1.

\begin{verbatim}
USE USDATA
IF EOF() THEN PRINT,
   "The CARDIO value for the last case is", CARDIO
\end{verbatim}

To print all but the last case, set the condition to one of the following:

\begin{verbatim}
IF EOF()=0 THEN...
IF NOT EOF() THEN...
\end{verbatim}
Example 37
Computing Totals within Groups

Suppose you want to sum values of the variable x within GROUP1 and within GROUP2. You start with a file called OURFILE and want to write a file called OURFILE2.

<table>
<thead>
<tr>
<th>GROUP</th>
<th>X</th>
<th>GROUP</th>
<th>X</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>1</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>1</td>
<td>6</td>
<td>13</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>2</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>-1</td>
<td>2</td>
<td>-1</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>10</td>
</tr>
</tbody>
</table>

The commands to do this are:

```
USE OURFILE
BY GROUP
IF BOG( ) THEN LET TOTAL=X,
ELSE LET TOTAL=LAG(TOTAL) + X
IF EOG( ) THEN PRINT GROUP, TOTAL
DSAVE OURFILE2
```

Condition is true (BOG ( ) returns 1) when a case is first in its group; otherwise, it is false.

Saving Only the Totals with Group Identifiers

If you want OURFILE2 to include only the summary record for each group,

```
GROUP   TOTAL
1       13
2       10
```

insert:

```
IF NOT EOG( )THEN DELETE
```

after DSAVE and

```
DELETE COLUMNS = X
```
after DSAVE in the last setup. EOG( ) means “end of group.”

**Example 38**  
**Subgroup Means**

You can use a variable \((n)\) that counts the number of cases within each group to compute the mean within each group:

\[
\text{mean} = \frac{\text{TOTAL}}{\text{N}}
\]

The data in the output file `OURFILE3` are:

<table>
<thead>
<tr>
<th>GROUP</th>
<th>MEAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4.333</td>
</tr>
<tr>
<td>2</td>
<td>3.333</td>
</tr>
</tbody>
</table>

The input follows:

```
USE OURFILE
BY GROUP
IF BOG( ) THEN LET TOTAL=X,
ELSE LET TOTAL=LAG(TOTAL) + X
IF BOG() THEN LET N=1,
ELSE LET N=LAG(N) + 1
IF EOG() THEN LET MEAN=TOTAL/N
IF NOT EOG() THEN DELETE
DSAVE OURFILE3
DELETE COLUMNS = X, TOTAL, N
```

**Example 39**  
**IF...THEN DELETE: Using the first \(N\) Cases**

For any set of SYSTAT commands, you can use the `IF...THEN DELETE` command to limit the computations to a certain number of cases—just as you can use `SELECT Case( ) <=10` to limit the action of commands like `LIST` to the first 10 cases.

You can use `IF...THEN DELETE` to test complex transformations. If you use `IF...THEN DELETE`, you can see whether the commands are correct or if you need to change them before running it on an entire file. For example:
If you made a mistake while writing the program, you would find the error before running the program on the entire file. For files with several hundred cases, a brief trial run can save time. If you use `SELECT( ) <= 3` in place of `IF CASE( ) > 3 THEN DELETE`, then after you correct any errors, type `SELECT( )` with no arguments to restore the counter.

**References**


Chapter 8

Matrix

Laszlo Engelman
(revised by A.V.Kharshikar, Amit Saxena, S.R.Kulkarni, Nandita Ingawale, and Santosh Ranjan)

Most of the algorithms for classical analyses provided by statistical software packages can be expressed using matrix algebra; thus, they can be computed using the Matrix feature. Students will find Matrix useful for gaining an understanding of matrix algebra and how statistical computations work. Researchers can prototype state-of-the-art procedures before they become available in statistical packages, and can execute complex data management tasks.

Useful material on matrix algebra may be found in Searle (1982), Schott (1997), Harville (1997), and Rao and Rao (1998). These books discuss the use of matrices in Statistics. Rao (1973), Seber and Lee (2003), Kuttner et al. (2004), and Birkes and Dodge (1993) have useful chapters or an appendix dealing with matrix algebra. Matrix in SYSTAT offers functions that allow you to use matrix algebra to perform statistical analyses and data management tasks. Matrix provides a flexible facility to manipulate matrices in addition to performing matrix algebra. Most of the standard matrix manipulations, operations, and computations can now be menu-driven. You have the choice, of course, to achieve them through commands as in earlier versions. Functions like QR, Cholesky, and singular value decomposition are available for many statistical analyses. Procedures like Kronecker product, raising a matrix to a power, and solving a consistent system of $n$ equations in $n$ variables are available. Reshaping a matrix, filling missing values, making a matrix symmetric, and many more facilities are provided to handle complex data management tasks.
Matrix in SYSTAT

The Matrix feature is organized as follows:

A matrix is referred to by its name. We refer the matrices in the memory as available matrices and the last handled matrix as the active matrix. Matrix allows as many matrices as the memory allows. A matrix may contain all columns with either numeric or character values. However, Matrix fails if a numeric operation is attempted on character data or if operands are mixed; for example, comparing a character datum to a numeric value.

Read Matrix Dialog Box

To open the Read Matrix dialog box, from the menus choose:

Utilities
Matrix
Read...
The following options can be specified:

**Matrix name.** Enter a name for the matrix that you want to create.

**Read from.** You need to choose one from the following options:
- **File.** Select a file that you want to read into a matrix.
- **Matrix type.** Choose either numeric or the string type of the matrix.
- **Selected variable(s).** Variable(s) selected for representing the columns of the matrix.
- **Selected case(s).** Type the row numbers separated by commas or spaces, use double period (..) to specify a range of rows (e.g. 1...20).
- **Keyboard.** Enter the values separated by commas or spaces. Separate each row with a semicolon, if you type several rows on one line.

**Row name(s).** You can specify names for the rows of the matrix. Use commas or spaces to separate the names.

**Column name(s).** You can specify names for the columns of the matrix. Use commas or spaces to separate the names.
Save matrix. You can save the created matrix to a specified file. The following options are available: Rectangular, SSCP, Covariance, Correlation, Dissimilarity and Similarity.

Show matrix. Prints the created matrix to the current output device.

**Generate Matrix Dialog Box**

To open the Generate Matrix dialog box, from the menus choose:

Utilities
Matrix
Generate…

Matrix name. Enter a name for the matrix you want to generate.
**Generate.** You need to choose one from the following options:

- **Identity matrix of order.** Generates an identity matrix of a specified order.
- **n x p matrix of values.** Generates a rectangular matrix with \( n \) rows and \( p \) columns filled with a value that you specify.

**Selected matrix/vector.** Generates a new matrix on the Selected matrix/vector for the following actions:

- **Diagonal elements as row.** Extracts the diagonal elements of the selected square matrix.
- **Diagonal matrix from vector.** Generates a diagonal matrix with diagonal elements from the selected vector.
- **Super-diagonal; diagonal; sub-diagonal.** Extracts the super-diagonal, diagonal, and sub-diagonal of the selected square matrix.

**Row name(s).** You can specify names for the rows of the matrix. Use commas or spaces to separate the names.

**Column name(s).** You can specify names for the columns of the matrix. Use commas or spaces to separate the names.

**Save matrix.** You can save the generated matrix to a specified file. The following options are available: Rectangular, SSCP, Covariance, Correlation, Dissimilarity and Similarity.

**Show matrix.** Prints the generated matrix to the current output device.

---

**Organize Dialog Box**

To open the Organize dialog box, from the menus choose:

Utilities
Matrix
Organize...
You can clear the desired matrices from memory, save the matrices, view matrices, and/or see the description of matrices.

**Directory.** Gives a description of the selected matrices.

**Show.** Prints the selected matrices to the current output device.

**Save.** You can save the selected matrices to a specified file.

**Clear.** Clears the selected matrices from memory.

**Save path.** Lets you choose the path for saving the matrices.

You can perform any of the above operations on the same matrix.

**Row/Column Operations Dialog Box**

To perform Row/Column operations on an active matrix, from the menus choose:

Utilities
Matrix
Row/Column Operations…
Column/Matrix and Argument. To specify the target column/matrix, select a column/matrix from the list and click the Column/Matrix button. If you want to create a new column (matrix), simply type it into the box. Select the function type, functions and column (matrix or matrices) to be used in the argument.

You can operate on existing matrices or create new ones. If the target column (matrix) already exists, it is replaced by the expression written in argument. If the matrix does not exist, then a new matrix with that name is created.

Operation. You need to choose one from the following options:

- **Let.** Creates a new column or alters an existing column in the active matrix. Note that LET works in the same manner here as LET does elsewhere in SYSTAT.

- If the active matrix has a column named *NAME*, it is overwritten; if it does not, a new column is added. The expression can be any combination of column names, constants, mathematical functions, and operators.
See Chapter 4: Data Transformations: “Let Dialog Box” on page 83 in *DATA* for more information about LET statements.

- **Drop column(s).** Deletes columns of the active matrix specified by variable names, column numbers, or a range of columns (for example, 10 .. 31).
- **Extract sub-matrix.** You can extract a sub-matrix from the active matrix. The expression for that is:

  \[
  \text{name} = \text{mat}_\text{name}(\text{ref to rows}; \text{ref to columns}).
  \]

Reference to rows/columns can be a list or range of row/column numbers or a list of row/column names or a condition involving column names (for example AGE > 21). Use a double period (..) to specify a range of rows or columns.

- **Select row(s).** Retains only those rows of the active matrix that meet the specified condition. You can specify the condition using functions and column names.
- **Delete row(s).** Deletes rows of the active matrix specified by row names, row numbers, or a range of rows.

**Show matrix.** Prints the generated matrix to the current output device.

**Save matrix.** Saves the generated matrix to a SYSTAT data file in the specified path. The following options are available: Rectangular, SSCP, Covariance, Correlation, Dissimilarity and Similarity.

**Matrix Operations Dialog Box**

To perform a variety of matrix operations, from the menus choose:

Utilities
Matrix
Matrix Operations…
**Matrix and Expression.** To specify the target matrix, select a matrix from the list and click the Matrix button. If you want to create a new matrix, simply type it into the box.

**Operation type.** Select the Operation type, Operations, and Matrix (matrices) to be used in the expression. Your expression will appear in the Expression area as you create it.

You can operate on existing matrices or create new ones. If the target matrix already exists, it is replaced by the Expression. If the matrix does not exist, then a new matrix with that name is created.

**Operations.** The following operations are available:

**Arithmetic**
- Addition (+)
- Subtraction (-)
- Scalar Division (/)
- Element by element multiplication (#)
- Raising each element to a power (##)

**Transformation**
- Square root (SQR)
- Hyperbolic tangent (TNH)
- Log base e (LOG)
- Log gamma (LGM)
- Log base 10 (L10)
- Integer part (INT)
<table>
<thead>
<tr>
<th>Algebraic Operations</th>
<th>Mathematical Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiplication (*)</td>
<td>** or ^</td>
</tr>
<tr>
<td>Inversion (INV)</td>
<td></td>
</tr>
<tr>
<td>Kronecker Product (KRON)</td>
<td></td>
</tr>
<tr>
<td>Determinant (DET)</td>
<td></td>
</tr>
<tr>
<td>Log Determinant (LOGDET)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Statistical Operations</th>
<th>Mathematical Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Row Mean (ROWMEAN)</td>
<td></td>
</tr>
<tr>
<td>Column Mean (COLMEAN)</td>
<td></td>
</tr>
<tr>
<td>Row Standard Deviation (ROWSTD)</td>
<td></td>
</tr>
<tr>
<td>Column Standard Deviation (COLSTD)</td>
<td></td>
</tr>
<tr>
<td>Row Sum (ROWSUM)</td>
<td></td>
</tr>
<tr>
<td>Column Sum (COLSUM)</td>
<td></td>
</tr>
<tr>
<td>Min. value in rows (ROWMIN)</td>
<td></td>
</tr>
<tr>
<td>Min. value in columns (COLMIN)</td>
<td></td>
</tr>
<tr>
<td>Max. value in rows (ROWMAX)</td>
<td></td>
</tr>
<tr>
<td>Max. value in columns (COLMAX)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Manipulations Operations</th>
<th>Mathematical Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transpose (TRP)</td>
<td>Number of columns (NCOL)</td>
</tr>
<tr>
<td>Sort rows (ROWSORT)</td>
<td>Column vector contains the elements as lower triangular Matrix (STRING)</td>
</tr>
<tr>
<td>Sort columns (COLSORT)</td>
<td></td>
</tr>
<tr>
<td>Lower triangular portion copy to upper triangular portion (FOLD)</td>
<td>Matrix with the elements of vector as the lower triangular portion and missing values above the diagonal (GNIRTS(V))</td>
</tr>
<tr>
<td>Reshape the Matrix (SHAPE)</td>
<td></td>
</tr>
<tr>
<td>Filling in missing values (FILL)</td>
<td></td>
</tr>
<tr>
<td>Drop all rows containing missing data (COMPLETE)</td>
<td>Concatenate end to end (///)</td>
</tr>
<tr>
<td>Dimension (DIM)</td>
<td>Concatenate corner to corner (/</td>
</tr>
<tr>
<td>Number of rows (NROW)</td>
<td>Compare two matrices element-by-element (EQUAL)</td>
</tr>
</tbody>
</table>

| Exponentiation (EXP)   | Arc sine (ASN) |
| Absolute value (ABS)   | Arc cosine (ACS) |
| Sine (SIN)             | Arc tangent (ATN) |
| Cosine (COS)           | Arc tangent hyperbolic (ATH) |
| Tangent (TAN)          | Arc tangent (AT2) |
Relational/Logical

- Less than (<)
- Greater than (>)
- Less than or equal to (<=)
- Greater than or equal to (>=)
- Not equal to (<>)

Equal to (= =)
AND
OR
NEGATION (NOT)

Design

- “0,1” design variables (DESIGN0)
- “1,0,-1” design variables (DESIGN1)
- Full rank design variables (DESIGNF)

Equally spaced orthogonal components (ORTHEQ)
Unequally spaced orthogonal components (ORTHUN)
Orthogonal components for polynomial regression (HORTHLIN)

Row name(s). You can specify names for the rows of the matrix. Use commas or spaces to separate the names.

Column name(s). You can specify names for the columns of the matrix. Use commas or spaces to separate the names.

Save matrix. You can save the generated matrix to a specified file. The following options are available: Rectangular, SSCP, Covariance, Correlation, Dissimilarity and Similarity.

Show matrix. Prints the generated matrix to the current output device.

Decomposition Dialog Box

To open the Decomposition dialog box, from the menus choose:

Utilities
Matrix
Decomposition…
The following options can be specified:

**Input matrix.** Enter a name for the matrix you want to decompose.

**Output matrices.** Select names for output matrices from the list or just type names separated by space.

**Decomposition type.** You need to choose one from the following options:

- **Eigen.** This gives two matrices, the first consists of eigenvalues and the second, eigenvectors of the input matrix.
- **Chol.** This gives the diagonal matrix and the lower triangular matrix obtained by Cholesky decomposition of the input matrix.
- **QRD.** This decomposes the input matrix into Q and R, where the elements of R are 0 below the diagonal; that is $r_{jk} = 0$ whenever $j > k$.
- **SVD.** This gives the orthogonal matrix U, matrix of singular values D, and the orthogonal matrix V obtained in the singular value decomposition of the input matrix.

**Show matrices.** Print the generated matrices to the current output device.

**Save matrices.** You can save all output matrices to specified files.
Using Commands

For reading a matrix:

USE filename/ MAT = name
MAT name = [a11, a12, ..., a1n; .......;am1, am2, ..., amn]
ROWNAME or COLNAME matrix = name1, name2 name3,...
SHOW matrix1, matrix2,...
SAVE namel / MAT = name2

For generating matrices:

MAT name = GAID or DIAG or DIAG3 (argument)
MAT name = I(order)
MAT name = M(n, p, value)
ROWNAME or COLNAME matrix = name1, name2 name3,...
SHOW matrix1, matrix2,...
SAVE namel / MAT = name2

For organizing matrices:

DIRECTORY or SHOW or CLEAR MATRIX = name1,name2,...
MSAVE namel / MAT = name2

For performing row/column operations:

MLET result = function(varname)
MDELETE COLUMNS = columnlist
MAT name = mat_name (ref to rows; ref to columns)
MDELETE ROWS = rowlist
MSELECT condition
SHOW matrix1, matrix2,...
MSAVE namel / MAT = name2

For performing matrix operations:

MAT result = name1 operator name2
MAT result = function (argument)
ROWNAME or COLNAME matrix = name1, name2 name3,...
SHOW matrix1, matrix2,...
MSAVE namel / MAT = name2

(operator +, -, /, #, ##,* , **, ^, ||, //, /|, <, >, <=, >=, <>, = =, and or, not)

(function = TRP, ROWSORT, COLSORT, FOLD, SHAPE, FILL, COMPLETE, DIM, NROW, NCOL, STRING, GNIRTS, EQUAL, INV, KRON, DET, LOGDET, TRACE, EIGVAL, SWEEP, SOLVE, CHOL, ROWMEAN, COLMEAN, ROWSTD, COLSTD, ROWSUM, COLSUM, ROWMIN, COLMIN, ROWMAX, COLMAX, ROWMIS, COLMIS, ROWNUM,
COLNUM, ROWZSC, COLZSC, ROWRANK, COLRANK, CORR, COVA, SSCP, DESIGN0, DESIGN1, DESIGNF, ORTHEQ, ORTHUN, HORTHUN, SQR, LOG, L10, EXP, ABS, SIN, COS, TAN, TNH, LGM, INT, ASN, ACS, ATN, ATH, AT2

For decomposing matrices:

CALL EIGEN (or CHOL or QRD or SVD) (name1, name2, … namen)
SHOW matrix1, matrix2,…
MSAVE name1 / MAT = name2

Examples

Example 1
Entering Data and Defining Matrices

Creating a Matrix

You can directly type a matrix or open a SYSTAT file as a matrix. Use the MAT command to create a matrix.

The input is:

MAT name = expression

where name is the name of the matrix and expression is either the entries of a matrix or an algebraic expression involving matrices, operators, and functions.

Typing a Matrix

To enter a matrix, type the entries separated by commas or spaces, with an open bracket before the first entry and a closed bracket after the last entry. If you are using the dialog box, then there is no need to type brackets. Separate each row with a semicolon, if you type several rows on one line. Use ‘Enter’ to type the next row in the dialog box. You can type each row on a separate line:
To enter a missing numeric value, type a period. To enter a missing character value, type a space surrounded by quotation marks.

**Character Values**

Most of the discussion in this chapter involves matrices with only numeric entries. A matrix can contain columns of character data. However, you cannot use most of the Matrix functions and operators on such matrices.

**Naming Rows and Columns**

If you save a matrix to a file, the column names are considered variable names, and the row names are saved as a variable named `ROWNAME$`, added as the last column.

**Use Column of a File to Label Rows**

You can name the rows of a matrix with the entries in a column while you read a data file as matrix. This is useful for matrices with a large number of rows. Suppose that the names of the students are stored in the `NAME$` column of the data file `GRADES`. You could name the rows of `GRADES` using `NAME$`.

The input is:

```
USE GRADES/MAT = GRADES ROWNAME = name$
```
**Requesting a Directory of Matrices**

You can get a description of all the matrices known to Matrix. As an example, we request descriptions of matrices $A$, $B$, and $C$. If you do not specify a matrix name, SYSTAT displays description for all matrices known to Matrix.

The input is:

```
MAT a = [2 4 ; 3 1 ; 1 5]
MAT b = [2 3 1 ; 4 1 5]
USE LONGLEY / MAT = c
DIR a  b  c
```

The output is:

```
Name of the Matrix | a
Number of Rows     | 3
Row Names          | R_1, R_2, R_3
Number of Columns  | 2
Column Names       | C_1, C_2

Name of the Matrix | b
Number of Rows     | 2
Row Names          | R_1, R_2
Number of Columns  | 3
Column Names       | C_1, C_2 , C_3

Name of the Matrix | c
Number of Rows     | 16
Row Names          | R_1, R_2, R_3, R_4, R_5, R_6, R_7, R..
Number of Columns  | 7
Column Names       | DEFLATOR, GNP, UNEMPLOY, ARMFORCE, POPU..
```

**The Active Matrix**

The Directory report shown above shows that the matrices $A$, $B$, and $C$ are known to the Matrix procedure. The distinction between active and known matrices is important for the MDELETE, Rows and Columns, and MSELECT options of Row/Column Operations and also for MLET statements because they operate on the last matrix used—that is, the active matrix. Issue a SHOW command to make some known matrix active. Other commands and functions include an argument that identifies the input matrix.
Generating Matrices

Matrix can generate an identity matrix of rank \( n \) or an \( n \times p \) matrix with a user-specified number. Matrix can also generate five types of design variables for categorical variables; this is discussed in Example 7: “Design Variables” on page 291.

Diagonal Matrix from a Vector

Use the GAID (DIAG spelled backwards) command to get a diagonal matrix from the specified vector. As an example, we request a diagonal matrix from the vector \( V \) and store the result as \( V_{\text{DIAGMAT}} \).

The input is:

\[
\begin{align*}
\text{MAT } v &= [2 \ 4 \ 6] \\
\text{MAT } v_{\text{diagmat}} &= \text{GAID}(v) \\
\text{SHOW } v_{\text{diagmat}}
\end{align*}
\]

The output is:

Matrix Name: \( v_{\text{diagmat}} \)

\[
\begin{array}{ccc}
| & c_1 & c_2 & c_3 \\
\hline
R_1 & 2 & 0 & 0 \\
R_2 & 0 & 4 & 0 \\
R_3 & 0 & 0 & 6 \\
\end{array}
\]

Example 2
Element-by-Element Operations and Functions

For element-by-element operations, you can think of each matrix entry as a separate entity. SYSTAT provides a variety of such operators and functions. For the examples, we use small matrices with integer entries so that you can easily follow the calculations.

Arithmetical Operations

You can add or subtract matrices with the same dimensions (number of rows and columns) by adding or subtracting corresponding elements. Use the operator + for addition and - for subtraction. Matrix performs several variations of matrix addition
and subtraction. In addition to adding and subtracting two matrices with the same dimensions element-by-element, you can:

- Add (subtract) the same number to (from) each element of a matrix.
- Add (subtract) a row vector to (from) each row of a matrix.
- Add (subtract) a column vector to (from) each column of a matrix.

As an example, we add and subtract matrices \( \mathbf{M1}, \mathbf{M2} \) and \( \mathbf{M3} \) from the matrix \( \mathbf{A} \) and store the results as \( \mathbf{SUM1}, \mathbf{SUM2}, \mathbf{SUM3}, \mathbf{DIFFRNCE1}, \mathbf{DIFFRNCE2}, \) and \( \mathbf{DIFFRNCE3} \) respectively.

The input is:

\[
\begin{align*}
\text{MAT } a &= \begin{bmatrix} 1 & 2 & 3; 4 & 5 & 6 \end{bmatrix} \\
\text{MAT } m1 &= [3] \\
\text{MAT } m2 &= [3; 2 1] \\
\text{MAT } m3 &= [3; 2] \\
\text{MAT } sum1 &= a + m1 \\
\text{MAT } sum2 &= a + m2 \\
\text{MAT } sum3 &= a + m3 \\
\text{MAT } dffrnce1 &= a-m1 \\
\text{MAT } dffrnce2 &= a-m2 \\
\text{MAT } dffrnce3 &= a-m3 \\
\text{SHOW } sum1 \ sum2 \ sum3 \ dffrnce1 \ dffrnce2 \ dffrnce3
\end{align*}
\]

The output is:

Mat Name: sum1

\[
\begin{array}{ccc}
 & \text{C}_1 & \text{C}_2 & \text{C}_3 \\
\hline \\
\text{R}_1 & 4 & 5 & 6 \\
\text{R}_2 & 7 & 8 & 9 \\
\end{array}
\]

The remaining output is not listed.

**Element-by-Element Multiplication and Division**

Use \# and / operators for element-by-element multiplication and division.

*Note:* Do not confuse element-by-element multiplication with matrix multiplication. Matrix multiplication is discussed in Example 4: “Matrix Algebra” on page 282.

In addition to scalar multiplication and division, the \# and / have other uses. You can:

- Multiply or divide two matrices of the same size element-by-element.
- Multiply or divide each row of a matrix element-by-element by a row matrix.
- Multiply or divide each column of a matrix element-by-element by a column matrix.

As an example, we request multiplication and division of the matrix $A$ by $S_1$, $S_2$ and $S_3$ and store the results as $\text{PROD1}$, $\text{PROD2}$, $\text{PROD3}$, $\text{QUOT1}$, $\text{QUOT2}$, and $\text{QUOT3}$ respectively.

The input is:

```plaintext
MAT a = [2 4 6; 8 10 12]
MAT s1 = [1 2 3; 4 5 6]
MAT s2 = [4; 2]
MAT s3 = [1 2 3]
MAT prod1 = a # s1
MAT prod2 = a # s2
MAT prod3 = a # s3
MAT quot1 = a/s1
MAT quot2 = a/s2
MAT quot3 = a/s3
SHOW prod1 prod2 prod3 quot1 quot2 quot3
```

The output is:

```
Matrix Name: prod1
            | C_1  C_2  C_3
-----+----------------
R_1  |  2    8   18
R_2  | 32    50  72
```

The remaining output is not listed.

**Raising Each Element to a Power**

The power operator (##) can be used in several different ways. You can:
- Raise each element of a matrix to the same power.
- Raise each element of a matrix to the power of the corresponding element of a second matrix with the same dimensions.
- Raise the elements of each row of a matrix to the powers of the corresponding elements of a row matrix.
- Raise the elements of each column of a matrix to the powers of the corresponding elements of a column matrix.
As an example, we use ## operator on the matrix \( A \) to raise powers \( P_1, P_2, P_3, \) and \( P_4 \) and store the results as HASHHASH1, HASHHASH2, HASHHASH3, and HASHHASH4.

The input is:

\[
\begin{align*}
\text{MAT } a & = \begin{bmatrix} 2 & 4; 3 & 1; 1 & 5 \end{bmatrix} \\
\text{MAT } p1 & = \begin{bmatrix} 3 & 2; 3 & 7; 5 & 0 \end{bmatrix} \\
\text{MAT } p2 & = \begin{bmatrix} 4; 3; 2 \end{bmatrix} \\
\text{MAT } p3 & = \begin{bmatrix} 3 & 2 \end{bmatrix} \\
\text{MAT } p4 & = 3 \\
\text{MAT } \text{hashhash1} & = a ## p1 \\
\text{MAT } \text{hashhash2} & = a ## p2 \\
\text{MAT } \text{hashhash3} & = a ## p3 \\
\text{MAT } \text{hashhash4} & = a ## p4 \\
\end{align*}
\]

The output is:

Matrix Name: hashhash1

\[
\begin{array}{ll}
\text{R}_1 & | 8 \ 16 \\
\text{R}_2 & | 27 \ 1 \\
\text{R}_3 & | 1 \ 1
\end{array}
\]

The remaining output is not listed.

**Transforming Matrices**

You can use SYSTAT to transform a matrix element by element. Matrix provides many of the functions available for single variables. As an example, we request a square root transform on the matrix \( K \) and store the result as SQRT.

The input is:

\[
\begin{align*}
\text{MAT } k & = \begin{bmatrix} 49 & 9 & 16; 4 & 0 & 36; 24 & 4 & 1 \end{bmatrix} \\
\text{MAT } \text{sqrt} & = \text{SQR}(k) \\
\end{align*}
\]

The output is:

Matrix Name: sqrt

\[
\begin{array}{llll}
\text{R}_1 & | 7.000 & 3.000 & 4.000 \\
\text{R}_2 & | 2.000 & 0.000 & 6.000 \\
\text{R}_3 & | 4.899 & 2.000 & 1.000
\end{array}
\]
Relational/Logical Operators

Matrix in SYSTAT provides relational operators to compare matrices with the same dimension element by element. The result is a matrix containing only 0's and 1's--0's in positions where the relation is false, and 1's where the relation is true. As an example, we compare the matrices \( A \) and \( B \) and store the result as \( C \).

The input is:

\[
\begin{align*}
\text{MAT } a &= [2 \ 4 \ 6 \ 2; \ 1 \ 1 \ 3 \ 7; \ 5 \ 3 \ 0 \ 6; \ 3 \ 6 \ 1 \ 3] \\
\text{MAT } b &= [3 \ 5 \ 3 \ 2; \ 3 \ 0 \ 3 \ 5; \ 9 \ 1 \ 3 \ 6; \ 4 \ 1 \ 0 \ 4] \\
\text{MAT } c &= a < b \\
\text{SHOW } c
\end{align*}
\]

The output is:

\[
\begin{array}{cccc}
| & \text{C}_1 & \text{C}_2 & \text{C}_3 & \text{C}_4 \\
\hline
\text{R}_1 & 1 & 1 & 0 & 0 \\
\text{R}_2 & 1 & 0 & 0 & 0 \\
\text{R}_3 & 1 & 0 & 1 & 0 \\
\text{R}_4 & 1 & 0 & 0 & 1 \\
\end{array}
\]

The Matrix procedure has logical AND, OR, and NOT operators. The matrices that result from these operators are matrices of 0's (false values) and 1's (true values).

- **Logical AND.** Use AND operator to compare two matrices with the same dimensions element-by-element. The resulting matrix has 1's in positions where both the original matrices have nonzero values and 0's elsewhere.

- **Logical OR.** Use OR operator to compare two matrices with the same dimensions element-by-element. The resulting matrix has 1's in positions where at least one of the original matrices has a nonzero value, and 0's elsewhere.

- **Logical negation.** Use NOT to negate the elements of a matrix. It operates on a single matrix. The resulting matrix gets a 1 wherever the original matrix is 0, and a 0 wherever the original matrix is nonzero.

- **Exclusive OR.** You can use a combination of operators to form an exclusive OR. The resulting matrix gets a true value (1) only in positions where one matrix or the other has a true value; if both matrices have a true value, the result is false. To get \( A \) EXCLUSIVE OR \( B \), request \((a \ \text{AND} \ (\text{NOT} \ b)) \ \text{OR} \ ((\text{NOT} \ a) \ \text{AND} \ b)\).
As an example, we compare the matrices $A$, $B$ and $C$ with various combinations of logical operators and store the results as $R_1$, $R_2$, $R_3$, $R_4$, $R_5$, $R_6$, and $R_7$.

The input is:

\[
\begin{align*}
\text{MAT } a &= [1 \ 0 \ 0 \ 1; 2 \ 1 \ 0 \ 1; 0 \ 1 \ 0 \ 0] \\
\text{MAT } b &= [0 \ 1 \ 0 \ 1; 1 \ 1 \ 0 \ 0; 0 \ 1 \ 0 \ 3] \\
\text{MAT } c &= [1 \ 1 \ 0 \ 1; 0 \ 0 \ 1 \ 1; 1 \ 1 \ 1 \ 0] \\
\text{MAT } r_1 &= a \text{ AND } b \\
\text{MAT } r_2 &= \text{ NOT } a \\
\text{MAT } r_3 &= a \text{ OR } b \\
\text{MAT } r_4 &= \text{ NOT } a \text{ OR } \text{ NOT } b \\
\text{MAT } r_5 &= (a \text{ AND } b) \text{ OR } c \\
\text{MAT } r_6 &= b \text{ AND } (\text{ NOT}(a \text{ OR } c)) \\
\text{MAT } r_7 &= (a \text{ AND } (\text{ NOT } b)) \text{ OR } ((\text{ NOT } a) \text{ AND } b)
\end{align*}
\]

SHOW $r_1$ $r_2$ $r_3$ $r_4$ $r_5$ $r_6$ $r_7$

The output is:

<table>
<thead>
<tr>
<th>Matrix Name: $r_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>; C_1</td>
</tr>
<tr>
<td>---------</td>
</tr>
<tr>
<td>R_1</td>
</tr>
<tr>
<td>R_2</td>
</tr>
<tr>
<td>R_3</td>
</tr>
</tbody>
</table>

The remaining output is not listed.

### Example 3

**Manipulating Matrices**

**Reordering Rows and Columns**

Use ROWSORT and COLSORT functions of Manipulations to reorder the rows and columns of your matrix. As an example, we request to reorder rows and columns of the matrix $N$.

The input is:

\[
\begin{align*}
\text{MAT } n &= [2 \ 4 \ 6 \ 2; 1 \ 1 \ 3 \ 7; 5 \ 3 \ 0 \ 6; 3 \ 6 \ 1 \ 3] \\
\text{MAT } \text{neword} &= \text{ROWSORT}(n, 3) \\
\text{MAT } \text{neword2} &= \text{COLSORT}(n, 4) \\
\text{SHOW } \text{neword} \text{ neword2}
\end{align*}
\]
The output is:

**Matrix Name: neword**

<table>
<thead>
<tr>
<th></th>
<th>C_1</th>
<th>C_2</th>
<th>C_3</th>
<th>C_4</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_1</td>
<td>5</td>
<td>3</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>R_2</td>
<td>3</td>
<td>6</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>R_3</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>R_4</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>2</td>
</tr>
</tbody>
</table>

**Matrix Name: neword2**

<table>
<thead>
<tr>
<th></th>
<th>C_1</th>
<th>C_2</th>
<th>C_3</th>
<th>C_4</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_1</td>
<td>6</td>
<td>2</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>R_2</td>
<td>3</td>
<td>1</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>R_3</td>
<td>0</td>
<td>5</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>R_4</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>6</td>
</tr>
</tbody>
</table>

*Fold*

The FOLD function of Manipulations copies the lower triangular portion of a matrix to the portion of the matrix above the diagonal so that the matrix is symmetric. As an example, we fold the matrix **SQRMAT** and store the result as **FOLDED**.

The input is:

```matlab
MAT sqrmat = [1  2  3  4  5; 6  7  8  9  10; 11 12 13 14, 15; 16 17 18 19 20; 21 22 23 24 25]
MAT folded = FOLD(sqrmat)
SHOW sqrmat folded
```

The output is:

**Matrix Name: sqrmat**

<table>
<thead>
<tr>
<th></th>
<th>C_1</th>
<th>C_2</th>
<th>C_3</th>
<th>C_4</th>
<th>C_5</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>R_2</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>R_3</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
</tr>
<tr>
<td>R_4</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
</tr>
<tr>
<td>R_5</td>
<td>21</td>
<td>22</td>
<td>23</td>
<td>24</td>
<td>25</td>
</tr>
</tbody>
</table>

**Matrix Name: folded**

<table>
<thead>
<tr>
<th></th>
<th>C_1</th>
<th>C_2</th>
<th>C_3</th>
<th>C_4</th>
<th>C_5</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_1</td>
<td>1</td>
<td>6</td>
<td>11</td>
<td>16</td>
<td>21</td>
</tr>
<tr>
<td>R_2</td>
<td>6</td>
<td>7</td>
<td>12</td>
<td>17</td>
<td>22</td>
</tr>
<tr>
<td>R_3</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>18</td>
<td>23</td>
</tr>
<tr>
<td>R_4</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>24</td>
</tr>
<tr>
<td>R_5</td>
<td>21</td>
<td>22</td>
<td>23</td>
<td>24</td>
<td>25</td>
</tr>
</tbody>
</table>
Reconfiguring a Matrix

You can change the dimensions of a matrix with the SHAPE function of Manipulations. SHAPE fills in the new matrix with the values of the original matrix, starting with the element in the first row and first column and working from left to right across the rows. SHAPE uses only as many elements as it needs. For example, if the original matrix is 4 x 5 and you specify dimensions 3 x 3 for the new matrix, SHAPE uses only the first nine elements for the original. Conversely, if the original matrix does not contain enough elements to fill the new matrix, SHAPE fills the extra positions with missing values.

Filling in Missing Values

You can use the FILL function of Manipulations to replace the missing values in your matrix with specified values. As an example, we fill the missing values of the matrix A with zeros and store the result as NEWA.

The input is:

```plaintext
MAT a = [1  2  3 .; 4  5  6  7; 8  9 . .; 10 . . .]  
MAT newa = FILL(a, 0)  
SHOW a newa
```

The output is:

```
Matrix Name: a

| C_1 | C_2 | C_3 | C_4 |
----+-----+-----+-----+
R_1 | 1   | 2   | 3   |
R_2 | 4   | 5   | 6   | 7   |
R_3 | 8   | 9   | .   | .   |
R_4 | 10  | .   | .   | .   |

Matrix Name: newa

| C_1 | C_2 | C_3 | C_4 |
----+-----+-----+-----+
R_1 | 1   | 2   | 3   | 0   |
R_2 | 4   | 5   | 6   | 7   |
R_3 | 8   | 9   | 0   | 0   |
R_4 | 10  | 0   | 0   | 0   |
```
**Determining the Dimensions of a Matrix**

You can use DIM, NROW, and NCOL functions of Manipulations for determining the number of rows and columns of a matrix. The NROW and NCOL functions return the number of rows and columns of a specified matrix. The DIM function returns both the number of rows and the number of columns as a 2 x 1 matrix.

**Changing a Matrix to a Vector and Vice Versa**

The STRING function of Manipulations writes the lower-triangular portion of a matrix as a vector. The GNIRTS (STRING spelled backwards) function writes a row or column vector to the lower-triangular portion of a matrix.

**Concatenating Matrices**

You can concatenate matrices side-by-side, end-to-end, or corner-to-corner. As an example, we request each type of concatenation from matrices A, B, C, and D.

The input is:

```
MAT a = [1; 0; 4]
MAT b = [1; 3; 5; 7]
MAT c = [2 6 8 9]
MAT d = [4 3 3; 2 3 1]
MAT concatenate = ((a || b) // c) || d
SHOW concatenate
```

The output is:

```
|   | C_1 | C_1 | C_1 | C_2 | C_3 | C_6 | C_7 |
|---+-----+-----+-----+-----+-----+-----+-----|
R_1 | 1   | 1   | .   | .   | .   | .   | .   |
R_2 | 0   | 3   | .   | .   | .   | .   | .   |
R_3 | 4   | 5   | .   | .   | .   | .   | .   |
R_4 | .   | 7   | .   | .   | .   | .   | .   |
R_5 | 2   | 6   | 8   | 9   | .   | .   | .   |
R_1 | .   | .   | .   | .   | 4   | 3   | 3   |
R_2 | .   | .   | .   | .   | 2   | 3   | 1   |
```

**Determining Whether Two Matrices are Equal**

The EQUAL function of Manipulations compares two matrices element-by-element and determines whether they are equal. The EQUAL function results in a scalar---a 1 if the matrices are equal or a 0 if they are not equal.
**Note:** Do not confuse the `EQUAL` function with the `==` operator. The `EQUAL` function results in a scalar Z (0 or 1). The `==` operator results in a matrix of 1’s (where corresponding elements are equal) and 0’s (where corresponding elements are unequal). As an example, we request a test of whether the matrices \( A \) and \( B \) are equal or not.

The input is:

```plaintext
MAT a = [2 3 4; 3 4 2; 1 2 4]
MAT b = [2 3 4; 3 4 2; 1 2 5]
MAT eq = EQUAL(a, b)
SHOW eq
```

The output is:

```
Matrix Name: eq

| c_1 |     |
|-----|--|--|
| R_1 | 0 |
```

**Example 4**

**Matrix Algebra**

You can use SYSTAT to perform a variety of algebraic operations on matrices. Some of them are discussed here.

**Raising a Matrix to a Power**

Use `**` (or `^`) operator to raise a matrix to a power. As an example, we request the cube of a matrix \( A \) and store the result as a matrix named `A_CUBED`.

The input is:

```plaintext
MAT a = [2 1 3; 0 2 3; 5 3 2]
MAT a_cubed = a ** 3
```

Note that only square matrices can be raised to a power.

**Kronecker Product**

The result of the Kronecker product of an $n \times p$ matrix $A$ and an $r \times s$ matrix $B$, contains a $r \times s$ sub-matrix for each element of $A$ (that is, it has $nr$ rows and $ps$ columns). Each $r \times s$ sub-matrix is the product of the corresponding element in $A$ with the matrix $B$. As an example, we request Kronecker product of matrices $A$ and $B$ to produce the matrix $C$.

The input is:

```
MAT a = [1  2; 3  4; 5  6]
MAT b = [7  8; 9  10]
MAT c = KRON(a,b)
SHOW c
```

The output is:

Matrix Name: c

<table>
<thead>
<tr>
<th></th>
<th>C_1</th>
<th>C_2</th>
<th>C_3</th>
<th>C_4</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_1</td>
<td>7</td>
<td>8</td>
<td>14</td>
<td>16</td>
</tr>
<tr>
<td>R_2</td>
<td>9</td>
<td>10</td>
<td>18</td>
<td>20</td>
</tr>
<tr>
<td>R_3</td>
<td>21</td>
<td>24</td>
<td>28</td>
<td>32</td>
</tr>
<tr>
<td>R_4</td>
<td>27</td>
<td>30</td>
<td>36</td>
<td>40</td>
</tr>
<tr>
<td>R_5</td>
<td>35</td>
<td>40</td>
<td>42</td>
<td>48</td>
</tr>
<tr>
<td>R_6</td>
<td>45</td>
<td>50</td>
<td>54</td>
<td>60</td>
</tr>
</tbody>
</table>

**Eigenvalues**

When you multiply a matrix by a column matrix (or vector), the result is another column matrix of the same dimension. Thus, a matrix is a rotation of vectors. A non-zero vector $v$ is called an *eigenvector* of a matrix $A$ if there exists a scalar $s$ such that:

$$Av = sv$$

A scalar $s$ is called an *eigenvalue* of $A$ if there exists a non-zero vector $v$ such that:

$$Av = sv$$

Use the EIGVAL function to compute the eigenvalues of a matrix.
**Cholesky Decomposition**

The Cholesky decomposition is often used to generate data with a specific correlation structure or to standardize correlated data (for example, to compute Mahalanobis distances). The Matrix procedure provides two forms of Cholesky decomposition. The first form is discussed here, the other version of Cholesky decomposition is discussed in the Example 6: “Matrix Decomposition” on page 288.

Use the CHOL function of Matrix Algebra to get the Cholesky decomposition of a symmetric matrix. If you start with a matrix $R$, the Cholesky decomposition finds $L$ such that

$$R = L \cdot L'$$

where $L$ has zeros above the diagonal. Since $L$ is triangular, it is much easier to invert than $R$. Note that:

$$R^{-1} = (L \cdot L')^{-1} = (L')^{-1} \cdot (L)^{-1} = (L^{-1})' \cdot L^{-1}$$

As an example, we create the matrix $SS$ using the SYSTAT data file $RANSAMPLE$ and request its Cholesky decomposition.

The input is:

```
USE RANSAMPLE / MAT = h
MAT ss = SSCP (h)
MAT my_L_mat = CHOL(ss)
SHOW ss my_L_mat
```

The output is:

**Matrix Name: ss**

```
<table>
<thead>
<tr>
<th></th>
<th>C_1</th>
<th>C_2</th>
<th>C_3</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_1</td>
<td>91.586</td>
<td>105.759</td>
<td>211.001</td>
</tr>
<tr>
<td>R_2</td>
<td>105.759</td>
<td>202.348</td>
<td>322.297</td>
</tr>
<tr>
<td>R_3</td>
<td>211.001</td>
<td>322.297</td>
<td>665.796</td>
</tr>
</tbody>
</table>
```

**Matrix Name: my_L_mat**

```
<table>
<thead>
<tr>
<th></th>
<th>C_1</th>
<th>C_2</th>
<th>C_3</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_1</td>
<td>9.570</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>R_2</td>
<td>11.051</td>
<td>8.957</td>
<td>0.000</td>
</tr>
<tr>
<td>R_3</td>
<td>22.048</td>
<td>8.780</td>
<td>10.128</td>
</tr>
</tbody>
</table>
```
Example 5
Statistical Functions

We use the GRADES data file to demonstrate some of the statistical functions. First we use some column functions.

The input is:

```
USE GRADES/MAT = grades
ROWNAME grades = Mark Cindy Jeff Greg Michele Nicky
MAT count = COLNUM (grades)
MAT test_avg = COLMEAN (grades)
MAT testhigh = COLMAX (grades)
MAT test_low = COLMIN (grades)
SHOW count test_avg testhigh test_low
```

The output is:

**Matrix Name: count**

```
<table>
<thead>
<tr>
<th>C_1</th>
<th>C_2</th>
<th>C_3</th>
<th>C_4</th>
<th>C_5</th>
<th>C_6</th>
</tr>
</thead>
<tbody>
<tr>
<td>COLNUM</td>
<td>6.000</td>
<td>5.000</td>
<td>6.000</td>
<td>5.000</td>
<td>6.000</td>
</tr>
</tbody>
</table>
```

**Matrix Name: test_avg**

```
<table>
<thead>
<tr>
<th>C_1</th>
<th>C_2</th>
<th>C_3</th>
<th>C_4</th>
<th>C_5</th>
<th>C_6</th>
</tr>
</thead>
<tbody>
<tr>
<td>COLMEAN</td>
<td>78.833</td>
<td>84.000</td>
<td>83.167</td>
<td>90.200</td>
<td>83.833</td>
</tr>
</tbody>
</table>
```

**Matrix Name: testhigh**

```
<table>
<thead>
<tr>
<th>C_1</th>
<th>C_2</th>
<th>C_3</th>
<th>C_4</th>
<th>C_5</th>
<th>C_6</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAXIMUM</td>
<td>97.000</td>
<td>90.000</td>
<td>95.000</td>
<td>100.000</td>
<td>95.000</td>
</tr>
</tbody>
</table>
```

**Matrix Name: test_low**

```
<table>
<thead>
<tr>
<th>C_1</th>
<th>C_2</th>
<th>C_3</th>
<th>C_4</th>
<th>C_5</th>
<th>C_6</th>
</tr>
</thead>
<tbody>
<tr>
<td>MINIMUM</td>
<td>59.000</td>
<td>77.000</td>
<td>73.000</td>
<td>81.000</td>
<td>64.000</td>
</tr>
</tbody>
</table>
```

The resulting matrices have an entry for each column of the original matrix. The COUNT matrix indicates that there is a missing score for both the second and third quiz. The TEST_AVG matrix shows that the lowest average score occurred on the first quiz, and the highest average score was for the final. TESTHIGH and TEST_LOW show the highest and lowest scores on each test.
Now let us try some row functions and concatenate the resulting columns (matrices):

\[
\begin{align*}
\text{MAT} & \quad \text{count} = \text{ROWNUM}(\text{grades}) \\
\text{MAT} & \quad \text{stu\_avg} = \text{ROWMEAN}(\text{grades}) \\
\text{MAT} & \quad \text{stu\_high} = \text{ROWMAX}(\text{grades}) \\
\text{MAT} & \quad \text{stu\_low} = \text{ROWMIN}(\text{grades}) \\
\text{MAT} & \quad \text{results} = \text{count} \ || \ \text{stu\_avg} \ || \ \text{stu\_high} \ || \ \text{stu\_low} \\
\text{ROWNAME} & \quad \text{results} = \text{Mark Cindy Jeff Greg Michele Nicky} \\
\text{COLNAME} & \quad \text{results}=\text{count stu\_avg stu\_high stu\_low} \\
\text{SHOW} & \quad \text{results}
\end{align*}
\]

The output is:

**Matrix Name: RESULTS**

<table>
<thead>
<tr>
<th></th>
<th>count</th>
<th>stu_avg</th>
<th>stu_high</th>
<th>stu_low</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mark</td>
<td>6.000</td>
<td>88.833</td>
<td>95.000</td>
<td>77.000</td>
</tr>
<tr>
<td>Cindy</td>
<td>6.000</td>
<td>94.000</td>
<td>100.000</td>
<td>85.000</td>
</tr>
<tr>
<td>Jeff</td>
<td>5.000</td>
<td>70.400</td>
<td>81.000</td>
<td>59.000</td>
</tr>
<tr>
<td>Greg</td>
<td>6.000</td>
<td>85.167</td>
<td>92.000</td>
<td>78.000</td>
</tr>
<tr>
<td>Michele</td>
<td>5.000</td>
<td>81.800</td>
<td>95.000</td>
<td>67.000</td>
</tr>
<tr>
<td>Nicky</td>
<td>6.000</td>
<td>87.333</td>
<td>93.000</td>
<td>82.000</td>
</tr>
</tbody>
</table>

The output provides information about rows (students). The first result (non-missing) is the number of tests taken by each student. The second is the average of the non-missing test scores for each student. The third and fourth panels give each student’s high and low grade, respectively. Each student’s name was specified using the function ROWNAME. Matrix knows the contents of each column, and it assigns the respective name.

**Correlation, Covariance, and SSCP Matrices**

SYSTAT has functions for computing covariance, correlation and cross product of deviations (SSCP) matrices in Statistical functions of Matrix Operations. Given an \( n \times p \) matrix.

\[
X = \begin{bmatrix}
    x_{11} & x_{12} & \cdots & x_{1p} \\
    x_{21} & x_{22} & \cdots & x_{2p} \\
    \vdots & \vdots & \ddots & \vdots \\
    x_{n1} & x_{n2} & \cdots & x_{np}
\end{bmatrix}
\]
the cross product of deviations, covariance, and correlation matrices of $X$ are defined, respectively, as:

$$
A = \begin{bmatrix}
a_{11} & a_{12} & \cdots & a_{1p} \\
a_{21} & a_{22} & \cdots & a_{2p} \\
\vdots & \vdots & \ddots & \vdots \\
a_{p1} & a_{p2} & \cdots & a_{pp}
\end{bmatrix},

S = \begin{bmatrix}
s_{11} & s_{12} & \cdots & s_{1p} \\
s_{21} & s_{22} & \cdots & s_{2p} \\
\vdots & \vdots & \ddots & \vdots \\
s_{p1} & s_{p2} & \cdots & s_{pp}
\end{bmatrix},

R = \begin{bmatrix}
r_{11} & r_{12} & \cdots & r_{1p} \\
r_{21} & r_{22} & \cdots & r_{23} \\
\vdots & \vdots & \ddots & \vdots \\
r_{p1} & r_{p2} & \cdots & r_{pp}
\end{bmatrix}
$$

for all $i, j = 1, \ldots, p$:

$$
a_{ij} = \sum_{k=1}^{n} \left( x_{ki} - \bar{x}_i \right) \left( x_{kj} - \bar{x}_j \right)

s_{ij} = a_{ij} / (n-1)

r_{ij} = \frac{s_{ij}}{\sqrt{s_{ii} \cdot s_{jj}}}
$$

**Correlations for Sets of Variables**

As an example, let us look at correlations between all pairs of variables in the LONGLEY data (TIME is omitted).

The input is:

```
USE LONGLEY/ MAT = longley
MAT xvars = longley (;DEFLATOR .. POPULATN)
MAT corrx = CORR(xvars)
SHOW corrx
```

The output is:

**Matrix Name: corrx**

<table>
<thead>
<tr>
<th></th>
<th>C_1</th>
<th>C_2</th>
<th>C_3</th>
<th>C_4</th>
<th>C_5</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_1</td>
<td>1.000</td>
<td>0.992</td>
<td>0.621</td>
<td>0.465</td>
<td>0.979</td>
</tr>
<tr>
<td>R_2</td>
<td>0.992</td>
<td>1.000</td>
<td>0.604</td>
<td>0.446</td>
<td>0.991</td>
</tr>
<tr>
<td>R_3</td>
<td>0.621</td>
<td>0.604</td>
<td>1.000</td>
<td>-0.177</td>
<td>0.687</td>
</tr>
<tr>
<td>R_4</td>
<td>0.465</td>
<td>0.446</td>
<td>-0.177</td>
<td>1.000</td>
<td>0.364</td>
</tr>
<tr>
<td>R_5</td>
<td>0.979</td>
<td>0.991</td>
<td>0.687</td>
<td>0.364</td>
<td>1.000</td>
</tr>
</tbody>
</table>
Chapter 8

Correlations for Two Sets of Variables

When there are many variables, the correlations matrix can be very large and hard to view—especially if you are interested only in correlations among one group of variables against another. Here, define DEFLATOR and GNP as one set and UNEMPLOY, ARMFORCE, and POPULATN as a second set.

```plaintext
USE LONGLEY / MAT = longley
MAT set_a=longley(; DEFLATOR, GNP)
MAT set_b=longley(; UNEMPLOY .. POPULATN)
MAT sda=SQR(DIAG(SSCP(set_a)))
MAT sdb=SQR(DIAG(SSCP(set_b)))
MAT corr_ab=TRP((&set_a-COLMEAN(set_a))/sda)
  *(set_b-, COLMEAN(set_b))/sdb)
SHOW corr_ab
```

The output is:

```
Matrix Name: corr_ab

<table>
<thead>
<tr>
<th>UNEMPLOY</th>
<th>ARMFORCE</th>
<th>POPULATN</th>
</tr>
</thead>
<tbody>
<tr>
<td>deflator</td>
<td>0.621</td>
<td>0.465</td>
</tr>
<tr>
<td>gnp</td>
<td>0.604</td>
<td>0.446</td>
</tr>
</tbody>
</table>
```

Example 6
Matrix Decomposition

SYSTAT provides Cholesky, QR, and Singular value decomposition options. It also computes eigenvectors.

**Eigenvectors**

Use the EIGEN function of the Matrix decomposition to get eigenvalues as well as eigenvectors. As an example, we compute the eigenvectors and eigenvalues of the matrix A and store the results as EIGVALS and EIGVECT respectively.

The input is:

```plaintext
MAT b = [2 1 5 3; 1 1 3 6; 5 3 0 1; 3 6 1 3]
MAT a = b+TRP(b)
CALL EIGEN (eigvals, eigvect, a)
FORMAT 3
SHOW eigvals eigvect
```
The output is:

Matrix Name: eigvals

<table>
<thead>
<tr>
<th></th>
<th>C_1</th>
<th>C_2</th>
<th>C_3</th>
<th>C_4</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_1</td>
<td>22.397</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>R_2</td>
<td>0.000</td>
<td>5.979</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>R_3</td>
<td>0.000</td>
<td>0.000</td>
<td>-4.155</td>
<td>0.000</td>
</tr>
<tr>
<td>R_4</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>-12.221</td>
</tr>
</tbody>
</table>

Matrix Name: eigvect

<table>
<thead>
<tr>
<th></th>
<th>C_1</th>
<th>C_2</th>
<th>C_3</th>
<th>C_4</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_1</td>
<td>0.468</td>
<td>0.620</td>
<td>0.451</td>
<td>0.440</td>
</tr>
<tr>
<td>R_2</td>
<td>0.515</td>
<td>-0.390</td>
<td>-0.532</td>
<td>0.548</td>
</tr>
<tr>
<td>R_3</td>
<td>0.400</td>
<td>0.486</td>
<td>-0.543</td>
<td>-0.556</td>
</tr>
<tr>
<td>R_4</td>
<td>0.597</td>
<td>-0.476</td>
<td>0.469</td>
<td>-0.445</td>
</tr>
</tbody>
</table>

**Cholesky**

The alternative version of the Cholesky decomposition is:

\[ R = L^* \cdot D \cdot L^* ' \]

where \( L^* \) has zeros above the diagonal and \( D \) is 0 everywhere but the diagonal. Some prefer this decomposition to the one shown in the Example 4: “Matrix Algebra” on page 282 because square roots are not computed. Note that in the previous form

\[ R = L \cdot L' \]

implies that

\[ L = L^* \cdot \sqrt{D} \]

**QR**

For an \( n \times p \) matrix \( X \), the QR decomposition is

\[ X = Q \cdot R \]

where \( Q \) is an \( n \times n \) matrix and \( R \) is an \( n \times p \) matrix such that:

\[ Q' \cdot Q = I \quad \text{and} \quad Q \cdot Q' = I \]

The elements of \( R \) are 0 below the diagonal, i.e., \( r_{jk} = 0 \) if \( j > k \).
This decomposition can make regression computations easier. Note that
\[ Y = X \beta + \varepsilon \]
\[ Q' \cdot Y = R \beta + \varepsilon \]
Therefore, by starting with this structure, the computation is easy via a “back solution”.

**Singular Value**

Both the QR and SVD (singular value decomposition) methods can be used to find eigenvalues and eigenvectors. For an \( n \times p \) matrix \( X \), the SVD decomposition is
\[ X = U \cdot D \cdot V \]
where \( U \) is an \( n \times n \) orthogonal matrix, \( D \) is an \( n \times p \) matrix that is 0 except for its diagonal, and \( V \) is a \( p \times p \) orthogonal matrix. Here the diagonal elements of \( D \) are \( d_{11}, d_{22}, \ldots d_{pp} \) when \( p \) is less than or equal to \( n \), and \( d_{11}, d_{22}, \ldots d_{nn} \) when \( n \) is less than \( p \). In addition:
\[ U \cdot U' = I_n \]
\[ V \cdot V' = I_p \]
As an example, we request Cholesky and QR decompositions of the matrix \( SS \) and store the results as \( MY_D_MAT, MY_LSTAR, MY_Q_MAT, \) and \( MY_R_MAT \) respectively.

The input is:

```plaintext
USE RANSAMPLE / MAT = h
MAT ss = SSCP(h)
CALL CHOL (my_D_mat, my_Lstar, ss)
CALL QRD (my_Q_mat, my_R_mat, ss)
SHOW my_D_mat my_Lstar my_Q_mat my_R_mat
```

Use SV option of the Matrix decomposition to get singular value decomposition. As an example, we request the Singular value decomposition of the matrix \( A \) and store the results as \( MY_U_MAT, MY_D_MAT, MY_V_MAT \) and \( A \) respectively.

The input is:

```plaintext
MAT a = [1 2; 3 4; 5 6]
CALL SVD (my_U_mat, my_D_mat, my_V_mat, a)
SHOW my_U_mat my_D_mat my_V_mat a
```
Example 7
Design Variables

If you are using Matrix to request an analysis of variance or if you need to generate a set of design variables for each categorical variable in a regression model, use DESIGN0, DESIGN1, DESIGNF, ORTHEQ, or ORTHUN functions of Design type. To generate design variables, the cases in your data file need no special ordering. For example, if you have a data file named MYDATA with the variables CITY$ (Los Angeles, Chicago, and New York) and DOSE$ (zero, low, medium, and high) and want to generate the “1, 0, −1” type variables for CITY$ and the orthogonal coefficients for DOSE$, use DESIGN1 and ORTHEQ functions to get the desired results.

The input is:

```matlab
MAT mydata = ['Los Angeles'; Chicago; 'New York'] || [zero; low; medium; high]
COLNAME mydata = city$ dose$
MAT my_design = DESIGN1 (mydata (1 .. 3; city$)) || ,
    ORTHEQ (mydata (; dose$))
SHOW my_design
```

The DESIGN1 function generates two design variables (A and B) for values of CITY$; ORTHEQ generates three design variables (C, D, and E) for levels of DOSE$.

The following are the values generated for specific cities and doses:

<table>
<thead>
<tr>
<th>CITY</th>
<th>A</th>
<th>B</th>
<th>DOSE</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>LA</td>
<td>1</td>
<td>0</td>
<td>zero</td>
<td>−3</td>
<td>1</td>
<td>−1</td>
</tr>
<tr>
<td>Chicago</td>
<td>0</td>
<td>1</td>
<td>low</td>
<td>−1</td>
<td>−1</td>
<td>3</td>
</tr>
<tr>
<td>New York</td>
<td>−1</td>
<td>−1</td>
<td>medium</td>
<td>1</td>
<td>−1</td>
<td>−3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>high</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Actually, the values of the orthogonal components are not quite what was stated here. SYSTAT normalizes each component by dividing by the square root of the sum of the squares of the coefficients (for example, for the design variable C, the value for “zero dose” is

\[
- \frac{3}{\sqrt{9 + 1 + 1 + 9}}
\]

or −0.671). The following are the actual values:
Use of ORTHUN in Regression Analysis

The generated observations are stored in the file named **REGORTHO**. The aim is to explore the polynomial regression

\[ Y = a + bx + cx^2 + dx^3 \]

to determine whether the terms with higher degrees are necessary or redundant.

The input is:

```plaintext
USE REGORTHO / MAT = d
Mat x = d (;1)
MAT y = d (;2)
MAT q = ORTHUN (x, x)
MAT z = TRP (y) * q
MAT s = z # z
MAT ssub = s (1; 1, 2, 3)
SHOW ssub
```

The output is:

**Matrix Name: ssub**

<table>
<thead>
<tr>
<th></th>
<th>C_1</th>
<th>C_2</th>
<th>C_3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>1671.255</td>
<td>33.294</td>
<td>4.099</td>
</tr>
</tbody>
</table>

The three terms, from right to left, in ssub are the sum of squares corresponding to the cubic, quadratic and linear regression coefficients.

```plaintext
MAT ss = SSCP(y)
SHOW ss
```

The output is:

**Matrix Name: ss**

<table>
<thead>
<tr>
<th></th>
<th>C_1</th>
</tr>
</thead>
<tbody>
<tr>
<td>r_1</td>
<td>1727.983</td>
</tr>
</tbody>
</table>
Here \textbf{SS} gives the total sum of squares (adjusted for the constant). Now the significance can be successively tested using F-tests to decide whether to retain the higher order terms, and thus a decision regarding the appropriate degree for the polynomial regression can be taken. (Indeed one finds that the 24 elements of the matrix S is a breakup of the total sum of squares into 24 components, corresponding to the coefficients in the polynomial equation of degree 24.) One has to take care that while using the \texttt{ORTHUN} function all the elements of the vector \textit{x} have to be distinct with the first element equal to one. Also if there are too many elements in vector \textit{x}, SYSTAT may find the problem difficult to handle. That is why a vector \texttt{xre} was created from the vector \textit{x}.

\textbf{Example 8} \\
\textbf{Packing Two Records into One}

Sometimes, you may have to restructure your data for a statistical analysis. For example, suppose you record the age and blood cholesterol levels for two groups of women. Women in the first group use contraceptive pills; women in the second group do not. The data in the \textit{CHOLESTEROL} file look like this:

\begin{tabular}{ccc}
\textbf{PILL} & \textbf{AGE} & \textbf{CHOL} \\
1 & 25 & 200 \\
2 & 25 & 211 \\
1 & 33 & 230 \\
2 & 33 & 243 \\
1 & 19 & 180 \\
2 & 19 & 215 \\
1 & 39 & 215 \\
2 & 39 & 175 \\
1 & 28 & 189 \\
2 & 28 & 163 \\
1 & 20 & 179 \\
2 & 20 & 175 \\
1 & 35 & 300 \\
2 & 35 & 224 \\
\end{tabular}

A \textit{PILL} value of 1 indicates that the woman takes the pill; a value of 2 indicates that she does not. You want to use a matched pairs \textit{t}-test to test the hypothesis that the mean difference in blood cholesterol between women who take the pill and women who do not is 0. First, using the \texttt{SHAPE} function, match a woman who takes the pill with a woman of the same age who does not:

\texttt{MAT matched = SHAPE(cholesterol,7,6)}
or

\[
\text{MAT matched} = \text{SHAPE(cholesterol, NROW(cholesterol)/2, NCOL(cholesterol)*2)}
\]

Now the data file looks like this:

<table>
<thead>
<tr>
<th>1</th>
<th>25</th>
<th>200</th>
<th>2</th>
<th>25</th>
<th>211</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>33</td>
<td>230</td>
<td>2</td>
<td>33</td>
<td>243</td>
</tr>
<tr>
<td>1</td>
<td>19</td>
<td>180</td>
<td>2</td>
<td>19</td>
<td>215</td>
</tr>
<tr>
<td>1</td>
<td>39</td>
<td>215</td>
<td>2</td>
<td>39</td>
<td>175</td>
</tr>
<tr>
<td>1</td>
<td>28</td>
<td>189</td>
<td>2</td>
<td>28</td>
<td>163</td>
</tr>
<tr>
<td>1</td>
<td>20</td>
<td>179</td>
<td>2</td>
<td>20</td>
<td>175</td>
</tr>
<tr>
<td>1</td>
<td>35</td>
<td>300</td>
<td>2</td>
<td>35</td>
<td>224</td>
</tr>
</tbody>
</table>

Each case has the cholesterol value for a pill user and for her age-matched control. You can drop all of the columns except the two that contain cholesterol and rename them \textit{P\_CHOL} and \textit{NOP\_CHOL}, respectively:

\[
\text{MDELETE COLUMNS} = 1, 2, 4, 5
\]
\[
\text{COLNAME matched} = \text{P\_CHOL}, \text{NOP\_CHOL}
\]
\[
\text{MSAVE matched}
\]

Now you can request the matched pairs \textit{t}-test:

\[
\text{TESTING}
\]
\[
\text{USE MATCHED}
\]
\[
\text{TTEST P\_CHOL, NOP\_CHOL}
\]

\section*{Example 9
\textbf{Writing a Correlation Matrix as a Vector}}

As an example of the \textit{STRING} function, we examine correlations among 16 variables from the \textit{USSTATES} file. These variables include death rates from nine causes, crime rates, median household income and other economic indicators, weather information, and health care data.

You can compute the correlation matrix in Matrix or Correlations. We computed the correlation matrix, named it \textit{USCORR}, and then used the \textit{STRING} function to write the correlations as a vector.

The following are the variable names:

\begin{tabular}{cccccccc}
\text{ACCIDENT} & \text{CARDIO} & \text{CANCER} & \text{PULMONAR} & \text{PNEU\_FLU} & \text{DIABETES} \\
\text{LIVER} & \text{VIOLRATE} & \text{PROPRATE} & \text{AVGPAY} & \text{TEACHERS} & \text{TCHRSAL} \\
\text{MARRIAGE} & \text{DIVORCE} & \text{HOSPITAL} & \text{DOCTOR}
\end{tabular}
You can compute the correlation matrix in Matrix or Correlations. We computed the correlation matrix, named it *USCORR*, and then used the STRING function to write the correlations as a vector.

The input is:

```plaintext
USE USCORR / MAT = uscorr  
MAT corrstr = STRING (uscorr,0)  
MSAVE CORRSTR  
FORMAT 6 2  
SHOW uscorr corrstr  
FORMAT
```

The output is:

```plaintext
<table>
<thead>
<tr>
<th>ACCIDENT</th>
<th>CARDIO</th>
<th>CANCER</th>
<th>PULMONAR</th>
<th>PNEU_FLU</th>
<th>DIABETES</th>
<th>LIVER</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_1</td>
<td>1.00</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>R_2</td>
<td>-0.15</td>
<td>1.00</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>R_3</td>
<td>-0.17</td>
<td>0.89</td>
<td>1.00</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>R_4</td>
<td>0.24</td>
<td>0.12</td>
<td>0.29</td>
<td>1.00</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>R_5</td>
<td>-0.16</td>
<td>0.47</td>
<td>0.33</td>
<td>0.24</td>
<td>1.00</td>
<td>.</td>
</tr>
<tr>
<td>R_6</td>
<td>0.03</td>
<td>0.53</td>
<td>0.63</td>
<td>0.13</td>
<td>-0.06</td>
<td>1.00</td>
</tr>
<tr>
<td>R_7</td>
<td>-0.16</td>
<td>0.05</td>
<td>0.27</td>
<td>0.21</td>
<td>-0.31</td>
<td>0.15</td>
</tr>
<tr>
<td>R_8</td>
<td>0.00</td>
<td>0.06</td>
<td>0.10</td>
<td>-0.24</td>
<td>-0.23</td>
<td>-0.14</td>
</tr>
<tr>
<td>R_9</td>
<td>-0.01</td>
<td>-0.37</td>
<td>-0.25</td>
<td>-0.11</td>
<td>-0.43</td>
<td>-0.34</td>
</tr>
<tr>
<td>R_10</td>
<td>-0.60</td>
<td>0.09</td>
<td>0.16</td>
<td>-0.30</td>
<td>-0.09</td>
<td>-0.05</td>
</tr>
<tr>
<td>R_11</td>
<td>0.24</td>
<td>-0.08</td>
<td>-0.22</td>
<td>0.07</td>
<td>0.26</td>
<td>0.03</td>
</tr>
<tr>
<td>R_12</td>
<td>-0.62</td>
<td>0.08</td>
<td>0.24</td>
<td>-0.22</td>
<td>-0.11</td>
<td>0.07</td>
</tr>
<tr>
<td>R_13</td>
<td>0.08</td>
<td>-0.14</td>
<td>-0.03</td>
<td>0.38</td>
<td>-0.19</td>
<td>-0.24</td>
</tr>
<tr>
<td>R_14</td>
<td>0.47</td>
<td>-0.27</td>
<td>-0.17</td>
<td>0.53</td>
<td>-0.25</td>
<td>-0.24</td>
</tr>
<tr>
<td>R_15</td>
<td>0.23</td>
<td>-0.05</td>
<td>-0.17</td>
<td>0.20</td>
<td>0.37</td>
<td>-0.12</td>
</tr>
<tr>
<td>R_16</td>
<td>-0.65</td>
<td>0.14</td>
<td>0.25</td>
<td>-0.23</td>
<td>0.06</td>
<td>0.06</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>VIOLRATE</th>
<th>PROPRATE</th>
<th>AVGPAY</th>
<th>TEACHERS</th>
<th>TCHRSAL</th>
<th>MARRIAGE</th>
<th>DIVORCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_1</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>R_2</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>R_3</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>R_4</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>R_5</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>R_6</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>R_7</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>R_8</td>
<td>1.00</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>R_9</td>
<td>0.68</td>
<td>1.00</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>R_10</td>
<td>0.57</td>
<td>0.34</td>
<td>1.00</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>R_11</td>
<td>-0.58</td>
<td>-0.43</td>
<td>-0.47</td>
<td>1.00</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>R_12</td>
<td>0.49</td>
<td>0.14</td>
<td>-0.01</td>
<td>-0.26</td>
<td>0.00</td>
<td>1.00</td>
</tr>
<tr>
<td>R_13</td>
<td>0.09</td>
<td>0.14</td>
<td>-0.01</td>
<td>-0.26</td>
<td>0.00</td>
<td>1.00</td>
</tr>
<tr>
<td>R_14</td>
<td>0.06</td>
<td>0.29</td>
<td>-0.29</td>
<td>-0.12</td>
<td>-0.36</td>
<td>0.73</td>
</tr>
<tr>
<td>R_15</td>
<td>-0.60</td>
<td>-0.48</td>
<td>-0.70</td>
<td>0.69</td>
<td>-0.65</td>
<td>-0.07</td>
</tr>
<tr>
<td>R_16</td>
<td>0.44</td>
<td>0.21</td>
<td>0.77</td>
<td>-0.37</td>
<td>0.81</td>
<td>-0.14</td>
</tr>
</tbody>
</table>
```
The following is a stem-and-leaf diagram of the 210 correlations:

```
USE CORRSTR
CLSTEM LTMATRIX
```
Let us use a SPLOM (scatterplot matrix) with 90\% confidence ellipses to examine the bivariate distributions of the variables with sizable correlations (–0.74, 0.80, 0.82, 0.91, 0.91, and 0.94).

USE USSTATES
SPLOM CANCER CARDIO MSTROKE FSTROKE,AVGPAY TCHRSAL INCOME,
POVRTY91 / HALF ELL = .99
The output is:

Five of the positive correlations are much larger than the others. Two of these involve death rates due to C\textit{ANCER}, C\textit{ARDIO}, M\textit{STROKE} (for males), and F\textit{STROKE} (for females). The other three involve economic indicators: A\textit{VGPAY} (average pay), T\textit{CHRSAL} (average salary of teachers), and I\textit{NCOME} (median family income). One negative correlation, PO\textit{VRTY91} (percentage of the population living below the poverty level) and I\textit{NCOME}, is fairly large.

\textbf{Example 10}
\textbf{Ridge Regression}

Ridge regression is a technique that tames the estimates of the regression coefficients when severe multicollinearity exists among the independent variables. The formula for the estimated coefficients is

$$\hat{\beta} = (Z' Z + \lambda I)^{-1} Z' y$$
where $Z$ is the matrix of standardized independent variables, $y$ is the vector of the standardized dependent variable, and $I$ is the identity matrix. The usual least-squares estimate is obtained when $\lambda$ is 0. Before selecting a value for $\lambda$, researchers usually try several values and plot the resulting estimates to get an indication of their stability. Then, they choose a value of $\lambda$ for which the coefficients smooth out and no longer make sudden changes.

The input is:

```plaintext
USE CEMENT / MAT = cement
FORMAT 11 7
MAT y = cement(;HEAT)
MAT x = cement(;INGREDIENT1 .. INGREDIENT4)
MAT y = COLZSC(y)
MAT x = COLZSC(x)
MAT cpdx = SSCP(x)
MAT beta = INV(cpdx) * TRP(x) * y
SHOW beta
```

The output is:

```
Matrix Name: beta

<table>
<thead>
<tr>
<th></th>
<th>C_1</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_1</td>
<td>0.6065120</td>
</tr>
<tr>
<td>R_2</td>
<td>0.5277056</td>
</tr>
<tr>
<td>R_3</td>
<td>0.0433897</td>
</tr>
<tr>
<td>R_4</td>
<td>-0.1602874</td>
</tr>
</tbody>
</table>
```

The input is:

```plaintext
MAT ridge = GAID(DIAG(cpdx))
MAT betar = INV(cpdx + ridge # 0.01) * TRP(x) * y
MAT beta = beta || betar
MAT betar = INV(cpdx + ridge # 0.02) * TRP(x) * y
MAT beta = beta || betar
MAT betar = INV(cpdx + ridge # 0.03) * TRP(x) * y
MAT beta = beta || betar
MAT betar = INV(cpdx + ridge # 0.04) * TRP(x) * y
MAT beta = beta || betar
MAT betar = INV(cpdx + ridge # 0.05) * TRP(x) * y
MAT beta = beta || betar
MAT betar = INV(cpdx + ridge # 0.06) * TRP(x) * y
MAT beta = beta || betar
MAT betar = INV(cpdx + ridge # 0.07) * TRP(x) * y
MAT beta = beta || betar
MAT betar = INV(cpdx + ridge # 0.08) * TRP(x) * y
MAT beta = beta || betar
MAT betar = INV(cpdx + ridge # 0.09) * TRP(x) * y
MAT beta = beta || betar
MAT beta = TRP(beta)
```
Chapter 8

The output is:

Matrix Name: beta

<table>
<thead>
<tr>
<th>R_1   R_2   R_3   R_4 ... C_1</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_1  0.607  0.514  0.497  0.485 ... 0.000</td>
</tr>
<tr>
<td>R_2  0.528  0.317  0.303  0.300 ... 0.010</td>
</tr>
<tr>
<td>R_3  0.043 -0.055 -0.069 -0.079 ... -0.020</td>
</tr>
<tr>
<td>R_4 -0.160 -0.382 -0.394 -0.396 ... -0.030</td>
</tr>
<tr>
<td>R_5  0.000  0.010  0.020  0.030 ... 0.040</td>
</tr>
<tr>
<td>R_6  0.514  0.317 -0.055 -0.382 ... 0.050</td>
</tr>
<tr>
<td>R_7  0.497  0.303 -0.069 -0.394 ... 0.060</td>
</tr>
<tr>
<td>R_8  0.485  0.300 -0.079 -0.396 ... 0.070</td>
</tr>
<tr>
<td>R_9  0.475  0.299 -0.086 -0.396 ... 0.080</td>
</tr>
<tr>
<td>R_10 0.466  0.298 -0.093 -0.394 ... 0.090</td>
</tr>
</tbody>
</table>

Plotting Ridge Estimates

The following is a plot of the results:

USE BETA1
CUT ROWNAME$
BEGIN
PLOT INGREDIENT1..INGREDIENT4 * RIDGE / OVERLAY LINE XMIN = -.01, XMAX = .1 YMIN = -1.1 YMAX = 1.5 XLABEL = 'Ridge Factor', YLABEL = 'Estimate of Beta' LEGEND = NONE
PLOT INGREDIENT1..INGREDIENT4 * RIDGE / OVERLAY SMOOTH = SPLINE, YMIN = -1.1 YMAX = 1.5 XPOW XLABEL = 'Ridge Factor', YLABEL = 'Estimate of Beta' LOC = 6IN, 0IN, LEGEND =-2.5IN, -2.3IN
END
In the plot on the right, we added XPOW and SMOOTH = SPLINE.

**Example 11**

**The Sweep Function**

The SWEEP function is extremely useful in many statistical procedures. To operate the SWEEP function the underlying matrix has to be a square matrix and zero should not appear as the pivoting or sweeping element anywhere during the operation. If pivoting is done to some elements, the result is a partially inverted matrix. The vector in the SWEEP argument contains 0 or 1 for each element of the diagonal of the input matrix, where '0' indicates no pivoting and '1' indicates pivoting. As an example, we use the sweep function on matrix $P$ and store the result as **MY_SWEEP**.

The input is:

```
MAT p = [2 1 1; 1 2 1; 2 3 4]
MAT my_sweep = SWEEP(p, [0 1 0])
SHOW my_sweep
```

The output is:

```
Matrix Name: my_sweep

<table>
<thead>
<tr>
<th></th>
<th>C_1</th>
<th>C_2</th>
<th>C_3</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_1</td>
<td>1.500</td>
<td>-0.500</td>
<td>0.500</td>
</tr>
<tr>
<td>R_2</td>
<td>0.500</td>
<td>0.500</td>
<td>0.500</td>
</tr>
<tr>
<td>R_3</td>
<td>0.500</td>
<td>-1.500</td>
<td>2.500</td>
</tr>
</tbody>
</table>
```
The SWEEP function has transformed $P$ into $\text{MY\_SWEEP}$ by using the indicated diagonal elements for sweeping. The vector in the above argument is $[0 \ 1 \ 0]$, the SWEEP function pivots the second diagonal element of the input matrix $P$.

SWEEP can find the inverse. As an example, we invert the matrix $K$ and store the result as $K\_\text{INV}$.

The input is:

\[
\text{MAT } k = \begin{bmatrix} 1 & 2 & 3; & 2 & 3 & 4; & 3 & 4 & 6 \end{bmatrix}
\]

\[
\text{MAT } k\_\text{inv} = \text{SWEEP}(k, \begin{bmatrix} 1 & 1 & 1 \end{bmatrix})
\]

\[
\text{SHOW } k\_\text{inv}
\]

The output is:

Matrix Name: $k\_\text{inv}$

<table>
<thead>
<tr>
<th></th>
<th>C.1</th>
<th>C.2</th>
<th>C.3</th>
</tr>
</thead>
<tbody>
<tr>
<td>R.1</td>
<td>-2.000</td>
<td>0.000</td>
<td>1.000</td>
</tr>
<tr>
<td>R.2</td>
<td>0.000</td>
<td>3.000</td>
<td>-2.000</td>
</tr>
<tr>
<td>R.3</td>
<td>1.000</td>
<td>-2.000</td>
<td>1.000</td>
</tr>
</tbody>
</table>

In general if,

\[
W = \begin{bmatrix} A & B \\ C & D \end{bmatrix}
\]

an $n \times n$ matrix with $A$ as a sub-matrix of order $m \times m$ then

\[
S = \text{SWEEP } (W, \begin{bmatrix} 1 & . & . & 0 & . & . \end{bmatrix})
\]

with $m$ 1's and $n-m$ 0's in the places as indicated, the SWEEP function changes $W$ to the $n \times n$ matrix $S$ given by

\[
S = \begin{bmatrix} A^{-1} & A^{-1}B \\ -CA^{-1} & D - CA^{-1}B \end{bmatrix}
\]

The SWEEP operator is the workhorse of computational statistics. Now let us see some applications of SWEEP in statistical analysis.

From a computational standpoint, the SWEEP function is useful in getting regression coefficients in the model

\[
Y = X\beta + \varepsilon
\]
Consider the matrix

\[ \mathbf{W} = \begin{bmatrix} \mathbf{X}'\mathbf{X} & \mathbf{X}'\mathbf{Y} \\ \mathbf{Y}'\mathbf{X} & \mathbf{Y}'\mathbf{Y} \end{bmatrix} \]

where \( \mathbf{X} \) is an \( n \times m \) design matrix.

A single application of SWEEP on matrix \( \mathbf{W} \) permits the simultaneous computation of \( \hat{\beta} \) and error sum of squares (ESS).

\[ \mathbf{S} = \begin{bmatrix} \mathbf{(X'X)^{-1}} & \hat{\beta} \\ -\hat{\beta}' & \text{ESS} \end{bmatrix} \]

where \( \hat{\beta} = \mathbf{(X'X)^{-1}}\mathbf{X}'\mathbf{Y} \) and \( \text{ESS} = \mathbf{Y}'\mathbf{Y} - \mathbf{Y}'\mathbf{X}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{Y}. \)

Let us demonstrate some uses of SWEEP through examples.

**Evaluating a Quadratic Form**

We are often required to carry out computations involving expressions of the form:

\[ (\bar{x} - \bar{y})' A^{-1} (\bar{x} - \bar{y}) \]

Such computations, for example, are required while computing Hotelling’s \( T^2 \) statistics in multivariate analysis or sums of squares in hypothesis testing problems.

Consider

\[ \mathbf{W} = \begin{bmatrix} A & \mathbf{x} - \mathbf{y} \\ \mathbf{x} - \mathbf{y} & 0 \end{bmatrix} \]

an \( n \times n \) matrix, and use the SWEEP function with choice of the vector being \( [1 \ 1 \ 1 \ \ldots \ 1\ 0] \), 0 only in the last place. The element in the last row, last column of the resulting matrix will be

\[ -(\bar{x} - \bar{y})' A^{-1} (\bar{x} - \bar{y}) \]
giving us the value of the required quadratic form with an altered sign. For example,

```plaintext
MAT A = [1 1 1; 1 2 1; 1 1 2]
MAT b = [2; 3; 4]
MAT w = A || b // (trp(b) || [0])
MAT s = SWEEP(w, [1 1 1 0])
MAT q = - s(4; 4)
SHOW q
```

The output is:

```
Matrix Name: q

| R_4 |
-----|------|
R_4  | 9.000 |
-----|------|
```

\( Q \) gives the value of \( b' A^{-1} b \).

This procedure can be used to evaluate a quantity like the one above, needed to compute Hotelling's \( T^2 \).

### Solving the System

\[ W x = 0 \]

where \( W \) is \( n \times n \) singular matrix with \( \text{rank}(W) = n-1 \).

Assuming that the \((n-1) \times (n-1)\) sub-matrix of \( W \) lying in the upper left hand corner is non-singular and writing \( W \) as

\[
W = \begin{bmatrix}
A & b \\
c' & d
\end{bmatrix}
\]

notice that

\[
\begin{bmatrix}
A^{-1} b \\
-1
\end{bmatrix}
\]

is a solution which can be obtained through the SWEEP function.

Now the above can be used to obtain the stationary distribution of a Markov chain.

For a Markov chain with a one-step transition probability matrix \( P \), a stationary distribution is given by a vector \( \mu \) which satisfies

\[ \mu' P = \mu' \]
with all elements of \( u \) non-negative, and adding up to one. For an irreducible aperiodic Markov chain, \( u \) is unique. Using SYSTAT matrix commands, one can find this unique stationary distribution.

As an example, we request the stationary distribution of a Markov chain having a transition probability matrix \( P \) and store the result as \( u \).

The input is:

\[
\begin{align*}
\text{MAT } p &= \begin{bmatrix}
0.1, & 0.2, & 0.2, & 0.1, & 0.4; \\
0.1, & 0.2, & 0.3, & 0.4, & 0; \\
0, & 0.1, & 0.2, & 0.3, & 0.4; \\
0.4, & 0.1, & 0.1, & 0, & 0.4; \\
0.2, & 0.2, & 0.2, & 0.2, & 0.2
\end{bmatrix} \\
\text{MAT } j &= \text{I}(5) \\
\text{MAT } w &= j - p \\
\text{MAT } s &= \text{SWEEP}(w, \begin{bmatrix} 1 & 1 & 1 & 1 & 0 \end{bmatrix}) \\
\text{MAT } c &= s \begin{bmatrix} 5; 1, 2, 3, 4 \end{bmatrix} || \begin{bmatrix} 1 \end{bmatrix} \\
\text{MAT } c\_\text{sum} &= \text{ROWSUM}(c) \\
\text{MAT } u &= c/c\_\text{sum} \\
\text{SHOW } u
\end{align*}
\]

The output is:

Matrix Name: u

<table>
<thead>
<tr>
<th>C_1</th>
<th>C_2</th>
<th>C_3</th>
<th>C_4</th>
<th>C_5</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_1</td>
<td>0.167</td>
<td>0.161</td>
<td>0.196</td>
<td>0.196</td>
</tr>
</tbody>
</table>

The matrix (vector) \( U \) obtained is the stationary distribution.

**Solving Two Systems of Equations**

Consider the problem of obtaining the vectors \( x \) and \( y \) satisfying the equations

\[
ax = b \text{ and } a^\prime y = c,
\]

where \( a \) is a non-singular matrix. Let

\[
S = \begin{bmatrix}
a & b \\
-c' & 0
\end{bmatrix}
\]
Chapter 8

To illustrate this, consider the following equations:

\[ x + y + z = 4 \]
\[ x - y + z = 2 \]
\[ x - y - z = 0 \]

and, simultaneously, also the system:

\[ u + v + w = 10 \]
\[ u - v - w = 2 \]
\[ u + v - w = 6 \]

The input is:

\[
\begin{bmatrix}
1 & 1 & 1 & 4; & 1 & -1 & 1 & 2; & 1 & -1 & -1 & 0; & -10 & -2 & -6 & 0
\end{bmatrix}
\]

\[
\text{MAT sol} = \text{Sweep}(s, \begin{bmatrix} 1 & 1 & 1 & 0 \end{bmatrix})
\]

\[
\text{MAT sol1} = \text{sol}(1, 2, 3 ; 4)
\]

\[
\text{MAT sol2} = \text{sol}(4; 1 2 3)
\]

\[
\text{SHOW sol sol1}
\]

\[
\text{MAT sol21} = \text{TRP(sol2) '}
\]

\[
\text{SHOW sol21}
\]

The output is:

**Matrix Name: sol**

<table>
<thead>
<tr>
<th></th>
<th>C_1</th>
<th>C_2</th>
<th>C_3</th>
<th>C_4</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_1</td>
<td>0.500</td>
<td>0.000</td>
<td>0.500</td>
<td>2.000</td>
</tr>
<tr>
<td>R_2</td>
<td>0.500</td>
<td>-0.500</td>
<td>0.000</td>
<td>1.000</td>
</tr>
<tr>
<td>R_3</td>
<td>0.000</td>
<td>0.500</td>
<td>-0.500</td>
<td>1.000</td>
</tr>
<tr>
<td>R_4</td>
<td>6.000</td>
<td>2.000</td>
<td>2.000</td>
<td>28.000</td>
</tr>
</tbody>
</table>

**Matrix Name: sol1**

<table>
<thead>
<tr>
<th></th>
<th>C_4</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_1</td>
<td>2.000</td>
</tr>
<tr>
<td>R_2</td>
<td>1.000</td>
</tr>
<tr>
<td>R_3</td>
<td>1.000</td>
</tr>
</tbody>
</table>

**Matrix Name: sol21**

<table>
<thead>
<tr>
<th></th>
<th>C_4</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_1</td>
<td>6.000</td>
</tr>
<tr>
<td>C_2</td>
<td>2.000</td>
</tr>
<tr>
<td>C_3</td>
<td>2.000</td>
</tr>
</tbody>
</table>

**SOL1** and **SOL21** are the solutions to the above two systems of equations.
Finding Multiple Correlation Coefficient

The example below demonstrates how the SWEEP command can be used to find the multiple correlation coefficient. You can find the multiple correlation coefficient between X and \((Y \ Z)\) as follows.

The input is:

```
USE AKIMA / MAT = h
MAT r = CORR(h)
MAT w = SWEEP(r, [0 1 1])
MAT w1 = w(1; 1)
MAT mult_r = SQR(1 - w1)
SHOW mult_r
```

The output is:

```
Matrix Name: mult_r

| C_1 |
---|---|
R_1 | 0.877 |
```

Example 12
Multiple Regression and Canonical Correlation

This example illustrates how Matrix in SYSTAT can be used to compute canonical correlation coefficients and multiple regressions. For this the IRIS data set is used. This data set consists of observations on sepal length, sepal width, petal length and petal width for three species, with 50 observations on each species.

The input is:

```
USE IRIS / MAT = iris
MAT rr = CORR(iris)
MAT r = rr(2 .. 5; 2 .. 5)
SHOW r
MAT r11 = r(1 2; 1 2)
MAT r12 = r(1 2; 3 4)
MAT r21 = TRP(r12)
MAT r22 = r(3 4; 3 4)
MAT l = CHOL(r11)
MAT c = INV(l)*r12*INV(r22)*r21*TRP(INV(l))
MAT cc = fold(c)
```
Although apparently the Matrix $C$ looks symmetric because of approximations involved in the computing process, it no longer remains exactly symmetric. Therefore the FOLD command is used.

It is to be noted that since SYSTAT computes characteristic roots of only symmetric matrices, $r_{11}$ is decomposed using the CHOL function. The characteristic roots of the $C$ matrix and those of $(r_{11})^{-1}(r_{12})(r_{22})^{-1}(r_{21})$ are the same and are squares of canonical correlation coefficients. Characteristic roots are obtained using the EIGVAL function.

```
MAT d = EIGVAL(cc)
SHOW d
```

The output is:

Matrix Name: $C$

<table>
<thead>
<tr>
<th></th>
<th>C_2</th>
<th>C_3</th>
<th>C_4</th>
<th>C_5</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_2</td>
<td>1.000</td>
<td>-0.118</td>
<td>0.872</td>
<td>0.818</td>
</tr>
<tr>
<td>R_3</td>
<td>-0.118</td>
<td>1.000</td>
<td>-0.428</td>
<td>-0.366</td>
</tr>
<tr>
<td>R_4</td>
<td>0.872</td>
<td>-0.428</td>
<td>1.000</td>
<td>0.963</td>
</tr>
<tr>
<td>R_5</td>
<td>0.818</td>
<td>-0.366</td>
<td>0.963</td>
<td>1.000</td>
</tr>
</tbody>
</table>

These are the two canonical correlation coefficients between the two sets of variables (sepal length, sepal width) and (petal length, petal width).

Now the coefficients of regression of (sepal length, sepal width) on (petal length, petal width) are to be found. To remove the first column 'species' from the IRIS data, the MDELETE function is used:

```
USE IRIS / MAT = iris
MDELETE COLUMNS=1
MAT ss = SSCP(iris)
SHOW ss
```
The output is:

Matrix Name: **ss**

<table>
<thead>
<tr>
<th></th>
<th>C_1</th>
<th>C_2</th>
<th>C_3</th>
<th>C_4</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_1</td>
<td>102.168</td>
<td>-6.323</td>
<td>189.873</td>
<td>76.924</td>
</tr>
<tr>
<td>R_2</td>
<td>-6.323</td>
<td>28.307</td>
<td>-49.119</td>
<td>-18.124</td>
</tr>
<tr>
<td>R_3</td>
<td>189.873</td>
<td>-49.119</td>
<td>464.325</td>
<td>193.046</td>
</tr>
<tr>
<td>R_4</td>
<td>76.924</td>
<td>-18.124</td>
<td>193.046</td>
<td>86.570</td>
</tr>
</tbody>
</table>

The input is:

```
MAT s11 = ss(1, 2; 1, 2)
MAT s12 = ss(1, 2; 3, 4)
MAT s21 = ss(3, 4; 1, 2)
MAT s22 = ss(3, 4; 3, 4)
MAT beta = s12*INV(s22)
MAT s11_2 = s11 - s12*INV(s22)*s21
MAT mean = COLMEAN(iris)
MAT c = TRP(mean)
MAT c1 = c(1, 2; 1)
MAT c2 = c(3, 4; 1)
MAT beta0 = c1 - beta * c2
MAT b = beta0 || beta
SHOW b
```

The output is:

Matrix Name: **b**

<table>
<thead>
<tr>
<th></th>
<th>R_1</th>
<th>C_1</th>
<th>C_2</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_1</td>
<td>4.191</td>
<td>0.542</td>
<td>-0.320</td>
</tr>
<tr>
<td>R_2</td>
<td>3.587</td>
<td>-0.257</td>
<td>0.364</td>
</tr>
</tbody>
</table>

Beta (b) is the matrix of regression coefficients of (sepal length, sepal width) on (petal length, petal width) and beta0 (b_0) is the constant coefficient vector.

Now the conditional covariance matrix of one set of variables given the other set assuming multivariate normality can be computed. In the following, the conditional variances of sepal length and sepal width on petal length, petal width is found for the IRIS data.

```
MAT condvar = s11_2 / 147
SHOW condvar
```

The output is:

Matrix Name: **condvar**

<table>
<thead>
<tr>
<th></th>
<th>C_1</th>
<th>C_2</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_1</td>
<td>0.162</td>
<td>0.099</td>
</tr>
<tr>
<td>R_2</td>
<td>0.099</td>
<td>0.152</td>
</tr>
</tbody>
</table>
This gives the conditional variances of (petal length, petal width) given (sepal length, sepal width).

**Example 13**

**Moore-Penrose Generalized Inverse**

The Moore-Penrose generalized inverse can be defined for any (even \( m \times n \) matrix, with numeric entries) matrix \( A \). This inverse is unique and it works like an inverse for solving many problems. It is denoted by \( A_\_ \) in SYSTAT. This inverse satisfies the following four conditions:

\[
A A_\_ A = A ,
A_\_ A A_\_ = A_\_
\]

\( A A_\_ \) and \( A_\_ A \) are symmetric.

SYSTAT functions available in the Matrix procedure are rich enough to yield the Moore-Penrose inverse. For any matrix \( A \), use the following commands to obtain the Moore-Penrose inverse:

\[
\begin{align*}
\text{MAT } & \text{aa = TRP(a)*a} \\
\text{MAT } & \text{c = M(NCOL(a), 1, 1)} \\
\text{MAT } & \text{e = I(NCOL(a))} \\
\text{MAT } & \text{null = M(NCOL(a), 1, 0.0000001)} \\
\text{MAT } & \text{k = EIGVAL(aa)} \\
\text{CALL } & \text{EIGEN(d, u, aa)} \\
\text{MAT } & \text{t = k > null} \\
\text{MAT } & \text{tt = c - 2 # t} \\
\text{MAT } & \text{k_ = k ## tt} \\
\text{MAT } & \text{d_ = e # k} \\
\text{MAT } & \text{a_ = u*d_*TRP(u)*TRP(a)}
\end{align*}
\]

\( A_\_ \) is the Moore-Penrose inverse of \( A \).

For a consistent system of linear equations

\[
Ax = b
\]

\( A_\_ b \) is a solution. Thus a solution of the system \( X'X = X'Y \) is given by \( (X'X)_X'Y \). You may note that this is equal to \( X_\_ Y \).
Consider the system of equations:

\[ \begin{align*}
    x + y + u + v &= 8 \\
    x + y - u - v &= 0 \\
    x - y + u - v &= 0
\end{align*} \]

The input is:

\[
\text{MAT } a = \begin{bmatrix} 1 & 1 & 1 & 1; 1 & 1 & -1 & -1; 1 & -1 & 1 & -1 \end{bmatrix} \\
\text{Mat } b = [8; 0; 0]
\]

After obtaining \( A_\) , the Moore-Penrose inverse of \( A \) using the above steps can be used to obtain a solution to the above system.

The input is:

\[
\text{MAT } \text{sol} = a_* b \\
\text{SHOW } a_\text{sol}^{-}
\]

The output is:

Matrix Name: \( a_\)

\[
\begin{array}{ccc}
   R_1 & R_2 & R_3 \\
  \hline
  R_1 & 0.250 & 0.250 & 0.250 \\
  R_2 & 0.250 & 0.250 & -0.250 \\
  R_3 & 0.250 & -0.250 & 0.250 \\
  R_4 & 0.250 & -0.250 & -0.250 \\
\end{array}
\]

Matrix Name: \( \text{sol} \)

\[
\begin{array}{c}
   C_1 \\
  \hline
  R_1 & 2.000 \\
  R_2 & 2.000 \\
  R_3 & 2.000 \\
  R_4 & 2.000 \\
\end{array}
\]

There are many solutions, e.g., \((4 \ 0 \ 0 \ 4)'\) is a solution. The above solution \( \text{SOL} \) is the one that corresponds to the Moore-Penrose inverse.
References


# Acronym & Abbreviation

## Expansions

<table>
<thead>
<tr>
<th>A</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABS - absolute value</td>
<td>C&amp;RT - classification and regression trees</td>
</tr>
<tr>
<td>ACF - autocorrelation function</td>
<td>CCF - cross-correlation function</td>
</tr>
<tr>
<td>ACT - actuarial life table</td>
<td>Cdf/CF - cumulative distribution function</td>
</tr>
<tr>
<td>AD test - Anderson Darling test</td>
<td>CFA - confirmatory factor analysis</td>
</tr>
<tr>
<td>AIC - Akaike information criterion</td>
<td>CGM - Computer graphics metafile: binary or</td>
</tr>
<tr>
<td>AID - automatic interaction detection</td>
<td>clear text</td>
</tr>
<tr>
<td>ALT - alternative</td>
<td>CI - confidence interval</td>
</tr>
<tr>
<td>ANCOVA - analysis of covariance</td>
<td>COL/col - column</td>
</tr>
<tr>
<td>ANOVA - analysis of variance</td>
<td>CONV - convergence</td>
</tr>
<tr>
<td>AR - autoregressive</td>
<td>COV - covariance</td>
</tr>
<tr>
<td>ARCH - Autoregressive Conditional</td>
<td>Cpk - process capability index</td>
</tr>
<tr>
<td>Heteroskedasticity</td>
<td>Cpk - Process capability index for off-centered</td>
</tr>
<tr>
<td>ARIMA - autoregressive integrated moving</td>
<td>process</td>
</tr>
<tr>
<td>average</td>
<td>CR - confidence region</td>
</tr>
<tr>
<td>ARL - average run length</td>
<td>CRN - Cauchy random number</td>
</tr>
<tr>
<td>ARMA - autoregressive moving average</td>
<td>CSV - comma separated values</td>
</tr>
<tr>
<td>ARS - adaptive rejection sampling</td>
<td>CV - coefficient of variation</td>
</tr>
<tr>
<td>ASCII - American Standard Code for</td>
<td>CVI - cross validation index</td>
</tr>
<tr>
<td>Information Interchange</td>
<td></td>
</tr>
<tr>
<td>ASE - asymptotic standard error</td>
<td></td>
</tr>
<tr>
<td>AVG - average</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>D</td>
</tr>
<tr>
<td>BC - Bray-Curtis similarity measure</td>
<td>DBF - Dbase files</td>
</tr>
<tr>
<td>BFGS - Broyden-Fletcher-Goldfarb-Shannon</td>
<td>df - degrees of freedom</td>
</tr>
<tr>
<td>BHHH - Berndt-Hall-Hall-Housman</td>
<td>DIM - dimension</td>
</tr>
<tr>
<td>BIC - Bayesian information criterion</td>
<td>DOS - disc operating system</td>
</tr>
<tr>
<td>BMP - Windows bitmap</td>
<td>DPMO - defects per million opportunities</td>
</tr>
<tr>
<td>BOOT - bootstrap</td>
<td>DPU - defects per unit</td>
</tr>
<tr>
<td></td>
<td>DTA - Stata files</td>
</tr>
</tbody>
</table>
Acronyms

DWASS - Dwass-Steel-Chritchlow-Fligner pairwise comparisons test  J  JB - Jarque-Bera
DWLS - distance weighted least-squares  JMP - JMP v3.2 data files
E  JPEG/JPG - joint photographic experts group
EM - expectation-maximization  K  K-M - Kaplan-Meier
EMF - Windows enhanced metafile  K-S test - Kolmogorov-Smirnov test
EWMA - exponentially weighted moving  KS1 - one sample Kolmogorov-Smirnov tests
average  KS2 - two sample Kolmogorov-Smirnov tests
F  L  LAD - least absolute deviations
G  LCL - lower control limit
GARCH - Generalized Autoregressive  LMS- least median of squares
Conditional Heteroskedasticity  LM Test - Lagrange Multiplier Tes
GG - Greenhouse Geisser  LR - likelihood ratio
GIF - Graphics Interchange Format  LRDEV - likelihood ratio of deviate
GLM - generalized linear models  LW - Lawless and Wang
GLS - generalized least-squares  M  MA - moving average
GMA - geometric moving average  MAD - mean absolute deviation
GN - Gauss-Newton method  MANCOVA - multivariate analysis of
covariance  MANOVA - multivariate analysis of variance
H  MAX - maximum
H & L - Hosmer and Lemeshow  MC Test - McLeod-Li Test
H-L trace - Holding-Lawley trace  MCMC - Markov Chain Monte Carlo
HTML - hyper text markup language  MDS - multidimensional scaling
I  MIN - minimum
IIDMC - independently and identically  M-H- Metropolis-Hastings
distributed Monte Carlo  ML - Maximum Likelihood
IMPSAMPI - importance sampling integration  MLA - maximum likelihood analysis
IMPSAMPR - importance sampling ratio  MLE - maximum likelihood estimate
IndMH - Independent Metropolis-Hastings  MML - maximum marginal likelihood
INDSCAL - individual differences scaling  MS - mean squares
INITSAMP - initial sample  MSE - mean square error
ITER - iterations
<table>
<thead>
<tr>
<th>Acronyms</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MTW - MINITAB v11 data files</td>
<td>SBC - Schwarz's Bayesian information criterion</td>
</tr>
<tr>
<td>MU2 - Guttman's mu2 monotonicity coefficients</td>
<td>SC - set correlation</td>
</tr>
<tr>
<td></td>
<td>SD - standard deviations</td>
</tr>
<tr>
<td>N</td>
<td>sd2/sas7bdat - SAS v9 files</td>
</tr>
<tr>
<td>NR - Newton-Raphson</td>
<td>SE/se/S.E. - standard error</td>
</tr>
<tr>
<td>O</td>
<td>SETCOR - Set and Canonical Correlations</td>
</tr>
<tr>
<td>OC - operating characteristic</td>
<td>SQL - structured query language</td>
</tr>
<tr>
<td>ODBC - open database capture and connectivity</td>
<td>SQRT/SQR - square-root</td>
</tr>
<tr>
<td>OLS - ordinary least-squares</td>
<td>SRWR - sum of rank weighted residuals</td>
</tr>
<tr>
<td>P</td>
<td>SS - sum of squares</td>
</tr>
<tr>
<td>PACF - partial autocorrelation function</td>
<td>SSCP - sum of squares and cross products</td>
</tr>
<tr>
<td>PCA - process capability analysis</td>
<td>SYZ/SYD/SYS - SYSTAT data files</td>
</tr>
<tr>
<td>PCF - iterated principal axis factoring</td>
<td>SYO - SYSTAT output files</td>
</tr>
<tr>
<td>pdf - probability density function</td>
<td>TLOSS - Taguchi's Loss Function</td>
</tr>
<tr>
<td>PLS - partial least squares</td>
<td>TOL - tolerance</td>
</tr>
<tr>
<td>pmf - probability mass function</td>
<td>TSLS - Two-Stage Least Squares</td>
</tr>
<tr>
<td>PNG - Portable Network Graphics</td>
<td>TSQ chart - Hotelling's $T^2$ chart</td>
</tr>
<tr>
<td>PVAF/p.v.a.f. -- present value annuity factor</td>
<td>TXT - text format</td>
</tr>
<tr>
<td>p-value - probability value</td>
<td>V</td>
</tr>
<tr>
<td>Q</td>
<td>U chart - chart showing defects per unit</td>
</tr>
<tr>
<td>QC - quality control</td>
<td>UCL - upper control limit</td>
</tr>
<tr>
<td>R</td>
<td>USL - upper specification limit</td>
</tr>
<tr>
<td>R &amp; R - repeatability and reproducibility</td>
<td>UTL - upper tolerance limit</td>
</tr>
<tr>
<td>RAMONA - Reticular Action Model or Near Approximation</td>
<td>V</td>
</tr>
<tr>
<td>ROC - receiver operating characteristic</td>
<td>VAR - variance</td>
</tr>
<tr>
<td>RSE- robust standard errors</td>
<td>VIF - variance inflation factor</td>
</tr>
<tr>
<td>RSM- response surface methods</td>
<td>W</td>
</tr>
<tr>
<td>RTF - rich text format</td>
<td>WMF - Windows metafile</td>
</tr>
<tr>
<td>S</td>
<td>SAV - SPSS files</td>
</tr>
</tbody>
</table>
Acronyms

X
XLS - excel format
X-MR chart - Individuals and moving range chart
XPT/TPT - SAS transport files
XTAB - Crosstabulations

Y

Z
Index

A
ABS function, 93
ACS function, 93
ACSH function, 93
adding the import business object, 16
AND, 89
append files, 30, 34
append cases, 34
commands, 37
examples, 40
ArcView files, 3, 6
ARRAY
    Dummy Subscripted Variables – The ARRAY command, 239
    Temporary Array, 239
ASCII function, 102, 107
ASCII files, 3, 6, 26
error messages, 219
examples, 206, 207, 208, 210, 215, 217, 218, 221, 223, 230
exporting, 221
importing, 8, 204, 213, 216
missing data, 54
troubleshooting, 218
ASN function, 93
ASNH function, 93
AT2 function, 93
ATH function, 93
ATN function, 93
AVG function, 94, 95
data entry, 204
entering triangular matrices, 216
functions, 245
loops, 228, 232
macro, 247
missing values, 210, 211
nested loops, 229
print data and text, 222
subscripted variables, 233, 234, 235
temporary array, 239
variable types, 220
Benford’s Law distribution, 110, 111
Beta distribution, 114
Beta general, 110, 115
Binomial distribution, 110, 112
BOF() function, 97, 252
BOG() function, 97, 252
Business Objects
    adding the import business object, 16
By Groups, 181
by groups
    see stratification
C
calculator, 140
    examples, 159
CALLMACRO, 247, 248
CAP$ function, 102, 103
Case numbers, 86
CASE(), 86
cases
    adding, 60
cutting and pasting, 63
Index

deleting, 63
  ID, 72
  rearranging, 63
CAT$ function, 102, 106
categorical variables, 182
Cauchy distribution, 115
CEIL function, 93
center, 132
character functions, 102
  examples, 149, 150, 151, 152, 154, 155, 156
Chi-square distribution, 115
Cholesky decomposition, 284, 289
CHR$ function, 102
CNT$ function, 102, 103
COD function, 97, 189
COLUMN function, 93
comments
  file comments, 26
comparing values
  examples, 199
  INC function, 190
COMPLETE, 86, 172
complete cases, 86
COMPLETE function, 95
concatenate
  examples, 37, 38, 39, 40
  horizontally, 30, 31
  vertically, 30, 34
concatenating matrix, 281
Converting Numbers from European to American Notation
  examples, 153
copy, 63
Copy as Text, 64
copying data, 64
correlation matrix, 25
COS function, 93
COSH function, 93
covariance matrix, 25
Cumulative distribution functions, 109, 110
cut, 63
  CUT function, 97, 190
cutpoints, 190
  examples, 200
cutting data, 64

D
dat, 3, 6
data
  advanced management of, 203
  entering, 55, 73
  options, 76
  see also data entry, 55
  selection, 57
  transformations, 81, 203
data edit bar, 3
data editing, 56
  adding variables, 60
  changing names, 60
  correcting values, 58
  cutting and pasting, 63
  dates and times, 59
  deleting cases, 63
  dropping variables, 62
  find values, 69
  insert variables, 65
  paste options, 64
  replace values, 69
  right-click options, 66
  undo and redo, 72
data editor, 43, 56
  active cell, 56
  case ID, 72
  navigation, 56
  options, 76
  value editing area, 58
  variable tab, 5
data entry
  examples, 206, 207, 208, 210, 215, 217, 218, 230
  fixed-format, 213, 214
  reading multiple cases per line, 204
  reading multiple lines per case, 204
  triangular matrices, 216
data files
  append cases, 34
  appending, 30, 37
Index

commands, 36
examples, 37, 38, 39, 40
exporting, 26, 27, 37
file comments, 26
importing, 36
merging, 30, 31, 33, 34, 36, 37
opening, 6
rectangular, 44
save options, 25
saving, 24, 36
saving compressed data files, 25
spaces in filenames, 37
spaces in paths, 37
structure, 44
temporary, 28, 29, 36
work files, 29
Data Interchange Format files, 3, 6, 26
dates, 47, 50, 59, 77
dBase files, 3, 6, 26
    importing, 9
DBF, 3, 6
DEFMACRO ENDMACRO, 247, 248
DIAGONAL, 218
DIF, 3, 6
DIM, 233
Discrete uniform distribution, 112
dissimilarity matrix, 25
distribution functions, 109
    examples, 157
Distributions
    univariate continuous, 114
    univariate discrete, 111
DOC function, 101
Double exponential (Laplace) distribution, 116
DOWS function, 101
Dummy Subscripted Variables, 235

E
edit, 45
editor tab
    Variable, 45
eigenvalues, 283
eigenvectors, 288
ELSE statements, 224
EOF() function, 97, 252
EOG() function, 97, 252
Erlang distribution, 116
EVEN function, 94
Excel files, 3, 6, 26
    formulas, 8
    importing, 8
EXP function, 93
Exponential distribution, 117
Exponential notation, 47
exponentiation, 88
export
    examples, 221, 223
export files, 24, 25
    ASCII files, 26, 221
    commands, 37
    Data Interchange Format files, 26
dBase files, 26
    Excel files, 26
    JMP files, 27
    Lotus files, 26
    MINITAB files, 27
    SAS files, 26, 27
    S-PLUS files, 27
    STATA files, 27
    STATISTICA files, 27
extract file, 170

F
F distribution, 117
FDAYM function, 98
FDAYW function, 98
file variable, 222
files
    see data files
FIND, 69
fixed-format, 213, 214
    examples, 215
FLOOR function, 93
FOR NEXT statements, 228
free-format
Index

examples, 206, 207, 208, 210
frequencies, 73
FUNCTION ENDFUNC, 245, 246
functions, 88
  character, 102
date and time, 98
distribution, 109
group, 97
  mathematical, 93
  multivariable, 94

G
 gamma distribution, 118
 Geometric distribution, 112
 Gompertz distribution, 118
 group functions, 97
grouping variables, 54, 181, 190
 Gumbel distribution, 119

H
 Hypergeometric distribution, 113

I
 ICH function, 102, 107
 IF THEN, 223
 import files, 6
  ASCII files, 8
  commands, 36
dBase files, 9
 Excel files, 8
 JMP files, 6
 Lotus files, 8
 MINITAB files, 6
 reading variable names, 7
 SAS files, 9
 SigmaPlot files, 8
 STATATA, 6
 STATISTICA files, 6
 StatView files, 6
 INC function, 97, 190
 IND function, 102, 104
 inputting data, 206

J
 JMP, 3, 6
 JMP files
  importing, 11

K
 Kronecker product, 283

L
 L10 function, 93
 LABS function, 102
 labels, 183
  commands, 192
  examples, 195, 197
  missing codes, 185
 LAG function, 93
 LDAYM function, 98
 LDAYW function, 98
 LDISPLAY, 185
 LEN function, 102
 LFTS function, 102, 103
 LGM function, 93
 list cases, 165, 248
  commands, 171
  examples, 172, 173, 174, 175, 176
  picture format, 166
 LOG function, 93
 Logarithmic series distributions, 113
 Logistic distribution, 120
 Logit normal distribution, 120
 Loglogistic distribution, 121
 Lognormal distribution, 121
 Lotus files, 6, 26

data entry and BASIC, 204
 INT function, 93
 intervals
  CUT function, 190
 Inverse distribution functions, 109
 Inverse Gaussian (Wald) distribution, 119
 ISNUM function, 102

J
 JMP, 3, 6
 JMP files
  importing, 11

K
 Kronecker product, 283
Index

formulas, 8
importing, 8
LOW$ function, 102, 103
LPD$ function, 102, 106

M
matching values
INC function, 190
mathematical functions
examples, 143
matrices
correlation, 76
covariance, 76
dissimilarity, 76
input, 76
similarity, 76
SSCP, 76
types, 216
Matrix
commands, 269
decomposition, 267
examples, 270, 273, 278, 282, 285, 288, 291, 293, 294, 298, 301, 307, 310
generate, 260
operations, 264
organize workspace, 261
overview, 257
read, 258
row/column, 262
merge files, 30, 31
append variables, 31
by key variables, 33
commands, 36
examples, 37, 38, 39
record replication, 34
MID$ function, 102, 106
MINITAB, 3, 6
Minitab files
importing, 10
missing codes, 185
missing values, 54
examples, 210
MOD function, 93
MON function, 98
Moore-Penrose inverse, 310
multicase functions, 96
multivariable functions
examples, 144, 145, 147, 148
multivariate repeated measures layout
MYSTAT files, 3

N
names, 45, 50
NCASE function, 94
NCAT function, 97, 183, 201
Negative binomial distribution, 113
Non-central chi-square distribution, 122
Non-central F distribution, 122
Non-central t distribution, 122
Normal distribution, 121
NOT, 89
NOW$ function, 99
NVAR function, 94

O
ODBC, 11
data sources, 13
databases, 13, 14
SQL, 15
tables, 13
variable selection, 14
ODD function, 94
operators, 88
arithmetic, 88
logical, 89
order of evaluation, 91
relational, 89
options
data, 76
ordering categories, 185
commands, 193

P
p value, 159
Pareto distribution, 123
Index

pasting data, 64
Pearson Type V distribution, 110, 124
Pearson Type VI distribution, 110, 124
Pert distribution, 110, 123
Poisson distribution, 114
PRINT, 223, 250, 251
  Printing Data and Text, 222
  Printing the first three cases, 251
  Saving data as a text without Quotes - PRINT, 223
  Using PRINT to save selected cases, 250
properties, 50, 52
PUTS function, 102, 106

Q
QR decomposition, 289

R
random deviates, 129
random number generator, 130
  Mersenne-Twister, 130
  Wichmann-Hill, 130
Random variate functions, 109
ranks, 131
  commands, 141
Rayleigh distribution, 124
recode, 187
  commands, 193
  examples, 197
recoding variables
  COD function, 189
  examples, 198, 199
rectangular files, 25, 45
repeated measures, 135
reshape data, 135
RETURN, 245, 246
RGT$ function, 102, 103
ROUND function, 93
RPDS function, 102, 106

data files, 9
dates, 27, 28
exporting, 27
importing, 9
numeric variables, 9, 27
PROC COPY, 9
PROC CPORT, 9
string variables, 9, 27
times, 28
transport files, 9
variable names, 27
variable values, 9
SAV, 7
save cases, 248
  examples, 250, 251
saving, 183
  category information, 183
  compressed data files, 25
  data files, 24
  value labels, 185
SD2, 3, 7
select cases, 169, 248
  commands, 171
  examples, 179, 249, 250, 251
selecting cells, 57
SGN function, 93
SHP, 3, 7
SigmaPlot files, 7, 8
significance levels, 129
similarity matrix, 25
SIN function, 93
singular value decomposition, 290
SINH function, 93
Smallest extreme value distribution, 125
SNDS function, 102, 107
sort cases, 167
  commands, 171
  examples, 177, 178
  sort order, 168
split plot designs
S-PLUS, 3
SPSS files, 3, 6
SQL
SELECT statement, 15
SQR function, 93
SQZ$ function, 102, 104
SSCP matrix, 25
stacking variables, 138
  examples, 161
standardized scores, 133
  commands, 141
  examples, 158
STATA, 3
STATISTICA, 3, 6
STATISTICA files
  importing, 11
StatView, 3, 6
STR$ function, 98, 100, 102, 104
stratification, 181, 190, 191
  commands, 192
string variables, 47
structured query language
  case selection, 15
Studentized maximum modulus distribution, 125
Studentized range distribution, 126
SUB$ function, 102, 106
subscripted variables, 233
SUM function, 95
sweep function
  evaluating a quadratic form, 303
  inverting matrices, 302
  multiple correlation coefficient, 307
  solving the system, 304
  stationary distribution for Markov chain, 305
SYS, 3, 6
SYSTAT files, 3, 6
SYZ, 25

T
  t distribution, 126
  TAN function, 93
  templates
    reserved tokens, 29
temporary data files, 28, 29, 36
temporary variables, 220
  creating temporary variables, 220
  printing temporary variables, 222
times, 50, 59, 77
TNH function, 93
transformations, 81
  commands, 141
  conditional, 85
  shortcuts, 87
  unconditional, 83
transport files, 9, 26	ranspose files, 135
  commands, 36
Triangular distribution, 127
triangular matrices, 216
  examples, 217, 218
trimming cases, 134
  examples, 159
TXT, 3, 6
TYPE, 216
types, 50

U
  Uniform distribution, 127
unwrap, 137
UPR$ function, 102, 103
user defined functions, 245

V
  VAL function, 98, 100, 102, 104
variable
  editor, 45
  properties, 45, 50, 52
Variable Types, 220
variables
  adding, 60
  built-in, 86
  changing names, 60
  comments, 54
  cutting and pasting, 63
  default names, 55
  defining, 45, 50, 52
Index

dropping, 62
frequencies, 73
grouping, 54
insert, 65
missing data, 54
reading names, 7
rearranging, 63
subscripted variables, 53

W
Weibull distribution, 128
weighting cases, 139
WHILE...ENDWHILE statements, 232
WK1, 3, 6, 8
WKS, 3, 6, 8
wrap, 137

X
XLS, 3, 6, 8
XPT, 3, 6

Y

Z
z scores, 133
Zipf distribution, 114